PART IV

Additional Information

Symbols

$*$	Conjugate operator
α, β	Constants
$\beta(x, y)$	Beta function
$\delta(\cdot)$	Dirac delta function
Δ_{Ab}	Entropy of $\mathbb{A}$ with respect to $\mathbb{B}$
$\gamma^2(\cdot)$	Frequency dependent coherence or magnitude-squared coherence
$\mathbb{X}$	An information source, including variants (*e.g.* $\mathbb{Y}$,$\mathbb{A}$), in the calculation of entropy
κ	Extensivity or tuning parameter for the Bi-Kappa distribution
ω	Angular frequency in rad/s
$\Psi(X, Y)$	Product of the means of X and Y less their covariance; the *hash* function
σ	Standard deviation
σ^2	Variance
$\|\cdot\|$	Euclidean distance operator
$A + b$	Concatenation of digital sequence, A, and digital sequence, b
$B(t)$	Signal envelope
C, K	Constants

$E\{\cdot\}$	Expectation operator
f	Frequency in Hertz
$F(\cdot)$	Cumulative distribution function
$G(p_1, p_2, p_3, \ldots, p_n)$	Channel transfer function of n parameters
G_a	Gain
$H(\mathbf{X})$	Theoretical entropy of X
$p(\cdot)$, $P(\cdot)$	Probability
$P_a(\cdot)$	Power
$P_{ab}(\cdot)$	Frequency dependent auto- and/or cross- power spectral density
Q	Number of symbols in an alphabet
R_{ab}	Time dependent auto- and/or cross- correlation
S_{T}	Entropic distance
S_{AB}	Relative entropy of $\mathbb{A}$ to $\mathbb{B}$
T	Time period
t	Time
X	Ideal or reference signal (synthetic or real)
Y	Noisy signal (synthetic or real) affected by the propagation medium
Z_a	Impedance
$u(\cdot)$	Unit step function

Glossary

4G	Fourth Generation
ACA	Australian Communications Authority (now ACMA)
ACMA	Australian Communications and Media Authority
AD6645	The partnumber of an ADC made by Analog Devices
ADC	Analog-to-Digital Converter
ADF	Adaptive De-correlation Filtering
AGC	Automatic Gain Control
AM	Amplitude Modulation
ANN	Artificial Neural Network
APRS	Amateur Packet Radio Service
AR	Auto-Regressive
ARM-9401	An HF modem built by BAE Systems
ASK	Amplitude Shift-Keying
AWGN	Additive White Gaussian Noise
bearing	An azimuthal direction
BER	Bit Error Rate
bi-kappa	A type of probability distribution
BPSK	Binary PSK or Bipolar PSK
BW	Bandwidth

Capon	A direction finding algorithm
CCIR	the Comité Consultatif International des Radiocommunications
CDF	Cumulative Distribution Function
CMD	Coherence Median Difference
CMHD	Cross Margenau-Hill Distribution
CNR	Carrier-to-Noise Ratio
coherence	A measure of the similarities of two signals across a bandwidth
CPM	Continuous Phase Modulation
CPU	Central Processing Unit
CTBT	Comprehensive Nuclear Test Ban Treaty
CTD	Capability Technology Demonstrator
CW	Continuous Wave
D, E, or F	Layers of the ionosphere
DAC	Digital-to-Analog Converter
DC	Direct Current, typically representing a frequency of 0 Hz
DDC	Digital Downconverter
DF	Direction Finding or Find
D-region	The lowest ionospheric region, known for its RF energy absorbing properties
DSSS	Direct Sequence Spread Spectrum
entropic	Adjective describing a distance metric based on a measure of entropy

entropy	The information content inherent in a signal
E-region	An ionospheric region above the D-region that refracts RF energy
FAX	Facsimile
FDM	Frequency Division Multiplexing
feature	A unique aspect of a signal
FFT	Fast Fourier Transform
FHSS	Frequency Hopping Spread Spectrum
FM	Frequency Modulation
FPGA	Field-Programmable Gate Array
F-region	An ionospheric region above the E-region that refracts RF energy
FSK	Frequency Shift-Keying
GC4016	A digital-downconvert chip produced by Graychip
GMSK	Gaussian Minimum Shift-Keying
groundwave	An electromagnetic wavefront propagating over the ground
Hamming Distance	A metric that indicates the number of bits different between two binary sequences
hash function	A convenient name for the PEP power estimator function
HD	Hamming Distance
HE011	An active whip antenna made by Rohde & Schwarz
HF	High Frequency
ICA	Independent Component Analysis

ICS554	A digital receiver PCI card made by Integrated Circuit Systems
IEEE	Institution of Electrical and Electronic Engineers
IF	Intermediate Frequency
ionosonde	A system for measuring ionospheric propagation conditions
I-Q	In-phase and Quadrature-phase
ISI	Inter-symbol Interference
ITU	International Telecommunications Union
LMR400	A coaxial cable produced by Times Microwave
LNA	Low Noise Amplifier
LT	Local Time
LZ77	Lempel-Ziv 1977 compression
LZW	Lempel-Ziv-Welch compression
M(3000)F2	The ratio of the MUF over a distance of 3000 km and the vertical critical frequency for the F2 layer
m-**ary**	*m* as prefix: *bin*, *terti*, *quartern*, *etc*
MAW	Mountain Associated Waves
MIL-STD	Military Standard
MIMO	Multiple Input Multiple Output
Modulation Recognition	The process of recognizing the modulation of an unknown signal
MSD	Mean Separation Distance

MUF	Maximum Useable Frequency
multipath	Term for describing the effects of multiple paths of propagation
MUSIC	Multiple Signal Classification
NATO	North Atlantic Treaty Organization
NCO	Numerically Controlled Oscillator
NS	Null Steering or Steer
OFDM	Offset Frequency Division Multiplexing
OOK	Offset Orthogonal Keying
PCI	Peripheral Component Interconnect
PDF	Probability Density Function
PEP	Peak Envelope Power
PM	Phase Modulation
PMC	PCI Mezzanine Card
PSD	Power Spectral Density
PSK	Phase Shift-Keying
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature PSK
RF	Radio Frequency
RG58	A 50 Ω coaxial cable commonly used in RF receiving systems
RM411	A rack-mountable chassis built by Chenbro
RMS	Root Mean Square

S-curve	An S-shaped curve representing the relationship between coherence and SNR
SIR	Signal-to-Interference Ratio
skywave	An electromagnetic wavefront propagating via refractive regions of the ionosphere
SNR	Signal-to-Noise Ratio
Software Radio	The concept that implements in software most modules, if not all, of a radio-communications system
SOI	Signal-of-Interest
SSB	Single Side-Band
SSL	Single Site Location or Single Station Location
STANAG	NATO Standardization Agreement
TDM	Time Division Multiplexing
TID	Traveling Ionospheric Disturbance
UTC	Universal Time Coordinates
V.29, V.32	CCITT V series standards for 9600bps modems
VHF	Very High Frequency
VLF	Very Low Frequency
WOSA	Weighted Overlapped Segment Averaging
Zip 2.3	A compression algorithm

Bibliography

Aisbett, J. (1986), Automatic modulation recognition using time domain parameters, Technical Report ERL-0367-TR, Electronic Research Laboratory DSTO, Department of Defence, Commonwealth Government of Australia.

Aiya, S. V. C. (1962), 'Structure of atmospheric radio noise', *Journal of Scientific and Industrial Research (India)* **2ID**, 203–220.

Akmouche, W. (1999), 'Detection of multicarrier modulations using 4th-order cumulants', *Proc. of IEEE MILCOM 1999 Military Communications Conference* **1**, 432–436. Atlantic City NJ, U.S.A.

BAE Systems (2002), *BAE Systems Adaptive Radio Modem ARM-9401 Series 200 and 210: Installation and Operation Manual*, ard/d/m/001.06 edn, 40-52 Talavera Road, North Ryde NSW 2113, Australia.

Baker, P. W., Clarke, R. H., Massie, A. D. & Taylor, D. (1997), 'Techniques for the measurement and decomposition of the time varying narrow bandwidth transfer function of an HF sky wave transmission', *Radio Science* **32**(5), 1813 – 1820.

Benedetto, D., Caglioti, E. & Loreto, V. (2002), 'Language trees and zipping', *Physical Review Letters* **88**(4). Art. No. 048702.

Bradley, P. A., Damboldt, T. & Suessmann, P. (2000), 'Propagation models for HF radio service planning', **474**, 175 – 179.

Brine, N. L., Lim, C. C., Massie, A. D. & Marwood, W. (2002), 'A pre-emptive null-steering ionosonde', *Proc. of the Fourth Symposium of Radiolocations and Direction Finding* . San Antonio TX, U.S.A.

Budden, K. G. (1988), *The Propagation of Radio Waves : The Theory of Radio Waves of Low Power in the Ionosphere and Magnetosphere*, Cambridge University Press, Cambridge, U.K.

Cao, L. J., Chua, K. S., Chong, W. K., Lee, H. P. & Gu, Q. M. (2003), 'A comparison of PCA, KPCA and ICA for dimensionality reduction in support vector machine', *Neurocomputing* **55**, 321–336.

Carter, G. C. (1993), *Coherence and Time Delay Estimation: An Applied Tutorial for Research, Development, Test, and Evaluation Engineers*, IEEE Press, Piscataway NJ, U.S.A.

CCIR (1986), Characteristics and applications of atmospheric radio noise data, Technical Report 322-3, International Telecommunication Union, Geneva.

Chan, D. C. B., Rayner, P. J. W. & Godsill, S. J. (1996), 'Multi-channel signal separation', *Proceedings of the IEEE 1996 International Conference on Acoustics, Speech, and Signal Processing* **2**, 649 – 652. Atlanta GA, U.S.A.

Choi, E. & Lee, C. (2003), 'Feature extraction based on the Bhattacharyya distance', *Pattern Recognition* **36**, 1703–1709.

Choi, I. S. & Kim, H. T. (2000), 'Efficient feature extraction from time-frequency analysis of transient response for target identification', *Microwave and Optical Technology Letters* **26**(6), 403–407.

Clarke, C. (1962), 'Atmospheric radio-noise studies based on amplitude-probability measurements at Slough, England during the International Astrophysical Year', *Proceedings of the IEE (London)* **109**(B), 393–404.

Coleman, C. J. (2006), 'Noise and the choice of antenna in HF communications', *Proceedings of the 10th IET International Conference on Ionospheric Radio Systems and Techniques (IRST2006)* pp. 321–325. London, U.K.

Couch II, L. W. (1990), *Digital and Analog Communication Systems*, 3 edn, Macmillan Publishing Co., New York NY, U.S.A.

Craig, J. (2001), 'Marconi's first transatlantic wireless experiment', *The Canadian Amateur* **29**(6).

Davies, K. (1990), *Ionospheric Radio*, IEE Electromagnetic Waves Series 31, Peter Peregrinus Ltd., London, U.K.

de Moivre, A. (1756), *The Doctrine of Chances*, 3 edn, A. Millar, London, U.K.

Demeure, C. J. & Laurent, P. A. (1997), 'A new modem for high quality sound broadcasting at short waves', *IEE Seventh International Conference on HF Radio Systems and Techniques* **441**, 50 – 54. London, U.K.

Distante, C., Leo, M., Siciliano, P. & Persaud, K. C. (2002), 'On the study of feature extraction methods for an electronic nose', *Sensors and Actuators B* **87**, 274–288.

Erten, G. & Salam, F. M. (1999), 'Voice extraction by on-line signal separation and recovery', *IEEE Transactions on Circuits and Systems - II: Analog and Digital Signal Processing* **46**(7).

Fabrizio, G. A., Abramovich, Y. I., Anderson, S. J., Gray, D. A. & Turley, M. D. (1998), 'Adaptive cancellation of nonstationary interference in HF antenna arrays', *IEE Proceedings on Radar and Sonar Navigation* **145**(1), 19 – 24.

Ferguson, B. (2001), 'Wavelet de-noising of optical terahertz pulse imaging data', *Fluctuation and Noise Letters* **1**(2), L65 – L69.

Fiala, E. R. & Greene, D. H. (1989), 'Data compression with finite windows', *Communications of the ACM* **32**(4), 490–505.

Foldes, G. (1960), 'The lognormal distribution and its applications to atmospheric studies', *Statistical Methods in Radio Wave Propagation* pp. 227–232.

Foschini, G. J. (1996), 'Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas', *Bell Labs Technical Journal* **Autumn 1996**, 41 – 59. Lucent Technologies Inc.

Foschini, G. J. & Gans, M. J. (1998), 'On limits of wireless communications in a fading environment when using multiple antennas', *Wireless Personal Communications* **6**, 311 – 335.

Fragoulis, D., Rousopoulos, G., Panagopoulos, T., Alexiou, C. & Papaodysseus, C. (2001), 'On the automated recognition of seriously distorted musical recordings', *IEEE Transactions on Signal Processing* **49**(4), 898–908.

Furman, W. N. (1997), 'Robust low bit rate HF data modems', *IEE Seventh International Conference on HF Radio Systems and Techniques* **441**, 149 – 153. London, U.K.

Furutsu, K. & Ishida, T. (1961), 'On the theory of amplitude distribution of impulsive random noise', *Journal of Applied Physics* **32**, 1206–1221.

Gething, P. J. D. (1991), *Radio Direction Finding and Super-resolution*, IEE Electromagnetic Waves Series 33, 2nd edn, Peter Peregrinus Ltd, London, U.K.

Giesbrecht, J., Clarke, R. & Abbott, D. (2006), 'An empirical study of the probability density function of HF noise', *Fluctuations and Noise Letters* **6**(2), L117–L125.

Goodwin, G. L., Jeffrey, Z. R. & Hichens, D. J. (1992), 'A high frequency radio location system', *Journal of Electrical and Electronics Engineering, Australia* **12**(3), 284 – 289.

Goris, M. (1998), Categories of radio-frequency interference, Technical report, National Foundation for Radio Astronomy, Holland.

Groller, R. (1990), 'Single station location HF direction finding', *Journal of Electronic Defense* **13**(6), 58 – 63, 83.

Gubbay, J. S. & Lynn, K. J. W. (1988), Scientific investigations of the Space Research Group, Technical report, Defence Science and Technology Organisation.

Ham, F. M. & Faour, N. A. (1999), 'Infrasound signal separation using independent component analysis', *21st Seismic Research Symposium* pp. 133–140. Las Vegas NV, U.S.A.

Harris, F. J. (1978), 'On the use of windows for harmonic analysis with the discrete fourier transform', *Proceedings of the IEEE* **66**(1), 51–83.

Hero III, A. O. & Hadinejad-Mahram, H. (1998), 'Digital modulation classification using power moment matrices', *Proc. of the IEEE 1998 International Conference on Acoustics, Speech, and Signal Processing* **6**, 3285–3288. Seattle WA, U.S.A.

Hippenstiel, R. D. & De Oliveira, P. M. (1990), 'Time-varying spectral estimation using the instantaneous power spectrum (IPS)', *IEEE Transactions on Signal Processing* **38**, 1752–1759.

Hoff, R. S. & Johnson, R. C. (1952), 'A statistical approach to the measurement of atmospheric noise', *Proceedings of the IRE* **40**, 185–187.

Hsue, S.-Z. & Soliman, S. S. (1990), 'Automatic modulation classification using zero crossing', *IEE Proceedings* **137**(6), 459–464.

Ibukun, O. (1964), 'Measurements of atmospheric noise levels', *Radio and Electronics Engineering* **28**, 405–415.

Ibukun, O. (1966), 'Structural aspects of atmospheric radio noise in the tropics', *Proceedings of the IEEE* **54**(3), 361–367.

IPS Radio and Space Services (1994), *HF Radio Propagation Course Manual*, Australian Government Department of Industry Tourism and Resources.

Jang, G.-J. & Lee, T.-W. (2002), 'A probabilistic approach to single channel blind signal separation', *Proceedings of the 15th Conference on Neural Information Processing Systems: Natural and Synthetic 2002 (NIPS2002)* pp. 1173–1180. Whistler BC, Canada.

Johnson, E. E., Desourdis Jr., R. I., Earle, G. D., Cook, S. C. & Ostergaard, J. C. (1997), *Advanced High-Frequency Radio Communications*, Artech House Inc., Norwood MA, USA.

Ketterer, H., Jondral, F. & Costa, A. H. (1999), 'Classification of modulation models using time-frequency methods', *Proceedings of the IEEE 1999 International Conference on Acoustics, Speech, and Signal Processing* **5**, 2471–2474. Phoenix AZ, U.S.A.

Kraus, J. D. (1966), *Radio Astronomy*, McGraw-Hill, New York NY, U.S.A.

Kuo, B.-C. & Landgrebe, D. A. (2004), 'Nonparametric weighted feature extraction for classification', *IEEE Transactions on Geoscience and Remote Sensing* **42**(5), 1096–1105.

Landgrebe, D. A. (1997), On information extraction principles for hyperspectral data, Technical report, School of Electrical and Computer Engineering, Purdue University, West Lafayette IN, U.S.A.

Laplace, P.-S. (1820), *Théorie Analytique des Probabilités*, 3 edn, Courcier. Paris.

Lemmon, J. J. & Behm, C. J. (1991), Wideband HF noise/interference modeling Part I: First-order statistics, Technical Report 91-277, U.S. Department of Commerce, National Telecommunications and Information Administration (NTIA).

Lemmon, J. J. & Behm, C. J. (1993), Wideband HF noise/interference modeling Part II: Higher-order statistics, Technical Report 93-293, U.S. Department of Commerce, National Telecommunications and Information Administration (NTIA).

Lempel, A. & Ziv, J. (1977), 'A universal algorithm for sequential data compression', *IEEE Transactions on Information Theory* **23**(3), 337–343.

Leone, A., Distante, C., Ancona, N., Persaud, K. C., Stella, E. & Siciliano, P. (2005), 'A powerful method for feature extraction and compression of electronic nose responses', *Sensors and Actuators B* **105**, 378–392.

Leubner, M. P. & Vörös, Z. (2005), 'A nonextensive entropy path to probability distributions in solar wind turbulence', *Nonlinear Processes in Geophysics* **12**, 171–180.

Likhter Ya, I. (1956), 'Some statistical properties of atmospherics', *Radiotekhnika i Elektronika* **1**, 1295–1304.

Lyapunov, A. M. (1901), 'Nouvelle forme du théorème sur la limite de probabilité', *Mém. Acad. Sci. St. Petersburg* **12**(5), 1–24.

Ma, J., Theiler, J. & Perkins, S. (2004), 'Two realizations of a general feature extraction framework', *Pattern Recognition* **37**, 875–887.

Mallat, S. (1999), *A Wavelet Tour of Signal Processing*, 2 edn, Academic Press, San Diego CA, U.S.A.

Marconi, G. (1909), 'Wireless telegraphic communication', *Nobel Lectures in Physics: 1901-1921, World Scientific (1998)* pp. 196–225.

Maslin, N. M. (1987), *HF Communications: A Systems Approach*, Plenum Press, New York NY, U.S.A.

McDonough, R. N. & Whalen, A. D. (1995), *Detection of Signals in Noise*, 2 edn, Academic Press, San Diego CA, U.S.A. pp. 129–130.

McNamara, L. F. (1988), 'Ionospheric modelling in support of single station location of long range transmitters', *Journal of Atmospheric and Terrestrial Physics* **50**(9), 781 – 795.

Mitchell, R. A. & Westerkamp, J. J. (1999), 'Robust statistical feature based aircraft identification', *IEEE Transactions on Aerospace and Electronic Systems* **35**(3), 1077–1093.

Nakai, T. (1960), 'The study of amplitude probability distribution of atmospheric radio noise', *Proceedings of the Research Institute of Atmospherics (Japan)* **7**, 12–17.

Nandi, A. K. & Azzouz, E. E. (1997), 'Modulation recognition using artificial neural networks', *Signal Processing* **56**, 165–175.

Nandi, A. K. & Azzouz, E. E. (1998), 'Algorithms for automatic modulation recognition of communication signals', *IEEE Transactions on Communications* **46**(4), 431–435.

NATO (1989), Stanag 4285 Ed. 1 Standardization Agreement, Technical report, North Atlantic Treaty Organization.

Noble, A. P., Spicer, J. A., Midwinter, M. P. & Farquhar, S. G. (1997), 'The viability of very high data rate modems for a specific HF channel type', *IEE Seventh International Conference on HF Radio Systems and Techniques* **441**, 55 – 59. London, U.K.

Nolan, K. E., Doyle, L., Mackenzie, P. & O'Mahony, D. (2002), 'Modulation scheme classification for 4G software radio wireless networks', *Proc. of the Second IASTED International Conference on Signal Processing, Pattern Recognition, and Applications (SPPRA 2002)* pp. 25–31. Crete, Greece.

Nuttall, A. H. (1958), Theory and application of the separable class of random processes, Technical Report 343, Massachusetts Institute of Electronics.

Park, H.-M., Jung, H.-Y., Lee, T.-W. & Lee, S.-Y. (1999), 'Subband-based blind signal separation for noise speech recognition', *Electronic Letters* **35**(23), 2011–2022.

Proakis, J. G. (1989), *Digital Communications*, Communications and Signal Processing, 2 edn, McGraw-Hill Book Company, New York NY, U.S.A.

Reed, J. H. (2002), *Software Radio: A Modern Approach to Radio Engineering*, Prentice Hall Communications Engineering and Emerging Technologies Series, 1 edn, Prentice Hall PTR, Upper Saddle River NJ, USA.

Sedgewick, R. (1988), *Algorithms*, 2 edn, Addison-Wesley, Reading MA, U.S.A.

Sevgi, L. & Ponsford, A. M. (1999), An HF radar based integrated maritime surveillance system, Technical report, Raytheon Systems Canada Ltd.

Shannon, C. E. (1948), 'A mathematical theory of communication', *Bell System Technical Journal* **27**, 379–423 (July); 623–656 (October).

Smith, J. (1986), *Modern Communication Circuits*, McGraw-Hill Series in Electrical Engineering, 1 edn, McGraw-Hill Inc., New York NY, USA. pp. 1-13.

Smith, O. J., Angling, M. J., Cannon, P. S., Jodalen, V., Jacobsen, B. & Gronnerud, O.-K. (2001), 'Simultaneous measurement of propagation characteristics on non-contiguous HF channels', *Proceedings of ICAP Eleventh International Conference on Antennas and Propagation* **1**, 83 – 87. Manchester, U.K.

Stanacevic, M., Cauwenberghs, G. & Zweig, G. (2002), 'Gradient flow adaptive beamforming and signal separation in a miniature microphone array', *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP2002)* . Orlando FL, U.S.A.

Sun, Z., Bebis, G. & Miller, R. (2004), 'Object detection using feature subset selection', *Pattern Recognition* **37**, 2165–2176.

Tan, H. C., Sakaguchi, K., Takada, J.-I. & Araki, K. (2002), DOA based signal combining aided automatic modulation recognition/demodulation for surveillance system, Technical report, The Institute of Electronics, Information and Communications Engineers (IEICE).

Tang, Y. Y., Tao, Y. & Lam, E. C. M. (2002), 'New method for feature extraction based on fractal behavior', *Pattern Recognition* **35**, 1071–1081.

Treharne, R. F. (1981), 'Non-military applications of high frequency single station location system', *Journal of Electrical and Electronics Engineering, Australia* **1**(1), 87 – 92.

U.S. Dept. of Defense (1991), Mil-Std-188-110A Military Standard 188—interoperability and performance standards for data modems, Technical report.

Vogler, L. E. & Hoffmeyer, J. A. (1988), A new approach to HF channel modeling and simulation Part I: Deterministic model, Technical Report 88-240, U.S. Department of Commerce, National Telecommunications and Information Administration (NTIA).

Vogler, L. E. & Hoffmeyer, J. A. (1990), A new approach to HF channel modeling and simulation Part II: Stochastic model, Technical Report 90-255, U.S. Department of Commerce, National Telecommunications and Information Administration (NTIA).

Vogler, L. E. & Hoffmeyer, J. A. (1992), A new approach to HF channel modeling and simulation Part III: Transfer function, Technical Report 92-284, U.S. Department of Commerce, National Telecommunications and Information Administration (NTIA).

Waller, J. R. & Brushe, G. D. (1999), 'A method for differentiation between frequency and phase modulated signals', *Proceedings of IEEE Information, Decision, and Control 1999* pp. 489–494.

Wang, X. & Paliwal, K. K. (2003), 'Feature extraction and dimensionality reduction algorithms and their applications in vowel recognition', *Pattern Recognition* **36**, 2429–2439.

Watterson, C. C. & Coon, R. M. (1969), Recommended specifications for ionospheric noise simulators, Technical Report ERL-127-ITS-89, Environmental Science Services Administration (ESSA).

Watterson, C. C., Juroshek, J. R. & Bensema, W. D. (1970), 'Experimental confirmation of an HF channel model', *IEEE Transactions on Communication Technology* **COM-18**, 792–803.

Welch, P. D. (1967), 'The use of the Fast Fourier Transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms', *IEEE Transactions on Audio and Electroacoustics* **AU-15**, 70–73.

Welch, T. A. (1984), 'A technique for high-performance data compression', *IEEE Computer* **17**(6), 8–19.

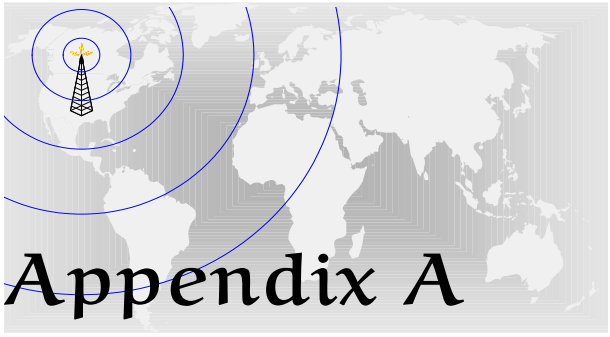
Whalen, A. D. (1971), *Detection of Signals in Noise*, Academic Press, San Diego CA, U.S.A. pp. 105.

Wolniansky, P. W., Foschini, G. J., Golden, G. D. & Valenzuela, R. A. (1998), 'V-BLAST: An architecture for realizing very high data rates over the rich-scattering wireless channel', *Proceedings of the 1998 International Symposium on Signals, Systems, and Electronics* pp. 295–300. Pisa, Italy.

Wong, M. L. D. & Nandi, A. K. (2001), 'Automatic digital modulation recognition using spectral and statistical features with multi-layer perceptrons', *Proc. of the IEEE International Symposium on Signal Processing and its Applications* **2**, 390–393. Kuala-Lampur, Malaysia.

Yen, K.-C. & Zhao, Y. (1996), 'Robust automatic speech recognition using a multi-channel signal separation front-end', *Proceedings of the IEEE 4th International Conference on Spoken Language Processing* **3**, 1337 – 1340. Philadelphia PA, U.S.A.

Yuhara, H., Ishida, T. & Higashimura, M. (1956), 'Measurement of the amplitude probability distribution of atmospheric noise', *Journal of Radio Research Laboratories (Japan)* **3**, 101–108.



Appendix A

Mathematical Derivations and Examples

DERIVATIONS of mathematical formulae and calculation examples are included in this appendix. The derivations address overlapping segments for the WOSA method and the Bi-Kappa distribution for the HF noise model. Various examples illustrate the difficulties in calculating a closed-form solution for the coherence function. The examples also demonstrate that coherence is not, by itself, a useful feature for modulation recognition.

A.1 Derivation of the Modified Bi-Kappa Distribution

Section 7.5 notes that the natural HF noise is similar to a Bi-Kappa distribution. The Bi-Kappa distribution is defined by Leubner and Vörös(2005) and is a sum of two components: a *halo* distribution and a *core* distribution. They also allow $-\infty < \kappa < +\infty$ for reasons necessary for their work. However, for this discussion, it is the shape of the Bi-Kappa distribution that is important as it approximates the observed results in Chapter 7. The desired shape can be obtained from either the *halo* or *core* components. For simplicity the *halo* component is chosen which necessarily limits κ to values greater than or equal to zero. Therefore, a modified Bi-Kappa distribution is defined as

$$p(X;\kappa) = C\left[1+\frac{X^2}{\kappa\sigma^2}\right]^{-\kappa} \quad \kappa >= 0 \tag{A.1}$$

where C is a normalization factor. In the limit, as $\kappa \to 0$, the distribution converges to a uniform distribution of probability C, and as $\kappa \to \infty$ the distribution converges on a Gaussian distribution. The proof follows.

For now, assume $C = 1$ and let $y(x;\kappa \to m) = p(X;\kappa)$ for the cases where $m \in \{0,\infty\}$ then

$$\lim_{\kappa\to m} y(x;\kappa) = \lim_{\kappa\to m}\left(1+\frac{x^2}{\kappa\sigma^2}\right)^{-\kappa} \tag{A.2}$$

$$\lim_{\kappa\to m} \ln y(x;\kappa) = \lim_{\kappa\to m} -\kappa\ln\left(1+\frac{x^2}{\kappa\sigma^2}\right) \tag{A.3}$$

$$\lim_{\kappa\to m} \ln y(x;\kappa) = \lim_{\kappa\to m} \frac{-\ln\left(1+\frac{x^2}{\kappa\sigma^2}\right)}{\frac{1}{\kappa}}. \tag{A.4}$$

With the limit in an indeterminate form (*i.e.* $\frac{\infty}{\infty}$ or $\frac{0}{0}$) L'Hôpital's rule can be used to further simplify to

$$\lim_{\kappa\to m} \ln y(x;\kappa) = \lim_{\kappa\to m} \frac{-x^2}{\sigma^2}\left(\frac{\kappa\sigma^2}{\kappa\sigma^2+x^2}\right) \tag{A.5}$$

$$y(x;m) = e^{w(x;m)}, \tag{A.6}$$

$$\text{where } w(x;m) = \frac{-x^2}{\sigma^2}\left(\frac{m\sigma^2}{m\sigma^2+x^2}\right).$$

So for $m = 0$ the distribution becomes uniform with $y(0) = 1$, and for $m = \infty$, the distribution converges on a Gaussian of $y(x;\infty) = e^{\frac{-x^2}{\sigma^2}}$.

To find C one must first solve

$$C \int_{-\infty}^{\infty} \left[1 + \frac{X^2}{\kappa\sigma^2}\right]^{-\kappa} \partial X = 1, \tag{A.7}$$

by letting $\sigma\sqrt{\kappa}\tan\theta = X$, and differentiating X to find $\partial X = \sigma\sqrt{\kappa}\sec^2\theta\partial\theta$. Substituting for X and ∂X, and changing the limits of integration yields

$$C\sigma\sqrt{\kappa} \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \left[1 + \tan^2\theta\right]^{-\kappa} \sec^2\theta\partial\theta = 1. \tag{A.8}$$

Realizing that $1 + \tan^2\theta = \sec^2\theta$, Eq. (A.8) becomes

$$C\sigma\sqrt{\kappa} \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \left[\sec^2\theta\right]^{-\kappa} \sec^2\theta\partial\theta = 1, \tag{A.9}$$

$$C\sigma\sqrt{\kappa} \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \cos^{2\kappa-2}\theta\partial\theta = 1, \tag{A.10}$$

$$C\sigma\sqrt{\kappa}2 \int_{0}^{\frac{\pi}{2}} \cos^{2\kappa-2}\theta\partial\theta = 1. \tag{A.11}$$

The final form of Eq. (A.11) is related to a common form of the beta function where

$$\beta(x,y) = 2\int_{0}^{\frac{\pi}{2}} \sin^{2x-1}\theta\cos^{2y-1}\theta\partial\theta. \tag{A.12}$$

Setting $x = \frac{1}{2}$ and $y = \kappa - \frac{1}{2}$ provides

$$\beta\left(\frac{1}{2}, \kappa - \frac{1}{2}\right) = 2\int_{0}^{\frac{\pi}{2}} \cos^{2\kappa-2}\theta\partial\theta, \tag{A.13}$$

which is suitable to replace the integral in Eq. (A.11). Hence,

$$C\sigma\sqrt{\kappa}\beta\left(\frac{1}{2},\kappa-\frac{1}{2}\right) = 1, \text{ and} \tag{A.14}$$

$$C = \frac{1}{\sigma\sqrt{\kappa}\beta(\frac{1}{2},\kappa-\frac{1}{2})}\ \forall\kappa>\frac{1}{2}. \tag{A.15}$$

This definition of C necessarily limits $\kappa > \frac{1}{2}$, since $\beta(\frac{1}{2},\kappa-\frac{1}{2})$ is undefined for $\kappa \leq 0$ and is infinite for $0 < \kappa \leq \frac{1}{2}$. There is, therefore, no justification for the uniform distribution at $p(x;0)$.

In the context of HF noise, a Gaussian distribution at $\kappa = \infty$ is apparent in thermal noise, galactic noise, and man-made noise. Conversely, a uniform distribution (or uncoloured white noise), at $\kappa = 0$, is not common unless the uniform distribution approximates another distribution over a narrow interval. The restriction on κ therefore avoids the conundrum of a physical meaning for the distribution at $\kappa = 0$.

Finally, the complete modified Bi-Kappa distribution is

$$p(X;\kappa) = \frac{1}{\sigma\sqrt{\kappa}\beta(\frac{1}{2},\kappa-\frac{1}{2})}\left(1+\frac{X^2}{\kappa\sigma^2}\right)^{-\kappa} \quad \kappa>\tfrac{1}{2}, -\infty<X<+\infty. \tag{A.16}$$

Equation Eq. (A.16) is fine if $-\infty < X < +\infty$, but what if $a < X < b$ where $|a| \ll \infty$ and $|b| \ll \infty$? For a finite range of X, the normalization constant, C, must be

$$C = \frac{1}{F(b)-F(a)}, \tag{A.17}$$

where $F(z)$ is the cumulative distribution function of Eq. (A.16) evaluated at $z \in \mathbb{R}$. An easy method to determine $F(b) - F(a)$ is to calculate the area under the probability curve in the range of interest.

The solution for C can be numerically confirmed. Table A.1 and Figure A.1 show that C is an effective normalization constant for $X \in [-R, R]$ where $R \gg 0$. The values of κ and σ affect the rate at which the product of C and the area under the distribution approaches one. For low κ the convergence is slow and therefore the range of X must be large in order for C to accurately normalize the distribution. For high κ the convergence is relatively fast and therefore the range of X necessary for convergence is

Table A.1. Normalizing coefficient for the modified Bi-Kappa distribution. The correctness of the normalizing coefficient, C, can be demonstrated by computing the Riemann sum over varying ranges of X. The table below shows that as the range of X approaches the inclusive set $[-\infty, \infty]$, the product of the normalization constant and the Riemann sum approaches one. For a low range of X, the normalization is inaccurate. In these instances, a more effective normalization is described by Eq. (A.17). The choice of κ and σ is arbitrary and serves only to provide an indication of their effects on the convergence of the product. Figure A.1 further demonstrates this.

κ	σ	$\mathbf{C}$	$\mathbf{X}$	$\Delta\mathbf{X}$	$\mathbf{C}\sum_{n=0}^{N-1}\left(1+\frac{\mathbf{X_n}^2}{\kappa\sigma^2}\right)^{-\kappa}\Delta\mathbf{X}$
0.6	1.0	0.1140143	$[-10, 10]$	0.01	0.4707447
			$[-100, 100]$	0.01	0.6659245
			$[-1000, 1000]$	0.01	0.7892100
			$[-10000, 10000]$	0.01	0.8670004
			$[-100000, 100000]$	0.1	0.9160829
			$[-1000000, 1000000]$	1.0	0.9490283
1.2	1.0	0.3643038	$[-10, 10]$	0.01	0.9743834
			$[-100, 100]$	0.01	0.9989735
			$[-1000, 1000]$	0.01	0.9999591
			$[-10000, 10000]$	0.01	0.9999983
1.2	3.2	0.1138449	$[-10, 10]$	0.01	0.8760388
			$[-100, 100]$	0.01	0.9947724
			$[-1000, 1000]$	0.01	0.9997917
			$[-10000, 10000]$	0.01	0.9999917

smaller. For cases with low κ, a better normalization constant is defined by Eq. (A.17). Finally, the rate of convergence has a weak dependence on σ (see Table A.1). The rate of convergence slows as σ increases, but, the dominant parameter controlling convergence is κ.

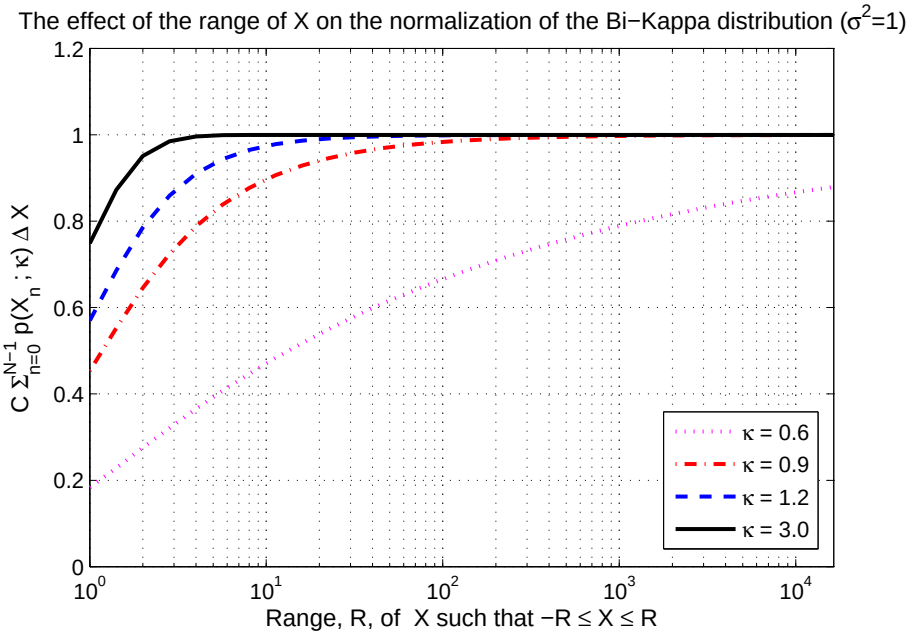


Figure A.1. The effect of range and κ on normalization. The range of the physical variable, X, and the tuning-parameter, κ, affects the rate at which the product of the Riemann sum of the probability curve and the normalization coefficient, C, approaches one. The constant C accurately normalizes the distribution as the range of X increases towards ∞. However, the normalization is inaccurate for a low range of X. The effect is more pronounced when κ is also low. In these instances, a more effective normalization is described by Eq. (A.17). The choice of κ and σ is arbitrary and serves only to provide an indication of their effects on the convergence of the product. Table A.1 further demonstrates this.

A.2 Mathematics of Overlapping Segments

Section 10.2 introduces the coherence function and, among other topics, discusses the effects of the overlapping in the coherence estimation via Welch's (1967) method. Consider a vector of samples with overlapping segments (see Figure A.2). In such a sequence, there are R segments of J samples each overlapped by δ samples. The equation governing this relationship can be written by inspection.

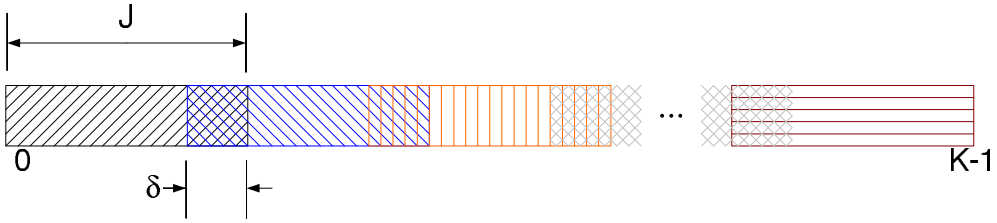


Figure A.2. A vector of overlapping segments. A vector of K samples with R segments of J samples each overlapped by δ samples.

For each segment there is one overlapping portion except for the last segment. Hence one writes,

$$K = RJ - (R-1)\delta \tag{A.18}$$

$$R = \frac{K-\delta}{J-\delta} \tag{A.19}$$

$$J = \frac{K+(R-1)\delta}{R}. \tag{A.20}$$

If δ is now expressed as a fraction of J such that $\delta = \alpha J$ where $0 \leq \alpha \leq 1$, then substituting for δ yields

$$K = J(R - R\alpha + \alpha) \tag{A.21}$$

$$R = \frac{K-\alpha J}{J(1-\alpha)} \tag{A.22}$$

$$J = \frac{K}{R - R\alpha + \alpha}. \tag{A.23}$$

Now suppose that the data in the vector is the result of a sampling process (*e.g.* a digital receiver) where the sampling rate is f_s over a time duration T, so that $K = f_s T$. The advanced broadband receiver, discussed in Chapters 6 and 11, utilizes the Texas

Instruments GC4016 digital downconverter (DDC). Details of this DDC can be found in Appendix E. The output of this DDC relies on Eq. (A.20). That is, the output complex sampling rate is

$$f_s = \frac{f_{clk}}{4N}\frac{P}{Q}, \tag{A.24}$$

where N is the decimation factor for the cascade-integrate-comb (CIC) filter of the DDC, f_{clk} is the DDC clock frequency in Hertz, P is the upsampling factor and Q is the downsampling factor of the DDC. Moreover, f_s is also related to the desired complex output signal bandwidth of the DDC, f_B, by

$$f_B = 2\nu f_s, \tag{A.25}$$

where $0 \leq \nu \leq 0.5$ is the DDC filter cutoff factor. If $\nu = 0.5$ the complex signal bandwidth equals the complex output sampling rate, f_s. If $\nu < 0.5$ the signal bandwidth is less than the output sampling rate. Combining Eq. (A.25), Eq. (A.24), and Eq. (A.20) yields

$$R = \frac{f_B}{2\nu}\frac{T}{J(1-\alpha)} + \frac{\alpha}{1-\alpha}. \tag{A.26}$$

The parameter, α, represents the percentage overlap and is the fraction $\frac{\delta}{J}$.

Constraints of the GC4016 dictate that N must be an integer and, by the nature of the discrete processing, so must K, J, R, and δ. Consequently, Eq. (A.26) is used to determine an acceptable value for N, J, and R given a desired overlap fraction, α, and signal bandwidth, f_B.

Example: Suppose 1 MHz $\leq f_B \leq$ 1.5 MHz. What is the acceptable CIC decimation factor for $\nu = 0.5$, $T = 1$ s, $\frac{P}{Q} = 1$, and $f_{clk} = 100$ MHz?
From Eq. (A.24) and Eq. (A.25), $\frac{25\times10^6}{N} = \frac{f_B}{2(0.5)}$ therefore $N = \frac{25\times10^6}{f_B}$. For conditions where $100 \leq J \leq 200$ and $0.3 \leq \frac{\delta}{J} \leq 0.7$ some cases that satisfy the requirements for integer values of N, K, R, δ and J are listed below.
Case 1: $f_B = 1$ MHz, $N = 25$, $\frac{\delta}{J} = 0.4$, $J = 100$, $R = 16666$
Case 2: $f_B = 1$ MHz, $N = 25$, $\frac{\delta}{J} = 0.5$, $J = 128$, $R = 15624$
Case 3: $f_B = 1$ MHz, $N = 25$, $\frac{\delta}{J} = 0.4$, $J = 160$, $R = 10416$
Case 4: $f_B = 1.25$ MHz, $N = 20$, $\frac{\delta}{J} = 0.4$, $J = 125$, $R = 16666$
Case 5: $f_B = 1.25$ MHz, $N = 20$, $\frac{\delta}{J} = 0.5$, $J = 160$, $R = 15624$
Case 6: $f_B = 1.25$ MHz, $N = 20$, $\frac{\delta}{J} = 0.4$, $J = 200$, $R = 10416$

A.3 Coherence Calculation Examples

Section 10.2 suggests that the coherence function can be difficult to compute. This section of the appendix contains example calculations that are trivial and intractable. Recall that coherence is defined by

$$\gamma^2(f) = \left| \frac{P_{xy}(f)}{\sqrt{P_{xx}(f)P_{yy}(f)}} \right|^2, \text{ or} \tag{A.27}$$

$$\gamma^2(f) = \left| \frac{P_{xy}(f)P^*_{xy}(f)}{P_{xx}(f)P_{yy}(f)} \right|, \tag{A.28}$$

where $P_{xy}(f)$ is the cross-power spectral density of signals X and Y. The auto-power spectral densities of signals X and Y are identified by $P_{xx}(f)$ and $P_{yy}(f)$ respectively. The term, $\gamma^2(f)$, is sometimes called the magnitude-squared coherence but, for this work, $\gamma^2(f)$ is simply referred to as coherence. The following examples demonstrate the coherence of some common signals.

Coherence of Two Pulses

Assume that the time-domain representation of X and Y are two pulses defined by

$$x(t) = A\left[u(t) - u(t - T_x)\right], \text{and} \tag{A.29}$$

$$y(t) = B\left[u(t - \delta) - u(t - T_y - \delta)\right], \tag{A.30}$$

where $u(t)$ is the unit step function, t is time, A and B are constants, δ is the time delay between pulses, and T_x and T_y are the pulse widths of $x\,(t)$ and $y\,(t)$ respectively. Using the correlation integral

$$R_{ab}(\tau) = \int_{-\infty}^{\infty} a(t)b(t+\tau)\rho_{ab}(t)\partial t, \tag{A.31}$$

where τ is time lag and t is time, and assuming that the statistics of $x(t)$ and $y(t)$ are jointly and individually stationary such that the joint probability, $\rho_{xy}(\tau) = 1$, the auto-correlations and cross-correlations of $x\,(t)$ and $y\,(t)$ are

$$R_{xx}(\tau) = \begin{cases} A^2(T_x - \tau) & 0 \le \tau < T_x \\ A^2(T_x + \tau) & -T_x \le \tau < 0 \\ 0 & \text{otherwise} \end{cases}, \tag{A.32}$$

$$R_{yy}(\tau) = \begin{cases} B^2(T_y - \tau) & 0 \le \tau < T_y \\ B^2(T_y + \tau) & -T_y \le \tau < 0 \\ 0 & \text{otherwise} \end{cases}, \tag{A.33}$$

$$R_{xy}(\tau) = \begin{cases} AB(T_y + \delta + \tau) & -\delta - T_y \le \tau < -\delta \\ ABT_y & -\delta \le \tau < -\delta + T_x - T_y \\ AB(T_x - \delta - \tau) & -\delta + T_x - T_y \le \tau < -\delta + T_x \\ 0 & \text{otherwise} \end{cases}, \tag{A.34}$$

$$R_{yx}(\tau) = \begin{cases} AB(T_x - \delta + \tau) & \delta - T_x \le \tau < \delta - T_x + T_y \\ ABT_y & \delta - T_x + T_y \le \tau < \delta \\ AB(T_y + \delta - \tau) & \delta \le \tau < \delta + T_y \\ 0 & \text{otherwise} \end{cases}. \tag{A.35}$$

Note that $R_{xy}(\tau) = R^*_{yx}(-\tau)$ and that $R_{xx}(\tau)$, $R_{yy}(\tau)$, $R_{xy}(\tau)$, and $R_{yx}(\tau)$ are real. Consequently, the cross-correlation of $x(t)$ and $y(t)$ depends only on the delay, τ, between the two signals, which is a direct result of the stationarity assumption. Then, performing the Fourier transform of Eq. (A.32), Eq. (A.33), and Eq. (A.34) yields

$$P_{xx}(\omega) = \frac{2A^2}{\omega^2}(1 - \cos\omega T_x), \tag{A.36}$$

$$P_{yy}(\omega) = \frac{2B^2}{\omega^2}(1 - \cos\omega T_y), \tag{A.37}$$

$$P_{xy}(\omega) = e^{\vec{\jmath}\omega\delta}\frac{AB}{\omega^2}\left(e^{\vec{\jmath}\omega(T_y - T_x)} - e^{\vec{\jmath}\omega T_y} - e^{-\vec{\jmath}\omega T_x} + 1\right). \tag{A.38}$$

The coherence is then

$$\begin{aligned}\gamma^2(\omega) &= \left|\frac{P_{xy}(\omega)P^*_{xy}(\omega)}{P_{xx}(\omega)P_{yy}(\omega)}\right| \\ &= \frac{1-\cos\omega T_x-\cos\omega T_y+\cos\omega T_x\cos\omega T_y}{(1-\cos\omega T_x)(1-\cos\omega T_y)} \\ \gamma^2(\omega) &= 1. \end{aligned} \tag{A.39}$$

A few conclusions can be drawn by this example. The first is that coherence is unity at all frequencies for two rectangular pulses of any amplitude and any width. Therefore coherence is of no use in identifying singular rectangular pulses. Lastly, the example demonstrates that coherence is insensitive to the time delay between rectangular pulses.

Coherence of Two Arbitrary Sinusoids

Now consider the coherence of a two arbitrary sinusoids. Let

$$x(t) = A\cos\left(\omega_x t+\theta\right), \text{ and} \tag{A.40}$$

$$y(t) = B\cos\left(\omega_y t+\beta\right), \tag{A.41}$$

where A and B are constants, ω_x and ω_y are angular frequencies, θ and β are arbitrary phase constants, and t is time. To control the correlation between $x(t)$ and $y(t)$ define a normalized frequency parameter such that

$$m = \frac{\omega_y}{\omega_x}. \tag{A.42}$$

Assuming a uniform probability distribution $\rho_x(t) = \frac{1}{t_2-t_1}$, and using Eq. (A.31) one can show that the auto-correlation of $x(t)$ is

$$\begin{aligned} R_{xx}(\tau) &= A^2\int_{t_1}^{t_2}\cos(\omega_x t+\theta)\cos(\omega_x(t+\tau)+\theta)\rho_x(t)\partial t, \\ &= \frac{A^2}{2(t_2-t_1)}\int_{t_1}^{t_2}\left\{\cos(2\omega_x t+\omega_x\tau+2\theta)+\cos(\omega_x\tau)\right\}\partial t, \end{aligned} \tag{A.43}$$

$$R_{xx}(\tau) = \frac{A^2}{4\omega_x(t_2-t_1)}\Big\{\sin(2\omega_x t_2+\omega_x\tau+2\theta) - \sin(2\omega_x t_1+\omega_x\tau+2\theta)+2\omega_x(t_2-t_1)\cos(\omega_x\tau)\Big\}, \quad (A.44)$$

where t_1 and t_2 are arbitrary limits of integration. In a similar manner, the auto-correlation of $y(t)$ is

$$R_{yy}(\tau) = \frac{B^2}{4\omega_y(t_2-t_1)}\Big\{\sin(2\omega_y t_2+\omega_y\tau+2\beta) - \sin(2\omega_y t_1+\omega_y\tau+2\beta)+2\omega_y(t_2-t_1)\cos(\omega_y\tau)\Big\}, \quad (A.45)$$

however, substituting for ω_y with Eq. (A.42) yields

$$R_{yy}(\tau) = \frac{B^2}{4m\omega_x(t_2-t_1)}\Big\{\sin(2m\omega_x t_2+m\omega_x\tau+2\beta) - \sin(2m\omega_x t_1+m\omega_x\tau+2\beta)+2m\omega_x(t_2-t_1)\cos(m\omega_x\tau)\Big\}. \quad (A.46)$$

The logic for this substitution will shortly be clear. Assuming a uniform joint probability distribution, $\rho_{xy}(t)=\frac{1}{t_2-t_1}$, the cross-correlation of $x(t)$ and $y(t)$ is

$$\begin{aligned} R_{xy}(\tau) &= AB\int_{t_1}^{t_2}\cos(\omega_x t+\theta)\cos(\omega_y(t+\tau)+\beta)\rho_{xy}(t)\partial t, \\ &= \frac{AB}{2(t_2-t_1)}\int_{t_1}^{t_2}\Big\{\cos([m+1]\omega_x t+m\omega_x\tau+\beta+\theta) \\ &\quad +\cos([m-1]\omega_x t+m\omega_x\tau+\beta-\theta)\Big\}\partial t, \\ R_{xy}(\tau) &= \frac{AB}{2\omega_x(t_2-t_1)}\Big\{\frac{1}{m+1}\sin([m+1]\omega_x t_2+m\omega_x\tau+\beta+\theta) \\ &\quad -\frac{1}{m+1}\sin([m+1]\omega_x t_1+m\omega_x\tau+\beta+\theta) \\ &\quad +\frac{1}{m-1}\sin([m-1]\omega_x t_2+m\omega_x\tau+\beta-\theta) \\ &\quad -\frac{1}{m-1}\sin([m-1]\omega_x t_1+m\omega_x\tau+\beta-\theta)\Big\}, \end{aligned} \quad (A.47)$$

The next step in the coherence calculation is the computation of the power spectral densities of Eq. (A.44), Eq. (A.46), and Eq. (A.47). However, each of these correlation functions do not have a Fourier transform if $t_1 \to -\infty$ and $t_2 \to \infty$. If only one frequency is of interest (*i.e.* $\omega_x = \omega_y$), the common method of achieving a deterministic function for $R_{xx}(\tau)$, $R_{yy}(\tau)$, and $R_{xy}(\tau)$ is to set $t_2 = \frac{T}{2}$ and $t_1 = -\frac{T}{2}$ where $\omega_x T = 2\pi$. In this general case $\omega_x \neq \omega_y$, therefore we desire that T be chosen such that at least one full period of each frequency is covered by the integration interval.

Since ω_x is the basis for ω_y, set $\omega_x T = 2\pi n$ where n is any integer. Also assume that $\omega_y T \geq 2\pi k$, where k is any integer. Substituting for ω_y and T reveals $m \geq \frac{k}{n}$ in order to ensure that at least one full cycle of each waveform is included in the integration period. Consequently, if $t_2 = \frac{T}{2}$ and $t_1 = -\frac{T}{2}$, where $T = \frac{2\pi n}{\omega_x}$, then

$$R_{xx}(\tau) = \frac{A^2}{2}\cos(\omega_x \tau), \tag{A.48}$$

$$R_{yy}(\tau) = \frac{B^2}{2}\cos(m\omega_x \tau) + \frac{B^2}{4\pi mn}\sin(2\pi mn)\cos(m\omega_x \tau + 2\beta), \text{ and} \tag{A.49}$$

$$\begin{aligned} R_{xy}(\tau) = \frac{AB}{2\pi n}\Bigg\{ & \frac{1}{m+1}\sin(\pi[m+1]n)\cos(m\omega_x \tau + \beta + \theta) \\ & + \frac{1}{m-1}\sin(\pi[m-1]n)\cos(m\omega_x \tau + \beta - \theta)\Bigg\}. \end{aligned} \tag{A.50}$$

The Fourier Transforms of Eq. (A.48) to Eq. (A.50) can now be written by inspection as:

$$P_{xx}(\omega) = \frac{A^2}{4}\Big\{\delta(\omega - \omega_x) + \delta(\omega + \omega_x)\Big\}, \tag{A.51}$$

$$\begin{aligned} P_{yy}(\omega) = \frac{B^2}{4}\Bigg\{ & \delta(\omega - m\omega_x) + \delta(\omega + m\omega_x) \\ & + \frac{\sin(2\pi mn)}{2\pi mn}\left[e^{\vec{j}2\beta}\delta(\omega - m\omega_x) + e^{-\vec{j}2\beta}\delta(\omega + m\omega_x)\right]\Bigg\}, \end{aligned} \tag{A.52}$$

$$\begin{aligned} P_{xy}(\omega) = \frac{AB}{4}\Big\{ & C_1\left[e^{\vec{j}(\beta+\theta)}\delta(\omega - m\omega_x) + e^{-\vec{j}(\beta+\theta)}\delta(\omega + m\omega_x)\right] \\ & + C_2\left[e^{\vec{j}(\beta-\theta)}\delta(\omega - m\omega_x) + e^{-\vec{j}(\beta-\theta)}\delta(\omega + m\omega_x)\right]\Big\}, \end{aligned} \tag{A.53}$$

where

$$\begin{aligned} C_1 &= \frac{\sin(\pi[m+1]n)}{\pi[m+1]n}, \text{ and} \\ C_2 &= \frac{\sin(\pi[m-1]n)}{\pi[m-1]n}, \end{aligned}$$

and where $\delta(\omega)$ is a the Dirac delta function in the angular frequency domain.

Using the identity that $\delta(a-b)\delta(a+b) = 0$ for $a \neq 0$ and $b \neq 0$, and $\delta^2(a) = \delta(a)$, one finds that

$$\begin{aligned} P_{xy}(\omega)P^*_{xy}(\omega) &= \left(\frac{AB}{4}\right)^2 \left\{C_1^2 + C_2^2 + 2C_1C_2\cos(2\theta)\right\} \\ &\quad \times \left\{\delta(\omega - m\omega_x) + \delta(\omega + m\omega_x)\right\}. \end{aligned} \tag{A.54}$$

Inserting Eq. (A.51), Eq. (A.52), and Eq. (A.54) into the coherence equation, Eq. (A.28), and solving for $\gamma^2(\omega)$ yields,

$$\begin{aligned} \gamma^2(\omega) &= \left|\frac{f_0(\omega)}{f_1(\omega) + f_2(\omega)}\right|, \text{ where} \\ f_0(\omega) &= \left\{C_1^2 + C_2^2 + 2C_1C_2\cos(2\theta)\right\}\left\{\delta(\omega - m\omega_x) + \delta(\omega + m\omega_x)\right\}, \\ f_1(\omega) &= \left\{1 + e^{\vec{j}2\beta}\,\mathrm{sinc}(2\pi mn)\right\}\delta(\omega - m\omega_x), \text{ and} \\ f_2(\omega) &= \left\{1 + e^{-\vec{j}2\beta}\,\mathrm{sinc}(2\pi mn)\right\}\delta(\omega + m\omega_x). \end{aligned} \tag{A.55}$$

Simplifying results in

$$\begin{aligned} \gamma^2(\omega) &= \frac{f_3(\omega)}{\sqrt{1 + 2\cos(2\beta)\,\mathrm{sinc}(2\pi mn) + \mathrm{sinc}^2(2\pi mn)}}, \text{ where} \\ f_3(\omega) &= \mathrm{sinc}^2(\pi[m+1]n) + \mathrm{sinc}^2(\pi[m-1]n) \\ &\quad + 2\,\mathrm{sinc}(\pi[m+1]n)\,\mathrm{sinc}(\pi[m-1]n)\cos(2\theta). \end{aligned} \tag{A.56}$$

So, it becomes clear that the coherence of two arbitrary sinusoids depends only on the normalized frequency, m, and their relative phases. For a given m and n the coherence is a constant—this is an example of separable sinusoidal process as described by

Nuttall (1958). Remember that m is a real number whereas n is an integer. So for the special case of $m = \pm 1$ (*i.e.* $\omega_y = \pm\omega_x$), the coherence function collapses to unity (the expected result) irrespective of phase. The coherence is only one when $m = \pm 1$ and is zero for $m \in \{0, \pm 2, \pm 3, \ldots\}$. For all other m, the coherence takes on a value between zero and one.

Assume that $\theta = 0$ and $\beta = 0$, then Eq. (A.56) becomes

$$\gamma^2(\omega) = \frac{\Big\{ \operatorname{sinc}(\pi[m+1]n) + \operatorname{sinc}(\pi[m-1]n) \Big\}^2}{1 + \operatorname{sinc}(2\pi m n)}. \tag{A.57}$$

Equation (A.57) is plotted in Figure A.3 for the case where $n = 1$. Note the null coherence at multiples of ω_x (*i.e.* integer values of m) and the local maxima at non-integer values of m.

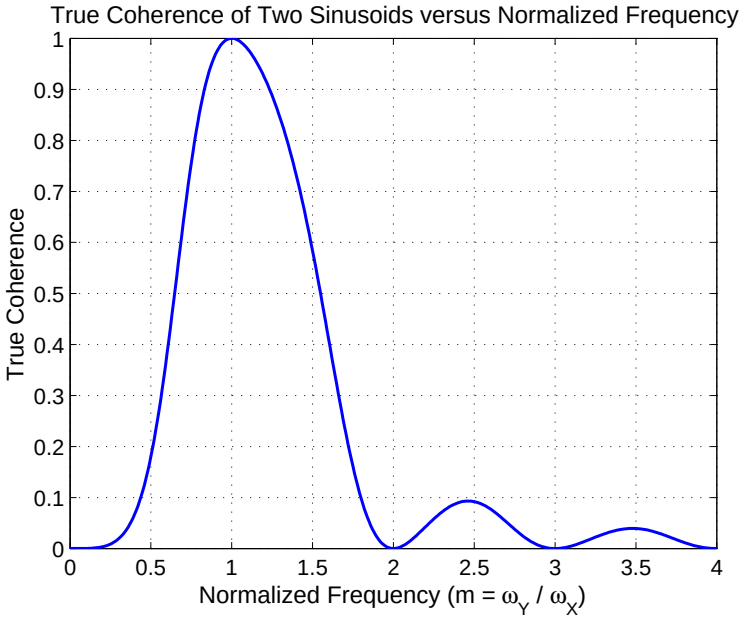


Figure A.3. Theoretical coherence of two arbitrary sinusoids (repeated). The true theoretical coherence of two sinusoids of arbitrary frequency is related to a sinc function of normalized frequency, $m = \frac{\omega_y}{\omega_x}$. Note the null coherence at multiples of ω_x (*i.e.* integer values of m). Also note the local maxima at non-integer values of m. Clearly the greater the frequency difference (*i.e.* $\omega_y - \omega_x$), the lower the coherence. In this example $n = 1$.

Coherence of Continuous Phase Modulated Signals

What is the coherence of two continuous phase modulated (CPM) signals? To answer this question, one must first determine the auto- and cross-power spectral densities of the two signals. Let the low-pass equivalent of a CPM signal be defined by

$$x(t) = e^{\mathrm{e}\vec{\mathrm{j}}\varphi(t;\boldsymbol{I})}, \tag{A.58}$$

where

$$\begin{aligned} \varphi(t;\boldsymbol{I}) &= 2\pi h \sum_{k=-\infty}^{\infty} I_k q(t-kT), \\ \boldsymbol{I} &\in \{\pm 1, \pm 3, \pm 5, \ldots, \pm(M-1)\}, \end{aligned}$$

$\boldsymbol{I}$ is the information sequence, h is the modulation index, M is the modulation level, T is the bit period, and $q(t)$ is an arbitrary pulse shape.

Assume that each symbol of the information sequence is statistically independent and identically distributed. With this assumption Proakis (1989) shows, through some effort, that the auto-power spectral density of Eq. (A.58) with a rectangular pulse shape is

$$P_{xx}(f) = T_x \left\{ \frac{1}{M_x} \sum_{n=1}^{M_x} A_n^2(f) + \frac{2}{M_x^2} \sum_{n=1}^{M_x} \sum_{m=1}^{M_x} B_{nm}(f) A_n(f) A_m(f) \right\} \tag{A.59}$$

where

$$\begin{aligned} A_n(f) &= \frac{\sin\left(\pi f T_x - \frac{\pi h_x}{2}[2n-1-M_x]\right)}{\pi\left(f T_x - \frac{h_x}{2}[2n-1-M_x]\right)}, \\ B_{nm}(f) &= \frac{\cos(2\pi f T_x - \alpha_{nm}) - \psi\cos\alpha_{nm}}{1+\psi^2-2\psi\cos 2\pi f T_x}, \\ \alpha_{nm} &= \pi h_x (m+n-1-M_x), \\ \psi &= \frac{\sin(M_x \pi h_x)}{M_x \sin(\pi h_x)}, \end{aligned}$$

M_x is the modulation level (*e.g.* 2, 4, 6, ...) of the modulating signal with information sequence $\boldsymbol{I}_x$, T_x is the bit period, h_x is the modulation index $2f_dT_x$, and f_d is the peak frequency deviation.

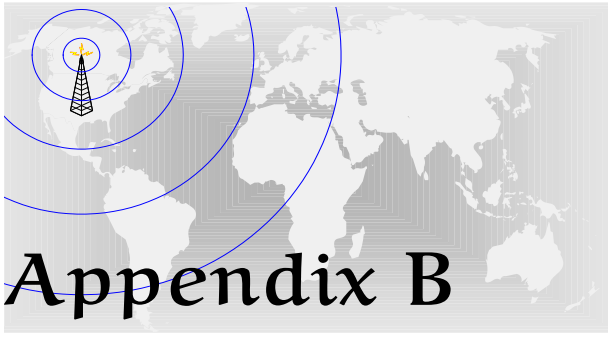
Given $y(t)$,

$$y(t) = \mathrm{e}^{\vec{\mathrm{j}}\lambda(t;\boldsymbol{J})}, \tag{A.60}$$

where

$$\begin{aligned} \lambda(t;\boldsymbol{J}) &= 2\pi h \sum_{k=-\infty}^{\infty} J_k q(t-kT), \\ \boldsymbol{J} &\in \{\pm 1, \pm 3, \pm 5, \ldots, \pm(M-1)\}, \end{aligned}$$

what is the cross-power spectrum of $x(t)$ and $y(t)$? Some mathematical rigour is required to answer to this question. This remains a future challenge.



Appendix B

Data Collection

THIS appendix summarizes the data sets collected by the swept-narrowband receiver and the broadband receiver. The data from the swept-narrowband receiver is used only for analyzing the HF noise PDF. Data from two versions of the broadband receiver are used for HF noise analyses and signal analyses. The narrowband receiver—not the swept-narrowband receiver—is the forerunner of the broadband receiver; signals collected from it are those used for signal analysis. Data from the broadband receiver is used for the broadband method of measuring the HF noise PDF.

B.1 Data Set for Part II

Swept-Narrowband Data—Adelaide

The description of the swept-narrowband data set provided by Chapter 5 is complete. The only additional comments are that the data set consists of 101 encoded data files; each file has an average size of 7.6 MB of 32-bit samples. In the first column of the list below, is the filename. The second column indicates the date and time of the start of the recording, while the third column represents the number of bytes in each file. Recordings occur approximately every 15 minutes for a 3 minute period. This schedule is aligned with an ionosonde transmission schedule discussed by Brine *et al* (2002). This list is only included for reference. Decoding of each file involves a complicated process best described by Brine.

1. MCR-20050216-061510-ChID08.rundata 16-Feb-2005 05:18:20 7702528
2. MCR-20050216-063011-ChID08.rundata 16-Feb-2005 05:33:18 7689216
3. MCR-20050216-064510-ChID08.rundata 16-Feb-2005 05:48:18 7700480
4. MCR-20050216-070010-ChID08.rundata 16-Feb-2005 06:03:18 7701504
5. MCR-20050216-071510-ChID08.rundata 16-Feb-2005 06:18:18 7700480
6. MCR-20050216-073010-ChID08.rundata 16-Feb-2005 06:33:18 7698432
7. MCR-20050216-074510-ChID08.rundata 16-Feb-2005 06:48:18 7699456
8. MCR-20050216-080010-ChID08.rundata 16-Feb-2005 07:03:18 7701504
9. MCR-20050216-081510-ChID08.rundata 16-Feb-2005 07:18:18 7701504
10. MCR-20050216-083010-ChID08.rundata 16-Feb-2005 07:33:18 7701504
11. MCR-20050216-084510-ChID08.rundata 16-Feb-2005 07:48:18 7699456
12. MCR-20050216-090010-ChID08.rundata 16-Feb-2005 08:03:18 7701504
13. MCR-20050216-091510-ChID08.rundata 16-Feb-2005 08:18:18 7698432
14. MCR-20050216-093010-ChID08.rundata 16-Feb-2005 08:33:18 7700480
15. MCR-20050216-094510-ChID08.rundata 16-Feb-2005 08:48:18 7700480
16. MCR-20050216-100010-ChID08.rundata 16-Feb-2005 09:03:18 7698432
17. MCR-20050216-101510-ChID08.rundata 16-Feb-2005 09:18:18 7699456
18. MCR-20050216-103010-ChID08.rundata 16-Feb-2005 09:33:18 7698432
19. MCR-20050216-104510-ChID08.rundata 16-Feb-2005 09:48:18 7704576

20. MCR-20050216-110010-ChID08.rundata 16-Feb-2005 10:03:18 7699456
21. MCR-20050216-111510-ChID08.rundata 16-Feb-2005 10:18:18 7700480
22. MCR-20050216-113010-ChID08.rundata 16-Feb-2005 10:33:18 7699456
23. MCR-20050216-114510-ChID08.rundata 16-Feb-2005 10:48:18 7699456
24. MCR-20050216-120010-ChID08.rundata 16-Feb-2005 11:03:18 7702528
25. MCR-20050216-121510-ChID08.rundata 16-Feb-2005 11:18:18 7698432
26. MCR-20050216-123010-ChID08.rundata 16-Feb-2005 11:33:18 7701504
27. MCR-20050216-124510-ChID08.rundata 16-Feb-2005 11:48:18 7700480
28. MCR-20050216-130010-ChID08.rundata 16-Feb-2005 12:03:18 7700480
29. MCR-20050216-131510-ChID08.rundata 16-Feb-2005 12:18:18 7700480
30. MCR-20050216-133010-ChID08.rundata 16-Feb-2005 12:33:18 7700480
31. MCR-20050216-134510-ChID08.rundata 16-Feb-2005 12:48:18 7699456
32. MCR-20050216-140010-ChID08.rundata 16-Feb-2005 13:03:18 7701504
33. MCR-20050216-141510-ChID08.rundata 16-Feb-2005 13:18:18 7700480
34. MCR-20050216-143010-ChID08.rundata 16-Feb-2005 13:33:18 7700480
35. MCR-20050216-144510-ChID08.rundata 16-Feb-2005 13:48:18 7699456
36. MCR-20050216-150010-ChID08.rundata 16-Feb-2005 14:03:18 7701504
37. MCR-20050216-151510-ChID08.rundata 16-Feb-2005 14:18:18 7702528
38. MCR-20050216-153010-ChID08.rundata 16-Feb-2005 14:33:18 7700480
39. MCR-20050216-154510-ChID08.rundata 16-Feb-2005 14:48:18 7699456
40. MCR-20050216-160010-ChID08.rundata 16-Feb-2005 15:03:18 7702528
41. MCR-20050216-161510-ChID08.rundata 16-Feb-2005 15:18:18 7702528
42. MCR-20050216-163010-ChID08.rundata 16-Feb-2005 15:33:18 7700480
43. MCR-20050216-164510-ChID08.rundata 16-Feb-2005 15:48:18 7701504
44. MCR-20050216-170010-ChID08.rundata 16-Feb-2005 16:03:18 7701504
45. MCR-20050216-171510-ChID08.rundata 16-Feb-2005 16:18:18 7698432
46. MCR-20050216-173010-ChID08.rundata 16-Feb-2005 16:33:18 7699456
47. MCR-20050216-174510-ChID08.rundata 16-Feb-2005 16:48:18 7700480
48. MCR-20050216-180010-ChID08.rundata 16-Feb-2005 17:07:40 18673664

49.	MCR-20050216-181510-ChID08.rundata	16-Feb-2005 17:18:18	7699456
50.	MCR-20050216-183010-ChID08.rundata	16-Feb-2005 17:33:18	7699456
51.	MCR-20050216-184510-ChID08.rundata	16-Feb-2005 17:48:18	7698432
52.	MCR-20050216-190010-ChID08.rundata	16-Feb-2005 18:03:18	7703552
53.	MCR-20050216-194510-ChID08.rundata	16-Feb-2005 18:52:40	18678784
54.	MCR-20050216-200010-ChID08.rundata	16-Feb-2005 19:03:14	7673856
55.	MCR-20050216-201510-ChID08.rundata	16-Feb-2005 19:18:18	7700480
56.	MCR-20050216-203010-ChID08.rundata	16-Feb-2005 19:33:18	7700480
57.	MCR-20050216-204510-ChID08.rundata	16-Feb-2005 19:48:18	7700480
58.	MCR-20050216-210010-ChID08.rundata	16-Feb-2005 20:03:18	7699456
59.	MCR-20050216-211510-ChID08.rundata	16-Feb-2005 20:18:18	7698432
60.	MCR-20050216-213010-ChID08.rundata	16-Feb-2005 20:37:40	18264064
61.	MCR-20050216-214510-ChID08.rundata	16-Feb-2005 20:48:18	7699456
62.	MCR-20050216-220010-ChID08.rundata	16-Feb-2005 21:03:18	7700480
63.	MCR-20050216-221510-ChID08.rundata	16-Feb-2005 21:18:18	7698432
64.	MCR-20050216-223010-ChID08.rundata	16-Feb-2005 21:33:18	7699456
65.	MCR-20050216-224510-ChID08.rundata	16-Feb-2005 21:48:18	7699456
66.	MCR-20050216-230010-ChID08.rundata	16-Feb-2005 22:03:18	7699456
67.	MCR-20050216-231510-ChID08.rundata	16-Feb-2005 22:18:18	7699456
68.	MCR-20050216-233010-ChID08.rundata	16-Feb-2005 22:33:14	7676928
69.	MCR-20050216-234510-ChID08.rundata	16-Feb-2005 22:48:18	7698432
70.	MCR-20050217-000010-ChID08.rundata	16-Feb-2005 23:03:18	7701504
71.	MCR-20050217-003011-ChID08.rundata	16-Feb-2005 23:33:14	7639040
72.	MCR-20050217-004510-ChID08.rundata	16-Feb-2005 23:48:14	7667712
73.	MCR-20050217-010010-ChID08.rundata	17-Feb-2005 00:03:18	7700480
74.	MCR-20050217-011510-ChID08.rundata	17-Feb-2005 00:18:20	7702528
75.	MCR-20050217-013010-ChID08.rundata	17-Feb-2005 00:33:18	7699456
76.	MCR-20050217-020010-ChID08.rundata	17-Feb-2005 01:03:18	7700480
77.	MCR-20050217-021510-ChID08.rundata	17-Feb-2005 01:18:18	7699456

78.	MCR-20050217-023010-ChID08.rundata	17-Feb-2005 01:33:18	7699456
79.	MCR-20050217-024510-ChID08.rundata	17-Feb-2005 01:48:20	7699456
80.	MCR-20050217-030010-ChID08.rundata	17-Feb-2005 02:03:18	7702528
81.	MCR-20050217-031510-ChID08.rundata	17-Feb-2005 02:18:18	7698432
82.	MCR-20050217-033010-ChID08.rundata	17-Feb-2005 02:33:18	7700480
83.	MCR-20050217-034510-ChID08.rundata	17-Feb-2005 02:48:18	7702528
84.	MCR-20050217-040010-ChID08.rundata	17-Feb-2005 03:03:18	7704576
85.	MCR-20050217-041510-ChID08.rundata	17-Feb-2005 03:18:18	7698432
86.	MCR-20050217-043010-ChID08.rundata	17-Feb-2005 03:33:18	7699456
87.	MCR-20050217-044510-ChID08.rundata	17-Feb-2005 03:48:18	7700480
88.	MCR-20050217-050010-ChID08.rundata	17-Feb-2005 05:27:46	7702528
89.	MCR-20050217-063011-ChID08.rundata	17-Feb-2005 05:33:18	7701504
90.	MCR-20050217-064510-ChID08.rundata	17-Feb-2005 05:49:52	7701503
91.	MCR-20050217-070011-ChID08.rundata	17-Feb-2005 06:03:18	7701504
92.	MCR-20050217-071510-ChID08.rundata	17-Feb-2005 06:18:18	7698432
93.	MCR-20050217-073010-ChID08.rundata	17-Feb-2005 06:33:18	7699456
94.	MCR-20050217-074510-ChID08.rundata	17-Feb-2005 06:48:18	7700480
95.	MCR-20050217-080010-ChID08.rundata	17-Feb-2005 07:03:18	7700480
96.	MCR-20050217-081510-ChID08.rundata	17-Feb-2005 07:18:18	7701504
97.	MCR-20050217-083010-ChID08.rundata	17-Feb-2005 07:33:18	7700480
98.	MCR-20050217-084510-ChID08.rundata	17-Feb-2005 07:48:18	7701504
99.	MCR-20050217-090010-ChID08.rundata	17-Feb-2005 08:03:20	7700480
100.	MCR-20050217-091510-ChID08.rundata	17-Feb-2005 08:18:14	7650304
101.	MCR-20050217-093010-ChID08.rundata	17-Feb-2005 08:33:18	7699456

Broadband Data—Swan Reach

For reasons of its own, Ebor Computing arranged to have known HF signals transmitted from various sites in Australia. These sites are far enough from Swan Reach that the signals propagate by ionospheric modes. The five different baseband signals transmitted from these sites are:

A : 8-PSK, 1200 baud, Stanag 4285, 511-bit pseudo-random sequence;

B : FSK Wide, 150 baud, Mil-Std-188-110A, 511-bit pseudo-random sequence;

C : FSK Narrow, 75 baud, Mil-Std-188-110A, 511-bit pseudo-random sequence;

D : Voice dialogue, band-limited to 3kHz, spoken English (female/male); and

E : Voice dialogue, band-limited to 3kHz, spoken Chinese (Mandarin, Cantonese).

Programs A, B, and C are generated from a BAE Systems ARM 9401 HF Modem. Baseband audio output from the modem is recorded with a SoundBlaster Vibra 16x CT4170 Sound Card with the following specifications.

Output Power : 4 W max (4 Ω load minimum)

Output Signal : 8.8 Vpp max.

Mic Input Impedance : 600 Ω

Mic Input Range : 30 mVpp – 200 mVpp

Line-in Impedance : 15 kΩ

Line-in Range : 0 – 2 Vpp

The baseband audio recorded by the sound card is stored in monaural .wav format with 16-bit samples at a sampling rate of 11,025 Hz. These files are used to key the transmitters at the various transmit sites. For the April, 2006 data collection session all the transmit sites broadcast the same program simultaneously. For the May, 2006 session all transmitters broadcast different programs.

In the context of this thesis, the programs are little more than points of interest. The primary use of the broadband data set is for measuring the HF noise PDF, but this does not preclude the use of the data set for further modulation recognition studies. In all, there are 176 data files containing 32-bit I-Q sample pairs. The average size of each file is 1.55 GB.

Filenames for recorded sessions follow the format of

YYYY.MM.DD-HH.MM.SS_<computer name>.Sahara.net.au_DDC_<ddc#>.bin

where YYYY is the year, MM is the month, DD is the day, HH is the hour, MM is the minute, and SS is the second that the recording started. The date and time of the recording is based on the computer clock, which is not synchronized to UTC. The computer name is either "Octopus" or "Cuttlefish" and the DDC number is 0, 1, 2, or 3. "Cuttlefish" was connected to antennas 1, 2, 3, and 4. "Octopus" was connected to antennas 5, 6, 7, and 8. Files with names ending in "DDC_0" correspond to data collected from antenna 1 (for "Cuttlefish") or antenna 5 (for "Octopus"). For filenames ending in "DDC_3", the data corresponds to antenna 3 (for "Cuttlefish") or antenna 8 (for "Octopus"). Files with names ending in "DDC_1" or "DDC_2" are similarly mapped.

The data in the files is stored as 32-bit integers in I-Q-I-Q-. . . format. The 20 most significant bits contain the actual quantized sample. Bits 8-12 contain a sample tag created by the DDC (see GC4016 documentation), and bits 0-7 are unary pad bits. The complex sampling rate is 153,600 Hz and 100% FIR filters (see GC4016 documentation) are used in the DDCs. Consequently the bandwidth of the downconverted data is 153,600 Hz centered at zero where zero represents the tuning frequency (*e.g.* 13.194 MHz).

In the first column of the list below, is the filename. The second column indicates the date and time of the start of the recording, while the third column represents the number of bytes in each file. Recordings occur approximately every 30 minutes for a 30 minute period. This schedule is prescribed by Ebor Computing for its purposes. This list is only included for reference.

1. 2006.04.06-16.57.02_Octopus.Sahara.net.au_DDC_0.bin 06-Apr-17:58:01 72335360
2. 2006.04.06-16.57.02_Octopus.Sahara.net.au_DDC_1.bin 06-Apr-17:58:01 72335360
3. 2006.04.06-16.57.02_Octopus.Sahara.net.au_DDC_2.bin 06-Apr-17:58:01 72335360
4. 2006.04.06-16.57.02_Octopus.Sahara.net.au_DDC_3.bin 06-Apr-17:58:01 72335360
5. 2006.04.06-17.18.08_Octopus.Sahara.net.au_DDC_0.bin 06-Apr-18:27:41 705331200
6. 2006.04.06-17.18.08_Octopus.Sahara.net.au_DDC_1.bin 06-Apr-18:27:41 705331200
7. 2006.04.06-17.18.08_Octopus.Sahara.net.au_DDC_2.bin 06-Apr-18:27:41 705331200
8. 2006.04.06-17.18.08_Octopus.Sahara.net.au_DDC_3.bin 06-Apr-18:27:41 705331200
9. 2006.04.06-17.28.19_Octopus.Sahara.net.au_DDC_0.bin 06-Apr-18:58:22 2070675456
10. 2006.04.06-17.28.19_Octopus.Sahara.net.au_DDC_1.bin 06-Apr-18:58:22 2070675456
11. 2006.04.06-17.28.19_Octopus.Sahara.net.au_DDC_2.bin 06-Apr-18:58:22 2070675456
12. 2006.04.06-17.28.19_Octopus.Sahara.net.au_DDC_3.bin 06-Apr-18:58:22 2070675456
13. 2006.04.06-18.02.01_Octopus.Sahara.net.au_DDC_0.bin 06-Apr-19:30:18 2093056000
14. 2006.04.06-18.02.01_Octopus.Sahara.net.au_DDC_1.bin 06-Apr-19:30:18 2093056000
15. 2006.04.06-18.02.01_Octopus.Sahara.net.au_DDC_2.bin 06-Apr-19:30:18 2093056000
16. 2006.04.06-18.02.01_Octopus.Sahara.net.au_DDC_3.bin 06-Apr-19:30:18 2093056000
17. 2006.04.06-18.30.38_Octopus.Sahara.net.au_DDC_0.bin 06-Apr-20:00:43 2067316736
18. 2006.04.06-18.30.38_Octopus.Sahara.net.au_DDC_1.bin 06-Apr-20:00:43 2067316736
19. 2006.04.06-18.30.38_Octopus.Sahara.net.au_DDC_2.bin 06-Apr-20:00:43 2067316736
20. 2006.04.06-18.30.38_Octopus.Sahara.net.au_DDC_3.bin 06-Apr-20:00:43 2067316736
21. 2006.04.06-19.01.40_Octopus.Sahara.net.au_DDC_0.bin 06-Apr-20:31:21 2097135616
22. 2006.04.06-19.01.40_Octopus.Sahara.net.au_DDC_1.bin 06-Apr-20:31:21 2097135616
23. 2006.04.06-19.01.40_Octopus.Sahara.net.au_DDC_2.bin 06-Apr-20:31:21 2097135616
24. 2006.04.06-19.01.40_Octopus.Sahara.net.au_DDC_3.bin 06-Apr-20:31:21 2097135616
25. 2006.04.06-19.31.42_Octopus.Sahara.net.au_DDC_0.bin 06-Apr-20:59:33 2062499840
26. 2006.04.06-19.31.42_Octopus.Sahara.net.au_DDC_1.bin 06-Apr-20:59:33 2062499840
27. 2006.04.06-19.31.42_Octopus.Sahara.net.au_DDC_2.bin 06-Apr-20:59:33 2062499840
28. 2006.04.06-19.31.42_Octopus.Sahara.net.au_DDC_3.bin 06-Apr-20:59:33 2062499840
29. 2006.04.06-20.00.03_Octopus.Sahara.net.au_DDC_0.bin 06-Apr-21:30:07 2070265856

30.	2006.04.06-20.00.03_Octopus.Sahara.net.au_DDC_1.bin	06-Apr-21:30:07	2070265856
31.	2006.04.06-20.00.03_Octopus.Sahara.net.au_DDC_2.bin	06-Apr-21:30:07	2070265856
32.	2006.04.06-20.00.03_Octopus.Sahara.net.au_DDC_3.bin	06-Apr-21:30:07	2070265856
33.	2006.04.06-20.30.40_Octopus.Sahara.net.au_DDC_0.bin	06-Apr-22:00:11	2108768256
34.	2006.04.06-20.30.40_Octopus.Sahara.net.au_DDC_1.bin	06-Apr-22:00:11	2108850176
35.	2006.04.06-20.30.40_Octopus.Sahara.net.au_DDC_2.bin	06-Apr-22:00:11	2108850176
36.	2006.04.06-20.30.40_Octopus.Sahara.net.au_DDC_3.bin	06-Apr-22:00:11	2108850176
37.	2006.04.06-21.00.31_Octopus.Sahara.net.au_DDC_0.bin	06-Apr-22:04:34	297943040
38.	2006.04.06-21.00.31_Octopus.Sahara.net.au_DDC_1.bin	06-Apr-22:04:34	297943040
39.	2006.04.06-21.00.31_Octopus.Sahara.net.au_DDC_2.bin	06-Apr-22:04:34	297943040
40.	2006.04.06-21.00.31_Octopus.Sahara.net.au_DDC_3.bin	06-Apr-22:04:34	297943040
41.	2006.04.06-21.04.52_Octopus.Sahara.net.au_DDC_0.bin	06-Apr-22:30:03	1864089600
42.	2006.04.06-21.04.52_Octopus.Sahara.net.au_DDC_1.bin	06-Apr-22:30:03	1864089600
43.	2006.04.06-21.04.52_Octopus.Sahara.net.au_DDC_2.bin	06-Apr-22:30:03	1864089600
44.	2006.04.06-21.04.52_Octopus.Sahara.net.au_DDC_3.bin	06-Apr-22:30:03	1864089600
45.	2006.04.06-21.30.19_Octopus.Sahara.net.au_DDC_0.bin	06-Apr-22:58:24	2078146560
46.	2006.04.06-21.30.19_Octopus.Sahara.net.au_DDC_1.bin	06-Apr-22:58:24	2078146560
47.	2006.04.06-21.30.19_Octopus.Sahara.net.au_DDC_2.bin	06-Apr-22:58:24	2078146560
48.	2006.04.06-21.30.19_Octopus.Sahara.net.au_DDC_3.bin	06-Apr-22:58:24	2078146560
49.	2006.04.06-21.58.51_Octopus.Sahara.net.au_DDC_0.bin	06-Apr-23:28:54	2069364736
50.	2006.04.06-21.58.51_Octopus.Sahara.net.au_DDC_1.bin	06-Apr-23:28:54	2069364736
51.	2006.04.06-21.58.51_Octopus.Sahara.net.au_DDC_2.bin	06-Apr-23:28:54	2069364736
52.	2006.04.06-21.58.51_Octopus.Sahara.net.au_DDC_3.bin	06-Apr-23:28:54	2069364736
53.	2006.04.06-22.29.27_Octopus.Sahara.net.au_DDC_0.bin	06-Apr-23:59:30	2072969216
54.	2006.04.06-22.29.27_Octopus.Sahara.net.au_DDC_1.bin	06-Apr-23:59:30	2072969216
55.	2006.04.06-22.29.27_Octopus.Sahara.net.au_DDC_2.bin	06-Apr-23:59:30	2072969216
56.	2006.04.06-22.29.27_Octopus.Sahara.net.au_DDC_3.bin	06-Apr-23:59:30	2072969216
57.	2006.04.07-04.31.14_Octopus.Sahara.net.au_DDC_0.bin	07-Apr-05:59:07	2062745600
58.	2006.04.07-04.31.14_Octopus.Sahara.net.au_DDC_1.bin	07-Apr-05:59:07	2062745600

59.	2006.04.07-04.31.14_Octopus.Sahara.net.au_DDC_2.bin	07-Apr-05:59:07	2062745600
60.	2006.04.07-04.31.14_Octopus.Sahara.net.au_DDC_3.bin	07-Apr-05:59:07	2062745600
61.	2006.04.07-04.59.26_Octopus.Sahara.net.au_DDC_0.bin	07-Apr-06:27:36	2084454400
62.	2006.04.07-04.59.26_Octopus.Sahara.net.au_DDC_1.bin	07-Apr-06:27:36	2084454400
63.	2006.04.07-04.59.26_Octopus.Sahara.net.au_DDC_2.bin	07-Apr-06:27:36	2084454400
64.	2006.04.07-04.59.26_Octopus.Sahara.net.au_DDC_3.bin	07-Apr-06:27:36	2084454400
65.	2006.04.07-05.29.08_Octopus.Sahara.net.au_DDC_0.bin	07-Apr-06:59:13	2070020096
66.	2006.04.07-05.29.08_Octopus.Sahara.net.au_DDC_1.bin	07-Apr-06:59:13	2070020096
67.	2006.04.07-05.29.08_Octopus.Sahara.net.au_DDC_2.bin	07-Apr-06:59:13	2070020096
68.	2006.04.07-05.29.08_Octopus.Sahara.net.au_DDC_3.bin	07-Apr-06:59:13	2070020096
69.	2006.04.07-05.59.30_Octopus.Sahara.net.au_DDC_0.bin	07-Apr-07:22:22	1692876800
70.	2006.04.07-05.59.30_Octopus.Sahara.net.au_DDC_1.bin	07-Apr-07:22:22	1692876800
71.	2006.04.07-05.59.30_Octopus.Sahara.net.au_DDC_2.bin	07-Apr-07:22:22	1692876800
72.	2006.04.07-05.59.30_Octopus.Sahara.net.au_DDC_3.bin	07-Apr-07:22:22	1692876800
73.	2006.04.07-06.29.16_Octopus.Sahara.net.au_DDC_0.bin	07-Apr-07:32:00	200867840
74.	2006.04.07-06.29.16_Octopus.Sahara.net.au_DDC_1.bin	07-Apr-07:32:00	200867840
75.	2006.04.07-06.29.16_Octopus.Sahara.net.au_DDC_2.bin	07-Apr-07:32:00	200867840
76.	2006.04.07-06.29.16_Octopus.Sahara.net.au_DDC_3.bin	07-Apr-07:32:00	200867840
77.	2006.04.07-06.32.32_Octopus.Sahara.net.au_DDC_0.bin	07-Apr-07:59:05	1964687360
78.	2006.04.07-06.32.32_Octopus.Sahara.net.au_DDC_1.bin	07-Apr-07:59:05	1964687360
79.	2006.04.07-06.32.32_Octopus.Sahara.net.au_DDC_2.bin	07-Apr-07:59:05	1964687360
80.	2006.04.07-06.32.32_Octopus.Sahara.net.au_DDC_3.bin	07-Apr-07:59:05	1964687360
81.	2006.04.07-06.59.24_Octopus.Sahara.net.au_DDC_0.bin	07-Apr-08:28:41	2126462976
82.	2006.04.07-06.59.24_Octopus.Sahara.net.au_DDC_1.bin	07-Apr-08:28:41	2126462976
83.	2006.04.07-06.59.24_Octopus.Sahara.net.au_DDC_2.bin	07-Apr-08:28:41	2126462976
84.	2006.04.07-06.59.24_Octopus.Sahara.net.au_DDC_3.bin	07-Apr-08:28:41	2126462976
85.	2006.04.07-07.28.55_Octopus.Sahara.net.au_DDC_0.bin	07-Apr-08:34:17	395509760
86.	2006.04.07-07.28.55_Octopus.Sahara.net.au_DDC_1.bin	07-Apr-08:34:17	395509760
87.	2006.04.07-07.28.55_Octopus.Sahara.net.au_DDC_2.bin	07-Apr-08:34:17	395509760

88.	2006.04.07-07.28.55_Octopus.Sahara.net.au_DDC_3.bin	07-Apr-08:34:17	395509760
89.	2006.04.07-07.34.37_Octopus.Sahara.net.au_DDC_0.bin	07-Apr-08:58:54	1798225920
90.	2006.04.07-07.34.37_Octopus.Sahara.net.au_DDC_1.bin	07-Apr-08:58:54	1798225920
91.	2006.04.07-07.34.37_Octopus.Sahara.net.au_DDC_2.bin	07-Apr-08:58:54	1798225920
92.	2006.04.07-07.34.37_Octopus.Sahara.net.au_DDC_3.bin	07-Apr-08:58:54	1798225920
93.	2006.04.07-07.59.18_Octopus.Sahara.net.au_DDC_0.bin	07-Apr-09:29:22	2069692416
94.	2006.04.07-07.59.18_Octopus.Sahara.net.au_DDC_1.bin	07-Apr-09:29:22	2069692416
95.	2006.04.07-07.59.18_Octopus.Sahara.net.au_DDC_2.bin	07-Apr-09:29:22	2069692416
96.	2006.04.07-07.59.18_Octopus.Sahara.net.au_DDC_3.bin	07-Apr-09:29:22	2069692416
97.	2006.04.07-08.29.35_Octopus.Sahara.net.au_DDC_0.bin	07-Apr-09:59:39	2068545536
98.	2006.04.07-08.29.35_Octopus.Sahara.net.au_DDC_1.bin	07-Apr-09:59:39	2068545536
99.	2006.04.07-08.29.35_Octopus.Sahara.net.au_DDC_2.bin	07-Apr-09:59:39	2068545536
100.	2006.04.07-08.29.35_Octopus.Sahara.net.au_DDC_3.bin	07-Apr-09:59:39	2068545536
101.	2006.04.07-08.59.57_Octopus.Sahara.net.au_DDC_0.bin	07-Apr-10:30:01	2069037056
102.	2006.04.07-08.59.57_Octopus.Sahara.net.au_DDC_1.bin	07-Apr-10:30:01	2069037056
103.	2006.04.07-08.59.57_Octopus.Sahara.net.au_DDC_2.bin	07-Apr-10:30:01	2069037056
104.	2006.04.07-08.59.57_Octopus.Sahara.net.au_DDC_3.bin	07-Apr-10:30:01	2069037056
105.	2006.04.07-09.30.13_Octopus.Sahara.net.au_DDC_0.bin	07-Apr-11:00:07	2082553856
106.	2006.04.07-09.30.13_Octopus.Sahara.net.au_DDC_1.bin	07-Apr-11:00:07	2082553856
107.	2006.04.07-09.30.13_Octopus.Sahara.net.au_DDC_2.bin	07-Apr-11:00:07	2082553856
108.	2006.04.07-09.30.13_Octopus.Sahara.net.au_DDC_3.bin	07-Apr-11:00:07	2082553856
109.	2006.04.07-10.00.25_Octopus.Sahara.net.au_DDC_0.bin	07-Apr-11:29:09	2126561280
110.	2006.04.07-10.00.25_Octopus.Sahara.net.au_DDC_1.bin	07-Apr-11:29:09	2126561280
111.	2006.04.07-10.00.25_Octopus.Sahara.net.au_DDC_2.bin	07-Apr-11:29:09	2126561280
112.	2006.04.07-10.00.25_Octopus.Sahara.net.au_DDC_3.bin	07-Apr-11:29:09	2126561280
113.	2006.04.07-10.29.27_Octopus.Sahara.net.au_DDC_0.bin	07-Apr-11:59:31	2069282816
114.	2006.04.07-10.29.27_Octopus.Sahara.net.au_DDC_1.bin	07-Apr-11:59:31	2069282816
115.	2006.04.07-10.29.27_Octopus.Sahara.net.au_DDC_2.bin	07-Apr-11:59:31	2069282816
116.	2006.04.07-10.29.27_Octopus.Sahara.net.au_DDC_3.bin	07-Apr-11:59:31	2069282816

117.	2006.05.26-04.50.10_Octopus.Sahara.net.au_DDC_0.bin	26-May-06:20:15	2066743296
118.	2006.05.26-04.50.10_Octopus.Sahara.net.au_DDC_1.bin	26-May-06:20:15	2066743296
119.	2006.05.26-04.50.10_Octopus.Sahara.net.au_DDC_2.bin	26-May-06:20:15	2066743296
120.	2006.05.26-04.50.10_Octopus.Sahara.net.au_DDC_3.bin	26-May-06:20:16	2066743296
121.	2006.05.26-05.20.36_Octopus.Sahara.net.au_DDC_0.bin	26-May-06:50:41	2068217856
122.	2006.05.26-05.20.36_Octopus.Sahara.net.au_DDC_1.bin	26-May-06:50:41	2068299776
123.	2006.05.26-05.20.36_Octopus.Sahara.net.au_DDC_2.bin	26-May-06:50:41	2068299776
124.	2006.05.26-05.20.36_Octopus.Sahara.net.au_DDC_3.bin	26-May-06:50:41	2068299776
125.	2006.05.26-05.50.58_Octopus.Sahara.net.au_DDC_0.bin	26-May-07:21:03	2067152896
126.	2006.05.26-05.50.58_Octopus.Sahara.net.au_DDC_1.bin	26-May-07:21:03	2067152896
127.	2006.05.26-05.50.58_Octopus.Sahara.net.au_DDC_2.bin	26-May-07:21:03	2067152896
128.	2006.05.26-05.50.58_Octopus.Sahara.net.au_DDC_3.bin	26-May-07:21:03	2067152896
129.	2006.05.26-06.50.01_Octopus.Sahara.net.au_DDC_0.bin	26-May-08:20:04	2070429696
130.	2006.05.26-06.50.01_Octopus.Sahara.net.au_DDC_1.bin	26-May-08:20:04	2070429696
131.	2006.05.26-06.50.01_Octopus.Sahara.net.au_DDC_2.bin	26-May-08:20:04	2070429696
132.	2006.05.26-06.50.01_Octopus.Sahara.net.au_DDC_3.bin	26-May-08:20:04	2070429696
133.	2006.05.26-07.20.23_Octopus.Sahara.net.au_DDC_0.bin	26-May-08:43:11	1687224320
134.	2006.05.26-07.20.23_Octopus.Sahara.net.au_DDC_1.bin	26-May-08:43:11	1687224320
135.	2006.05.26-07.20.23_Octopus.Sahara.net.au_DDC_2.bin	26-May-08:43:11	1687224320
136.	2006.05.26-07.20.23_Octopus.Sahara.net.au_DDC_3.bin	26-May-08:43:11	1687224320
137.	2006.05.26-07.43.44_Octopus.Sahara.net.au_DDC_0.bin	26-May-08:48:48	373063680
138.	2006.05.26-07.43.44_Octopus.Sahara.net.au_DDC_1.bin	26-May-08:48:48	373063680
139.	2006.05.26-07.43.44_Octopus.Sahara.net.au_DDC_2.bin	26-May-08:48:48	373063680
140.	2006.05.26-07.43.44_Octopus.Sahara.net.au_DDC_3.bin	26-May-08:48:48	373063680
141.	2006.05.26-07.58.51_Octopus.Sahara.net.au_DDC_0.bin	26-May-09:28:55	2068381696
142.	2006.05.26-07.58.51_Octopus.Sahara.net.au_DDC_1.bin	26-May-09:28:55	2068381696
143.	2006.05.26-07.58.51_Octopus.Sahara.net.au_DDC_2.bin	26-May-09:28:55	2068381696
144.	2006.05.26-07.58.51_Octopus.Sahara.net.au_DDC_3.bin	26-May-09:28:55	2068381696
145.	2006.05.26-08.29.28_Octopus.Sahara.net.au_DDC_0.bin	26-May-09:44:29	1111572480

146.	2006.05.26-08.29.28_Octopus.Sahara.net.au_DDC_1.bin	26-May-09:44:29	1111572480
147.	2006.05.26-08.29.28_Octopus.Sahara.net.au_DDC_2.bin	26-May-09:44:29	1111572480
148.	2006.05.26-08.29.28_Octopus.Sahara.net.au_DDC_3.bin	26-May-09:44:29	1111572480
149.	2006.05.26-08.45.09_Octopus.Sahara.net.au_DDC_0.bin	26-May-10:15:11	2073542656
150.	2006.05.26-08.45.09_Octopus.Sahara.net.au_DDC_1.bin	26-May-10:15:11	2073542656
151.	2006.05.26-08.45.09_Octopus.Sahara.net.au_DDC_2.bin	26-May-10:15:11	2073542656
152.	2006.05.26-08.45.09_Octopus.Sahara.net.au_DDC_3.bin	26-May-10:15:11	2073542656
153.	2006.05.26-09.19.46_Octopus.Sahara.net.au_DDC_0.bin	26-May-10:23:55	306462720
154.	2006.05.26-09.19.46_Octopus.Sahara.net.au_DDC_1.bin	26-May-10:23:55	306462720
155.	2006.05.26-09.19.46_Octopus.Sahara.net.au_DDC_2.bin	26-May-10:23:55	306462720
156.	2006.05.26-09.19.46_Octopus.Sahara.net.au_DDC_3.bin	26-May-10:23:55	306462720
157.	2006.05.26-09.24.27_Octopus.Sahara.net.au_DDC_0.bin	26-May-10:47:07	1678950400
158.	2006.05.26-09.24.27_Octopus.Sahara.net.au_DDC_1.bin	26-May-10:47:07	1678950400
159.	2006.05.26-09.24.27_Octopus.Sahara.net.au_DDC_2.bin	26-May-10:47:07	1678950400
160.	2006.05.26-09.24.27_Octopus.Sahara.net.au_DDC_3.bin	26-May-10:47:07	1678950400
161.	2006.05.26-09.49.03_Octopus.Sahara.net.au_DDC_0.bin	26-May-11:19:09	2067972096
162.	2006.05.26-09.49.03_Octopus.Sahara.net.au_DDC_1.bin	26-May-11:19:09	2067972096
163.	2006.05.26-09.49.03_Octopus.Sahara.net.au_DDC_2.bin	26-May-11:19:09	2067972096
164.	2006.05.26-09.49.03_Octopus.Sahara.net.au_DDC_3.bin	26-May-11:19:09	2067972096
165.	2006.05.26-10.23.20_Octopus.Sahara.net.au_DDC_0.bin	26-May-11:24:23	75448320
166.	2006.05.26-10.23.20_Octopus.Sahara.net.au_DDC_1.bin	26-May-11:24:23	75448320
167.	2006.05.26-10.23.20_Octopus.Sahara.net.au_DDC_2.bin	26-May-11:24:23	75448320
168.	2006.05.26-10.23.20_Octopus.Sahara.net.au_DDC_3.bin	26-May-11:24:23	75448320
169.	2006.05.26-10.25.16_Octopus.Sahara.net.au_DDC_0.bin	26-May-11:26:21	79462400
170.	2006.05.26-10.25.16_Octopus.Sahara.net.au_DDC_1.bin	26-May-11:26:21	79462400
171.	2006.05.26-10.25.16_Octopus.Sahara.net.au_DDC_2.bin	26-May-11:26:21	79462400
172.	2006.05.26-10.25.16_Octopus.Sahara.net.au_DDC_3.bin	26-May-11:26:21	79462400
173.	2006.05.26-10.26.32_Octopus.Sahara.net.au_DDC_0.bin	26-May-11:27:36	78069760
174.	2006.05.26-10.26.32_Octopus.Sahara.net.au_DDC_1.bin	26-May-11:27:36	78069760

175.	2006.05.26-10.26.32_Octopus.Sahara.net.au_DDC_2.bin	26-May-11:27:36	78069760
176.	2006.05.26-10.26.32_Octopus.Sahara.net.au_DDC_3.bin	26-May-11:27:36	78069760

Tables B.1 and B.2 describe these signals and their transmission schedules along with noteworthy events. Three digital HF signals (*i.e.* Stanag 4285, Mil-Std-188-110A FSK Wide, and Mil-Std-188-110A FSK Narrow) and two voice signals comprise the transmissions. The voice dialogue of programs D and E is from Craig (2001) and quoted here.

> It was on the 12^{th} hour of the 12^{th} day of the 12^{th} month that Marconi received the first transatlantic radio signal on Signal Hill in 1901. His claim was met with some skepticism; Edison thought Marconi may have heard static instead of signals and the scientists, unaware of the ionosphere, thought the earth's curvature would preclude distant transmission of radio signals.
>
> The F layers of the ionosphere reflect short waves and the D layer absorbs long waves during the hours of daylight which was when the signals were received. It is unlikely, therefore, that long waves would propagate across the Atlantic, and very unlikely that, even under exceptional conditions, long waves could be detected with the apparatus Marconi was using. If he detected the signals from Poldhu, they would have to be short waves, probably in the upper limit of the 5-15 MHz range, bearing in mind the path and the prevailing solar minimum of 1901.
>
> Contemporary references to the 1901 Poldhu transmitter have specified the frequency of the Poldhu transmitter was about 800 kHz, others have placed it around 100 kHz; there is much uncertainty in the historical accounts. This is further complicated by the unreliability with which frequency measurements could be determined in 1901. To overcome this uncertainty, Dr. Jack Belrose, VE2CV, did extensive work with scale and numerical models. The two methods gave moderately consistent results for the frequency response of the fan-shaped Poldhu aerial, but suggested a resonant frequency of the antenna of about 850 kHz.
>
> In addressing the Royal Institution, Marconi stated he could hear the Poldhu signals using untuned detectors, but not with a tuned receiver. Marconi attributed this to 'the varying capacity of the aerial wire' as the kite supporting it moved about in the wind. Two months later, he determined that the maximum distance over the Atlantic that the Poldhu transmitter could be detected with a tuned long wave receiver during the daytime was about 700 miles. This suggests that long waves were not detected in Newfoundland in December of 1901. Short waves would propagate across the Atlantic and would be rejected with a receiver tuned to long waves. Hence the historical information is consistent with the short waves comprising any signals that may have been detected.
>
> Whether there was enough radiation in the short wave spectrum to produce audible signals in St. John's without the aid of amplification is a point of debate amongst radio

scientists. Dr. Belrose has argued that the amount of short-wave radiation from Poldhu was not sufficient to be detected with the apparatus Marconi was using. Other authors have made a case in favour of Marconi.

In the final analysis, we will never know for certain because the accuracy at which the frequency response of the Poldhu transmitter and aerial system can be determined is dependent on how well the details of its construction and operation are known. Most authors acknowledge this information is very sparse and often unreliable. This works in Marconi's favour: it cannot be proved conclusively that the system did not have a parasitic response in the HF region. Also, we must bear in mind that until Marconi achieved transatlantic wireless communications beyond a shadow of doubt in Glace Bay one year later, it was thought impossible. Science often does a better job saying what is possible than what is not, and at its most fundamental level, can tell us only about probabilities. Marconi was lucky to have received the signals. It may have been improbable, but not impossible.

Table B.1. Skywave recordings made at Swan Reach SA 6-7 April 2006. Skywave recordings made at Swan Reach SA 6-7 April 2006. Details of items in the table are included in the *information box* following the table.

Date	Time (start:stop)	Transmit Frequency USB (MHz)	Signal Program	Gain Setting (dB)	Note
06/04/06	1657:1658	15.00	No Signal	36	Only 1 min of data to test operation of receiver.
	1718:1728	13.194	A	36	
	1729:1759	11.401	C	36 / 30	Tx start @ 1737 Gain dropped to 30 @ 1745 hr.
	1800:1830	8.122	D	30	
	1830:1900	10.213	A	30	
	1902:1931	13.194	C	24	
	1932:1959	11.401	D	24	
	2000:2030	8.122	A	30	
	2031:2100	10.213	C	30	
	2100:2105	11.401	No signal	24	Tuned to wrong channel.
	2105:2129	13.194	D	24	Ionosphere unsuitable communications during this period – data kept for reference.
	2130:2157	11.401	A	24 / 18	Gain dropped to 18 @ 2130 hr.
	2159:2228	8.122	C	18	Transmission started at 2200 hr.
	2230:2259	10.213	D	18	Tx started at 2231, FSK co-channel present on 10.213 MHz.
07/04/06	0431:0459	4.600	C	30	Transmission started at 0430 hr.
	0459:0528	6.378	E	30	Quiet at start, more co-channel, voice from 0500-0510 hr and 0513-0523 hr.
	0529: -	8.122	A	30	Transmission starts @ 0532 hr.
	0559:0621	10.213	C	24	Transmission starts @ 0601 hr.
	0629: -	13.194	No signal	-	Wrong channel.
	0632:0659	4.600	E	24	Voice stops @ 0641 hr, restarted later.
	0659:0729	6.378	A	30	
	0730:0734	8.122	No signal	30	Weak signal, wrong channel.
	0734:0759	10.213	C	30	D-region causing low signal levels on gain box.
	0759: -	13.194	E	36	Voice @ 0800-0809 hr, and 0822-0825 hr.
	0829:0859	11.401	A	36	Transmission starts 0831 hr.
	0859: -	8.122	C	36	Transmission starts 0901 hr.
	0930:1000	10.213	E	36	Voice from 0931-0941 hr.
	1000:1029	13.194	A	36	Transmission starts 1000 hr.
	1029:1059	11.401	C	36	Transmission starts 1031 hr.

Table B.1 Note:

⇒ With respect to Adelaide SA, transmit sites are in north-eastern Australia, northern Australia, north-western Australia, and south-eastern Australia. During each data collection session, all sites broadcast the same program simultaneously.

⇒ Signal programs are transmitted by keying an USB HF transmitter with the .wav files representing the five signal programs A, B, C, D, and E.

⇒ At the receiver, signal program E (Chinese voice) is received intermittently as a result of a transmission problem at the transmit sites. The intermittent reception occurs only with program E.

⇒ Non-existent information is indicated by the "–" placeholder.

⇒ Data files attributed to antenna 3 (DDC_2) contain weak signals compared to all other data files. The fault was traced to a defective balun.

⇒ The data sets collected from the eight antennas comprise over 390 GB of information, of which only 49 GB are processed to generate noise PDFs. That is, data from only one antenna – Antenna 5 from Figure 6.9.

Table B.2. Skywave recordings made at Swan Reach SA 25-26 May 2006. Skywave recordings made at Swan Reach SA 25-26 May 2006. Details of items in the table are included in the *information box* following the table.

Date	Time (start:stop)	Transmit Frequency USB (MHz)	Signal Program (Tx Power)	Gain Setting (dB)	Note
25/05/06	2025:2100	7.526	$\vec{NW}$(4 kW);$\vec{E}$(1 kW); $\vec{N}$(2.5 kW)	36	
	2100:2115	7.526	$\vec{NW}$(4 kW);$\vec{E}$(1 kW); $\vec{N}$(2.5 kW)	36	
26/05/06	0449:0519	6.742	$\vec{NW}$(1 kW);$\vec{E}$(1 kW); $\vec{N}$(1 kW)	36	Transmitters not all operational.
	0520:0550	5.717	$\vec{NW}$(1 kW);$\vec{E}$(1 kW); $\vec{N}$(1 kW)	36	All transmitters operational.
	0550:0620	4.721	$\vec{NW}$(1 kW);$\vec{E}$(1 kW); $\vec{N}$(1 kW)	36	
	0621:0649	10.212	No test signals	36	Co-channel morse, radar @ 0640 hr
	0649:0719	11.217	$\vec{NW}$(1 kW);$\vec{E}$(1 kW); $\vec{N}$(1 kW)	36	
	0720:0743	11.217	$\vec{NW}$(4 kW);$\vec{E}$(1 kW); $\vec{N}$(1 kW)	30	$\vec{NW}$ increases power to 4 kW
	0743:0748	11.1	No test signals	30	radar @ 11.130 MHz tone @ 11.030 MHz, voice @ 11.084 MHz, carrier @ 11.103 MHz, signal @ 11.166 MHz
	0758:0828	13.2	$\vec{NW}$(4 kW);$\vec{E}$(1 kW); $\vec{N}$(1 kW)	130	
	0829:0844	13.2	$\vec{NW}$(4 kW);$\vec{E}$(1 kW); $\vec{N}$(1 kW)	30	
	0845:0915	14.684	$\vec{NW}$(4 kW);$\vec{E}$(1 kW); $\vec{N}$(1 kW)	36	
	0919:0923	7.526	$\vec{NW}$(4 kW)	36	Ebor signal only.
	0924:0946	10.212	$\vec{NW}$(4 kW);$\vec{E}$(1 kW); $\vec{N}$(1 kW)	36	Ebor signal drops out.
	0949:1019	18.433	$\vec{NW}$(4 kW);$\vec{E}$(1 kW); $\vec{N}$(1 kW)	36	Ebor signal drops out; returns @ 1000 hr.
	1023:1024	25	Car ignition	36	Arcing against engine.
	1025:1026	14.5	Car ignition	36	Arcing against engine.
	1026:1027	3	Car ignition	36	Arcing against engine.

Table B.2 Note:

⇒ With respect to Adelaide SA, transmit sites are in eastern Australia, northern Australia, and north-western Australia. Directions are indicated in the table by $\vec{\text{E}}$, $\vec{\text{N}}$, and $\vec{\text{NW}}$ respectively. Transmitter powers are indicated by, for example, (1 kW).

⇒ Signal programs are transmitted by keying an USB HF transmitter. The signals from the three transmit sites are:

$\vec{\text{NW}}$: .wav file (recorded in 16-bit mono at 11,025 Hz) containing a concatenation of the following signal programs A, B, C, D, and E

$\vec{\text{N}}$: 300 baud PSK

$\vec{\text{E}}$: 1200 baud PSK

⇒ The data sets collected from the eight antennas comprise over 156 GB of information, of which only 19 GB are processed to generate noise PDFs. That is, data from only one antenna – Antenna 5 from Figure 6.9.

B.2 Data Set for Part III

Four different modulation schemes are transmitted by the HF modem(BAE Systems 2002) on 15.824 MHz USB (upper side-band): FSK Narrow (MIL-STD-188-110A), FSK Wide (MIL-STD-188-110A), FSK Alternate Wide (MIL-STD-188-110A), and Stanag 4285. See the ARM-9401 User Manual (BAE Systems 2002) for more information on the modulation types. Two different signal bandwidths in the narrowband receiver are also used: 3 kHz and 6 kHz. The 3 kHz bandwidth allows only the signal of interest to be captured while the 6 kHz bandwidth allows the signal of interest plus noise and/or interference in the adjacent channel to be captured. Furthermore, each modulation type is transmitted at various transmit levels: -30 dBm, -24 dBm, -18 dBm, -12 dBm, and -6 dBm.

Recordings of the real HF signals are compressed in archival files having names following the format (see Table B.3):

<modulation>_<power level>_<baud rate>_<interleaving>.tar.gz.

The modulation string is either "fn" for FSK Narrow, "fw" for FSK Wide, "faw" for FSK Alt. Wide, or "s4" for Stanag 4285. Note that "x" represents 6, 12, 18, 24, or 30 corresponding to -6, -12, -18, -24, and -30 dBm transmit powers respectively. The baud rate is either 75, 600, 2400, or 3600, though only the 75 baud data files are used in the thesis. The Stanag 4285 signals either have long or short interleaving. Signals in the data files have, by design of the receiver, a 3 kHz filter applied before being recorded. Some files, however, have a suffix "_6k.tar.gz" that indicates the data is collected with a 6 kHz filter. Also note, that the data stored in these compressed files are 32-bit integers of which the upper 20-bits are data. The upper nibble of the remaining 12-bits is a tag indicating from which DDC the sample originated, and the lower 8 bits are set to 1 and are unused. The format of the samples is the same as that described in Figure 6.7.

Table B.3. Files of real HF signals. Recordings of the real HF signals are compressed in archival files having names following the format of "<modulation>_<power level>_<baud rate>_<interleaving>.tar.gz". The modulation string is either "fn" for FSK Narrow, "fw" for FSK Wide, "faw" for FSK Alt. Wide, or "s4" for Stanag 4285. Note that "x" represents 6, 12, 18, 24, or 30 corresponding to -6, -12, -18, -24, and -30 dBm transmit powers respectively. The baud rate is either 75, 600, 2400, or 3600, though only the 75 baud data files are used in the thesis. The Stanag 4285 signals either have long or short interleaving.

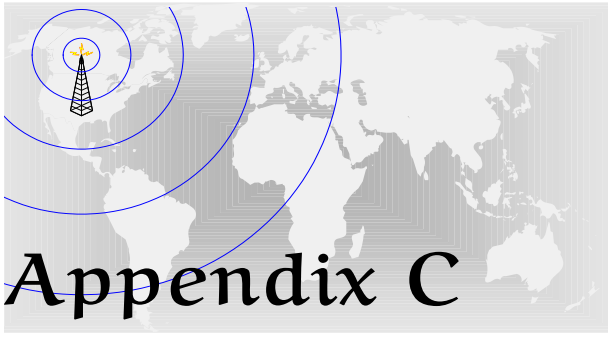
Modulation	Specifics	Filenames
FSK Narrow	75 baud, no interleaving, mark 2762.5 Hz, space 2847.5 Hz, mark hold -40 dBm	fn_x_75b.tar.gz
FSK Wide	75 baud, no interleaving, mark 1575 Hz, space frequency 2425 Hz, mark hold -40 dBm	fw_x_75b.tar.gz
	600 baud, no interleaving, mark 1575 Hz, space frequency 2425 Hz, mark hold -40 dBm	fw_x_600b.tar.gz
FSK Alt. Wide	75 baud, no interleaving, mark 1915 Hz, space frequency 2085 Hz, mark hold -40 dBm	faw_x_75b.tar.gz
Stanag 4285	75 baud, long interleaving	s4_x_75b_long.tar.gz
	600 baud, long interleaving	s4_x_600b_long.tar.gz
	2400 baud, short interleaving	s4_x_2400b_short.tar.gz
	3600 baud, uncoded	s4_x_3600b.tar.gz

The list of files below is here for reference only. In the first column is the filename. The second column indicates the date and time of the start of the recording, while the third column represents the number of bytes in each file. Recordings occur as needed and do not follow a schedule.

1. faw_12_75b.tar.gz 02-Dec-2003 12:40:26 3935601
2. faw_12_75b_6k.tar.gz 02-Dec-2003 14:31:44 4201364
3. faw_18_75b.tar.gz 02-Dec-2003 12:39:10 4089034
4. faw_18_75b_6k.tar.gz 02-Dec-2003 14:29:36 4319629
5. faw_24_75b.tar.gz 02-Dec-2003 12:37:48 4100410
6. faw_24_75b_6k.tar.gz 02-Dec-2003 14:28:34 4471064
7. faw_30_75b.tar.gz 02-Dec-2003 12:36:10 3162554
8. faw_30_75b_6k.tar.gz 02-Dec-2003 14:27:28 3809110
9. fn_12_75b.tar.gz 02-Dec-2003 12:11:04 4246280
10. fn_12_75b_6k.tar.gz 02-Dec-2003 14:10:36 5069837

11.	fn_18_75b.tar.gz	02-Dec-2003 12:08:22	3990353
12.	fn_18_75b_6k.tar.gz	02-Dec-2003 14:08:52	4859494
13.	fn_24_75b.tar.gz	02-Dec-2003 12:06:54	2658678
14.	fn_24_75b_6k.tar.gz	02-Dec-2003 13:53:18	3635717
15.	fn_30_75b.tar.gz	02-Dec-2003 12:04:26	1772703
16.	fn_30_75b_6k.tar.gz	02-Dec-2003 13:51:20	2670312
17.	fn_6_75b.tar.gz	02-Dec-2003 12:15:26	4321401
18.	fn_6_75b_6k.tar.gz	02-Dec-2003 14:11:58	5118272
19.	fw_12_600b.tar.gz	02-Dec-2003 12:31:42	5104652
20.	fw_12_600b_6k.tar.gz	02-Dec-2003 14:25:24	5294932
21.	fw_12_75b.tar.gz	02-Dec-2003 12:23:54	4608598
22.	fw_12_75b_6k.tar.gz	02-Dec-2003 14:18:42	4936212
23.	fw_18_600b.tar.gz	02-Dec-2003 12:30:12	5123462
24.	fw_18_600b_6k.tar.gz	02-Dec-2003 14:24:12	5329388
25.	fw_18_75b.tar.gz	02-Dec-2003 12:22:00	4774576
26.	fw_18_75b_6k.tar.gz	02-Dec-2003 14:17:32	4905162
27.	fw_24_600b.tar.gz	02-Dec-2003 12:29:00	5029884
28.	fw_24_600b_6k.tar.gz	02-Dec-2003 14:22:40	5243527
29.	fw_24_75b.tar.gz	02-Dec-2003 12:20:40	4435209
30.	fw_24_75b_6k.tar.gz	02-Dec-2003 14:16:12	4672715
31.	fw_30_600b.tar.gz	02-Dec-2003 12:25:52	4394864
32.	fw_30_600b_6k.tar.gz	02-Dec-2003 14:20:08	4681620
33.	fw_30_75b.tar.gz	02-Dec-2003 12:18:54	3387152
34.	fw_30_75b_6k.tar.gz	02-Dec-2003 14:13:36	3988466
35.	s4_12_2400b_short.tar.gz	02-Dec-2003 13:03:46	5547739
36.	s4_12_2400b_short_6k.tar.gz	02-Dec-2003 14:55:26	5711349
37.	s4_12_3600b.tar.gz	02-Dec-2003 13:09:44	5524386
38.	s4_12_3600b_6k.tar.gz	02-Dec-2003 14:58:06	5695307
39.	s4_12_600b_long.tar.gz	02-Dec-2003 12:56:10	5555388

40. s4_12_600b_long_6k.tar.gz 02-Dec-2003 14:42:28 5720910
41. s4_12_75b_long.tar.gz 02-Dec-2003 12:47:34 5535060
42. s4_12_75b_long_6k.tar.gz 02-Dec-2003 14:37:14 5709378
43. s4_18_2400b_short.tar.gz 02-Dec-2003 13:01:12 5489746
44. s4_18_2400b_short_6k.tar.gz 02-Dec-2003 14:54:28 5697692
45. s4_18_600b_long.tar.gz 02-Dec-2003 12:54:48 5514402
46. s4_18_600b_long_6k.tar.gz 02-Dec-2003 14:41:32 5681769
47. s4_18_75b_long.tar.gz 02-Dec-2003 12:45:42 5538388
48. s4_18_75b_long_6k.tar.gz 02-Dec-2003 14:36:00 5689085
49. s4_24_2400b_short.tar.gz 02-Dec-2003 12:59:54 5478093
50. s4_24_2400b_short_6k.tar.gz 02-Dec-2003 14:53:14 5633437
51. s4_24_600b_long.tar.gz 02-Dec-2003 12:52:36 5460095
52. s4_24_600b_long_6k.tar.gz 02-Dec-2003 14:40:16 5643072
53. s4_24_75b_long.tar.gz 02-Dec-2003 12:44:28 5479993
54. s4_24_75b_long_6k.tar.gz 02-Dec-2003 14:35:02 5640031
55. s4_30_2400b_short.tar.gz 02-Dec-2003 13:05:54 5316608
56. s4_30_2400b_short_6k.tar.gz 02-Dec-2003 14:51:54 5463577
57. s4_30_3600b.tar.gz 02-Dec-2003 13:07:28 5280513
58. s4_30_3600b_6k.tar.gz 02-Dec-2003 14:57:10 5497048
59. s4_30_600b_long.tar.gz 02-Dec-2003 12:50:36 5348735
60. s4_30_600b_long_6k.tar.gz 02-Dec-2003 14:49:08 5504818
61. s4_30_75b_long.tar.gz 02-Dec-2003 12:42:10 5296659
62. s4_30_75b_long_6k.tar.gz 02-Dec-2003 14:33:44 5467092



Appendix C

Cascade Analysis of Gain Control System

DETERMINING the noise figure of the gain control system involves analyzing the various stages in the cascade of system components. Typically this is done with the help of the Friis equation (Kraus 1966).

Table C.1. Overall noise figure for the wideband gain control system. Applying Friis' equation to the stages in Table C.2 yields the overall noise figure values in this table. The first column contains the allowable gain settings for the gain control system.

Configured Gain (dB)	Noise Factor	Noise Figure (dB)
-6	4.4	6.5
0	1.3	1.2
6	16.9	12.3
12	4.4	6.5
18	2.4	3.8
24	2.3	3.7
30	2.4	3.8
36	2.4	3.8

In a cascade of gain and loss stages in any communication system, the Friis equation (Kraus 1966) can be used to calculate the overall noise factor such that

$$F = F_1 + \frac{F_2 - 1}{G_1} + \frac{F_3 - 1}{G_1 G_2} + \frac{F_4 - 1}{G_1 G_2 G_3} + \ldots, \tag{C.1}$$

where F is the total noise factor, F_n is the noise factor of the n^{th} stage, and G_n is the gain of the n^{th} stage. Note that Friis' equation requires absolute noise factors and gains, not logarithmic values. The noise figure is then the logarithm of the overall noise factor,

$$\text{NF} = 10 \log_{10}(F). \tag{C.2}$$

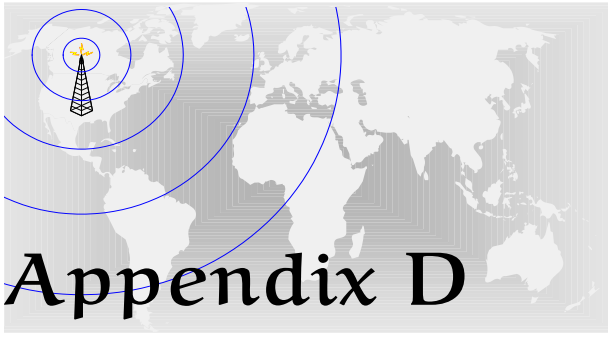
To determine the overall noise figure of the wideband gain control system Friis' equation is necessary. Table C.1 lists the results of applying Friis' equation to the stages in Table C.2, which highlights the manufacturer's stated noise factors and gains for each stage of the gain control system.

Table C.2. Gains & noise figures for the wideband gain control system. Manufacturer's stated gains and noise figures for modules of the wideband gain control system. The first column contains the allowable gain settings for the gain control system.

Configured Gain(dB)	Manufacturer's stated gains (dB) & noise figures (dB) for stages in the Gain Control System											
	2 MHz HPF*		32 MHz LPF*		-20 dB Coupler#		24 dB Amp.		6 dB Atten.		12 dB Amp.	
	Gain	NF	Gain	NF	Gain	NF	Gain	NF	Gain	NF	Gain	NF
-6	-0.1	0.1	-0.1	0.1	0	1	0	0	-6	6	0	0
0	-0.1	0.1	-0.1	0.1	0	1	0	0	0	0	0	0
6	-0.1	0.1	-0.1	0.1	0	1	0	0	-6	6	12	6
12	-0.1	0.1	-0.1	0.1	0	1	0	0	0	0	12	6
18	-0.1	0.1	-0.1	0.1	0	1	24	3	-6	6	0	0
24	-0.1	0.1	-0.1	0.1	0	1	24	3	0	0	0	0
30	-0.1	0.1	-0.1	0.1	0	1	24	3	-6	6	12	6
36	-0.1	0.1	-0.1	0.1	0	1	24	3	0	0	12	6
	Stage 1		Stage 2		Stage 3		Stage 4		Stage 5		Stage 6	

*Values vary slightly with frequency. #Values are assumed based on knowledge of system.

Configured Gain(dB)	Linear gains & noise factors calculated from manufacturer's stated values above											
	2 MHz HPF		32 MHz LPF		-20 dB Coupler#		24 dB Amp.		6 dB Atten.		12 dB Amp.	
	Gain	NF	Gain	NF	Gain	NF	Gain	NF	Gain	NF	Gain	NF
-6	0.98	1.02	0.98	1.02	1.00	1.26	1.00	1.00	0.25	3.98	1.00	1.00
0	0.98	1.02	0.98	1.02	1.00	1.26	1.00	1.00	1.00	1.00	1.00	1.00
6	0.98	1.02	0.98	1.02	1.00	1.26	1.00	1.00	0.25	3.98	15.85	3.98
12	0.98	1.02	0.98	1.02	1.00	1.26	1.00	1.00	1.00	1.00	15.85	3.98
18	0.98	1.02	0.98	1.02	1.00	1.26	251.19	2.00	0.25	3.98	1.00	1.00
24	0.98	1.02	0.98	1.02	1.00	1.26	251.19	2.00	1.00	1.00	1.00	1.00
30	0.98	1.02	0.98	1.02	1.00	1.26	251.19	2.00	0.25	3.98	15.85	3.98
36	0.98	1.02	0.98	1.02	1.00	1.26	251.19	2.00	1.00	1.00	15.85	3.98
	Stage 1		Stage 2		Stage 3		Stage 4		Stage 5		Stage 6	



Appendix D

ITU Predicted versus Measured Noise Levels

THE ITU report (CCIR 1986) on atmospheric noise predicts the natural HF noise level at any location on earth. The procedure involves selecting a mean antenna noise figure of a lossless short vertical antenna over a perfectly conducting ground plane for a particular location on earth. Then from the charts in the report a median antenna noise figure is selected for the frequency of interest and applied by equation Eq. (5.5) to predict the RMS noise field strength in units of dBμV/m. The tables in this appendix contain the predicted atmospheric noise levels for Swan Reach, Australia in autumn of the southern hemisphere. The tables also include the measured RMS noise field strengths from Figures 7.2 to 7.11, though the anomalous RMS noise field strength from Figure 7.10b is not included. In general the measured noise levels are within 10 dB of the ITU predicted levels.

Table D.1. Predicted & measured HF noise levels at Swan Reach, South Australia—06 April 2006. The measured noise levels for Swan Reach, Australia in autumn—06 April 2006—are generally within 10 dB of the ITU predicted noise levels. The predicted noise levels are tabulated under the **Predicted** multi-column. The **Measured** multi-column contains the measured noise field strengths, in units of dBμV/m, from Figures 7.2 to 7.11. The anomalous RMS noise field strength from Figure 7.10b is not included in this table. For both the **Predicted** and **Measured** multi-columns the noise bandwidth is 153.6 kHz.

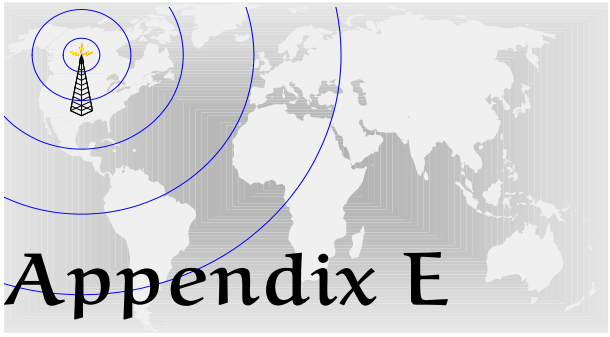
Date (Time)	Centre Frequency F_{MHz}, (MHz)	ITU **Predicted** Noise for Swan Reach in Autumn: Ant. Noise Figure Mean F_{a}, (dB)	Ant. Noise Figure Median F_{am}, (dB)	RMS Noise Field Strength, E_{n} (dBμV/m)	RMS Noise Field Strength, E_{n} (μV/m)	ITU Report Figure Ref.	**Measured** Noise: RMS Noise Field Strength, σ (dBμV/m)	RMS Noise Field Strength, σ (μV/m)	Thesis Figure Ref.	Difference $\lvert\sigma - E_{\text{n}}\rvert$ (dB)	Within 10 dB of ITU?
6-Apr-06	8.12	55	40	14.56	5.34	24a & 24b	25.85	19.60	7.2d)	11.29	N
(1600:2000)	10.20	55	37	13.54	4.75		22.73	13.70	7.3a)	9.20	Y
	11.40	55	35	12.50	4.22		20.09	10.10	7.2c)	7.58	Y
	11.40	55	35	12.50	4.22		18.06	8.00	7.3c)	5.56	Y
	13.20	55	33	11.78	3.88		17.80	7.76	7.2b)	6.02	Y
	13.20	55	33	11.78	3.88		20.42	10.50	7.3b)	8.65	Y
	15.00	55	28	7.89	2.48		13.10	4.52	7.2a)	5.22	Y
6-Apr-06	8.12	70	45	19.56	9.50	25a & 25b	26.89	22.10	7.3d)	7.33	Y
(2000:2400)	8.12	70	45	19.56	9.50		24.81	17.40	7.4e)	5.26	Y
	10.20	70	40	16.54	6.71		21.87	12.40	7.4a)	5.33	Y
	10.20	70	40	16.54	6.71		18.54	8.45	7.4f)	2.00	Y
	11.40	70	36	13.50	4.73		15.50	5.96	7.4b)	2.00	Y
	11.40	70	36	13.50	4.73		10.13	3.21	7.4d)	3.37	Y
	13.20	70	32	10.78	3.46		11.50	3.76	7.4c)	0.73	Y

Table D.2. Predicted & measured HF noise levels at Swan Reach, South Australia—07 April 2006. The measured noise levels for Swan Reach, Australia in autumn—07 April 2006—are generally within 10 dB of the ITU predicted noise levels. The predicted noise levels are tabulated under the **Predicted** multi-column. The **Measured** multi-column contains the measured noise field strengths, in units of dBμV/m, from Figures 7.2 to 7.11. The anomalous RMS noise field strength from Figure 7.10b is not included in this table. For both the **Predicted** and **Measured** multi-columns the noise bandwidth is 153.6 kHz.

Date (Time)	Centre Frequency F_{MHz}, (MHz)	ITU **Predicted** Noise for Swan Reach in Autumn: Ant. Noise Figure Mean F_a, (dB)	Ant. Noise Figure Median F_{am}, (dB)	RMS Noise Field Strength, E_n (dBμV/m)	RMS Noise Field Strength, E_n (μV/m)	ITU Report Figure Ref.	**Measured** Noise: RMS Noise Field Strength, σ (dBμV/m)	RMS Noise Field Strength, σ (μV/m)	Thesis Figure Ref.	Difference $\lvert\sigma - E_n\rvert$ (dB)	Within 10 dB of ITU?
7-Apr-06	4.60	60	47	16.62	6.78	21a & 21b	31.39	37.10	7.5a)	14.77	N
(0400:0800)	4.60	60	47	16.62	6.78		25.15	18.10	7.6b)	8.53	Y
	6.38	60	44	16.46	6.65		26.11	20.20	7.5b)	9.65	Y
	6.38	60	44	16.46	6.65		23.17	14.40	7.6c)	6.71	Y
	8.12	60	39	13.56	4.76		19.22	9.14	7.5c)	5.66	Y
	8.12	60	39	13.56	4.76		14.98	5.61	7.6d)	1.42	Y
	10.20	60	34	10.54	3.36		17.68	7.66	7.5d)	7.15	Y
	10.20	60	34	10.54	3.36		15.31	5.83	7.7a)	4.78	Y
	13.20	60	27	5.78	1.94		0.75	1.09	7.6a)	5.03	Y
7-Apr-06	8.12	20	27	1.56	1.20	22a & 22b	21.87	12.40	7.7d)	20.31	N
(0800:1200)	10.20	20	28	4.54	1.69		7.16	2.28	7.8a)	2.62	Y
	11.40	20	27	4.50	1.68		13.35	4.65	7.7c)	8.85	Y
	11.40	20	27	4.50	1.68		17.17	7.22	7.8c)	12.67	N
	13.20	20	26	4.78	1.73		9.74	3.07	7.7b)	4.97	Y
	13.20	20	26	4.78	1.73		8.91	2.79	7.8b)	4.14	Y

Table D.3. Predicted & measured HF noise levels at Swan Reach, South Australia—26 May 2006. The measured noise levels for Swan Reach, Australia in autumn—26 May 2006—are generally within 10 dB of the ITU predicted noise levels. The predicted noise levels are tabulated under the **Predicted** multi-column. The **Measured** multi-column contains the measured noise field strengths, in units of dBμV/m, from Figures 7.2 to 7.11. The anomalous RMS noise field strength from Figure 7.10b is not included in this table. For both the **Predicted** and **Measured** multi-columns the noise bandwidth is 153.6 kHz.

Date (Time)	Centre Frequency F_{MHz}, (MHz)	ITU **Predicted** Noise for Swan Reach in Autumn: Ant. Noise Figure Mean F_{a}, (dB)	Ant. Noise Figure Median F_{am}, (dB)	RMS Noise Field Strength, E_{n} (dBμV/m)	RMS Noise Field Strength, E_{n} (μV/m)	ITU Report Figure Ref.	**Measured** Noise: RMS Noise Field Strength, σ (dBμV/m)	RMS Noise Field Strength, σ (μV/m)	Thesis Figure Ref.	Difference $\lvert\sigma - E_{\text{n}}\rvert$ (dB)	Within 10 dB of ITU?
26-May-06	4.72	60	47	16.84	6.95	21a & 21b	25.71	19.30	7.9c)	8.87	Y
(0400:0800)	5.72	60	45	16.51	6.69		23.46	14.90	7.9b)	6.95	Y
	6.74	60	43	15.94	6.26		22.48	13.30	7.9a)	6.54	Y
	11.10	60	30	7.27	2.31		30.63	34.00	7.10b)	23.36	N
	11.20	60	30	7.35	2.33		13.37	4.66	7.9d)	6.02	Y
	11.20	60	30	7.35	2.33		15.58	6.01	7.10a)	8.23	Y
26-May-06	7.53	20	26	-0.10	0.99	22a & 22b	17.43	7.44	7.11b)	17.53	N
(0800:1200)	10.20	20	28	4.54	1.69		11.98	3.97	7.11c)	7.44	Y
	13.20	20	26	4.78	1.73		15.93	6.26	7.10c)	11.16	N
	13.20	20	26	4.78	1.73		7.27	2.31	7.10d)	2.50	Y
	14.70	20	25	4.71	1.72		7.99	2.51	7.11a)	3.28	Y
	18.40	20	15	-3.34	0.68		5.67	1.92	7.11d)	9.01	Y



Appendix E

Data Sheets

THIS appendix contains information on where to access manufacturer's datasheets for key components of the narrowband, swept narrowband, and broadband receivers. These components are the active antenna, feeder cable, ADC, DDC, and digital receiver card. The gain control system is a custom system and no datasheet exists for it; likewise, no data sheet exists for the passive antenna used at Swan Reach, South Australia. Chapters 6 and 11 refer to this appendix.

Active Rod Antenna HE011

Manufacturer : ROHDE & SCHWARZ GmbH & Co. KG Mühldorfstraße 15 D-81671 München, Germany, P.O.B. 801469 D-81614 München; http://www.rsd.de

Datasheet URL : http://www.thiecom.de/pdf/he011.pdf

LMR-400 Flexible Low-Loss Communications Coax

Manufacturer : Times Microwave Systems, 358 Hall Avenue, PO Box 5039 Wallingford, CT 06492-5039, USA; http://www.timesmicrowave.com

Datasheet URL : http://www.timesmicrowave.com/content/pdf/lmr/22-25.pdf

AD6645 14-bit, 105 MS/s ADC

Manufacturer : Analog Devices Inc., Corporate Headquarters, Building Three, Three Technology Way, Norwood MA, USA; http://www.analog.com

Datasheet URL : http://www.analog.com/UploadedFiles/Data_Sheets/AD6645.pdf

GC4016 Quad DDC Chip

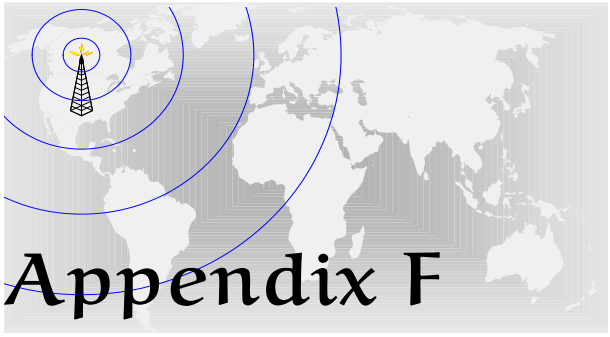
Manufacturer : Texas Instruments, P.O. Box 660199 Dallas TX 75266-0199, USA; http://www.ti.com

Datasheet URL : http://www.ti.com/lit/gpn/gc4016.pdf

ICS554 Digital Receiver Card

Manufacturer : GE-Fanuc Intelligent Platforms, 2500 Austin Drive, Charlottesville VA 22911, USA; http://www.gefanucembedded.com

Datasheet URL : http://www.gefanucembedded.com/products/resources/2052



Appendix F

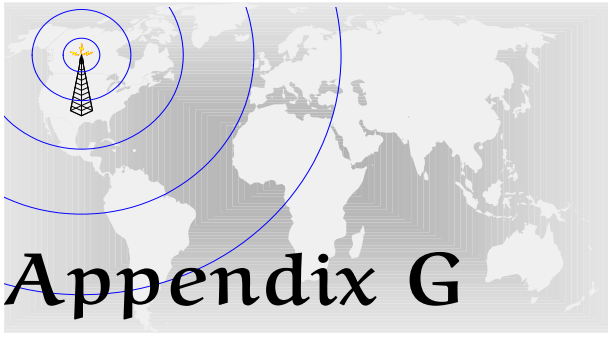
Australian HF Spectrum Allocations

MANAGEMENT of the RF spectrum is necessary because the RF spectrum is a scarce resource; the demand for its use outweighs the supply of available spectrum. The International Telecommunications Union (ITU) governs international spectrum allocations, but in Australia the governing body is the Australian Communications Authority (ACA).

This appendix contains a subset of the allocations made by the ACA. In particular, allocations for the HF band (nominally 3 MHz to 30 MHz).

NOTE: This figure is included on page 325 of the print copy of the thesis held in the University of Adelaide Library.

Figure F.1. Australian HF spectrum allocations. Australian HF spectrum allocations. The allocation of the RF spectrum (a scarce resource) is governed, internationally, by the International Telecommunications Union (ITU) and, in Australia, by the Australian Communications Authority (ACA). This figure is an excerpt from the Australian RF Spectrum Allocations Chart produced by the ACA.



Appendix G

Signs Toolbox Guide

GENERALLY a user's guide for software is comprehensive in its description. The scope of this thesis does not target operational guides. Nevertheless, as the work of this thesis requires the toolbox, a brief description of it is therefore necessary.

G.1 Introduction to Ⓢigns

Results in this thesis are, in part, achieved through a custom **MATLAB**® toolbox called Ⓢ**igns** —so named because modulation recognition involves analysis of signal identifying parameters (or signs) from a received signal in order to determine its modulation type. The Ⓢ**igns** toolbox is useful for observing the performance of a particular parameter against synthetic or real HF signals. Though brief, this guide is intended to help the toolbox user understand the general workings of the Ⓢ**igns** toolbox. A detailed listing of the Ⓢ**igns** toolbox can be found in Appendix H.

Ⓢigns Concept of Operation

The analysis process in the Ⓢ**igns** toolbox follows a staged approach. The output of each stage is the input to the next. The first stage, called the configuration stage, prepares the analysis environment. The output of this stage is an analysis profile that is used by all subsequent stages. The second stage generates binary data sequences for the third stage: transmission. Transmission involves the creation of baseband and passband signals. Stage four applies HF propagation models to the passband signals. The fifth stage receives the propagated signals. A sixth stage extracts features from the received signals. The next stage classifies the features and the last stage reports on findings of the analysis.

The staged approach is beneficial for a number reasons. Each stage performs a specific set of tasks and is therefore, from this perspective, independent of the other stages. Each stage can be run separately, provided output from the previous stage is available, so that parts of the analysis can be repeated without having to repeat the entire analysis (for the processing of recorded signals, only stages 1, 5, 6, and 7 are required). Furthermore, the analysis environment is not passed from stage to stage; rather each stage loads the analysis profile, as well as the output from the previous stage, prior to completing its specified tasks. All profiles and output files are stored in the **MATLAB**® .mat format.

For synthetic signals, usage of Ⓢ**igns** typically involves setting up a configuration module, creation of the analysis profile, and then executing **sign_s.m** which controls the calling of all related toolbox functions. For analysis of real signals, the approach is

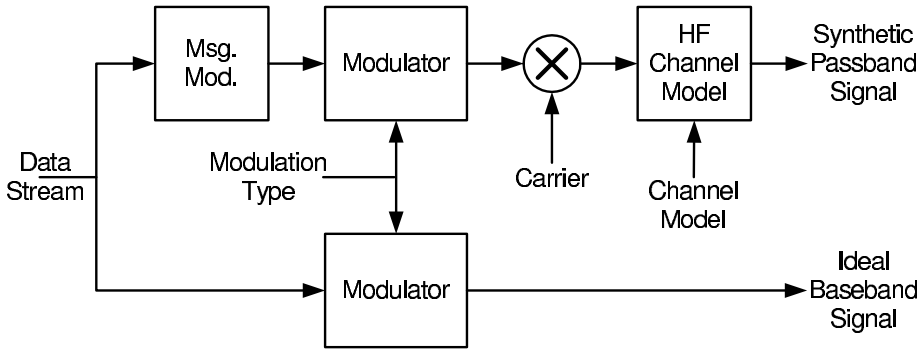


Figure G.1. Transmit section of the Ⓢigns toolbox (repeated). The transmit section of the Ⓢ**igns** toolbox provides two outputs: a synthetic passband signal and an ideal baseband signal. A binary stream is modulated to form an ideal baseband signal. The signal is ideal because it is not affected by the transmission medium. The other branch models the effects of the transmission medium on the signal. The first box allows the user to modify the bit stream (*e.g.* introduce bit errors). The modulator encodes the bit stream in an analog waveform. Upshifting to the carrier frequency occurs next, followed by modification of the passband signal by the HF channel model.

the same, except that the analysis profile causes Ⓢ**igns** to bypass stages 2 to 4. Figures G.1, G.2, and G.3 illustrate the system blocks in this toolset.

The transmit stage takes as an input a binary data stream from the data generation stage. This data stream is modulated with the chosen modulation scheme to produce an ideal baseband reference signal—X in the context of Figure 10.1. The other path generates a synthetic passband signal that ultimately, after downconversion, becomes Y in the context of Figure 10.1. In this path, the data stream can have its message modified to suit the purpose of the investigation (*e.g.* simulation of message variations, bit errors, addition of error-correcting codes, and so on). The modified message is then modulated with the chosen modulation type, upconverted, and then passed through an appropriate HF channel model. The modulator, in the transmitter, is capable of generating a number of baseband signals such as m-ary FSK, m-ary PSK, Stanag 4285 signal, and Mil-Std-188-110A FSK signals. The HF channel model can be configured to follow a user-specified algorithm but, whatever the model, it is important that it account for Doppler shifts, deep fades, impulsive noise (as opposed to Gaussian noise), and multi-modal signals (*i.e.* signals propagating by more than one ionospheric layer).

The receive blocks are designed to reverse the processing applied by the transmitter. However, the receiver also incorporates switches that allow for the synthetic signals

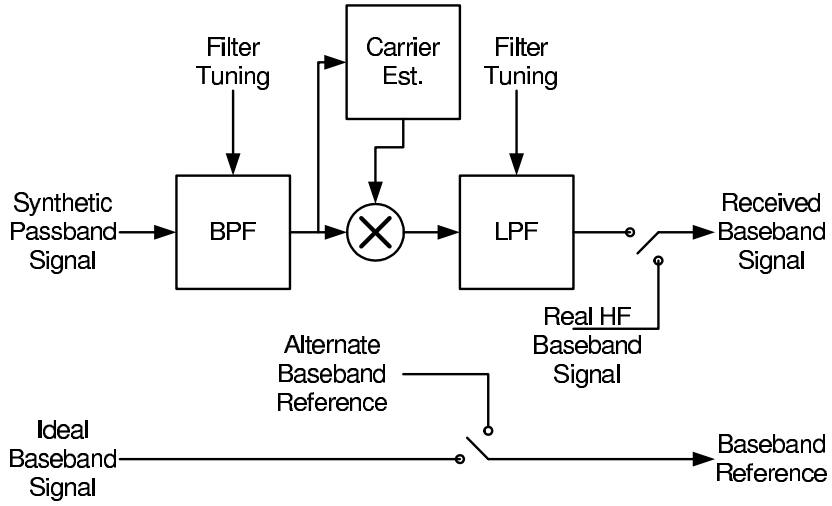


Figure G.2. Receive section of the Ⓢigns toolbox (repeated). The receive section of the Ⓢ**igns** toolbox operates in a number of modes. In the simulation mode, the ideal baseband signal from the transmit section is allowed to pass through with no modification. The synthetic passband signal, however, is filtered and downconverted to baseband. Alternatively, real HF signals can be switched in to replace the synthetic signals in either the baseband reference path, or the path for the received baseband signal.

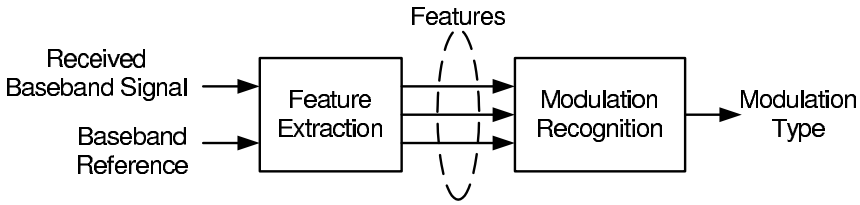


Figure G.3. Modulation recognition section of the Ⓢigns toolbox (repeated). The modulation recognition section of the Ⓢ**igns** toolbox consists of two modules: a feature extraction module and a classification module. Both modules can be customized. Currently, the feature extraction module can generate twelve different signal features. Classification can be implemented any way the user chooses.

to be ignored and for alternate signals (real or synthetic) to be output to the feature extraction stage. For example, one may develop a feature extraction tool, test it on synthetic signals and then apply a recorded real signal to test the robustness of the new tool. The downconversion path provides filtering and carrier estimation. The filters are tuneable to suit the particular HF signal. A carrier extraction function can be programmed with the true carrier frequency, or it can estimate the carrier based on the received signal. Other modules can be added if necessary to support analysis of technically complex signals.

Features can be extracted from simulated or real HF signals in the feature extraction stage. The intention of the feature extraction block is to include algorithms to determine all of these parameters: centre frequency, bandwidth, SNR, coherence, signal envelope, symbol frequencies, cross-Margenau-Hill distribution, kurtosis, signal constellation, auto- and cross- PSD, modulation level, auto-regressive covariance, and entropic distance. However, only a few of these algorithms are currently implemented.

The Ⓢ**igns** toolbox, though sufficient for the work in this thesis, is not complete. The HF Channel Model provides an interface for Watterson's (1969, 1970) narrowband model, Vogler's (1988, 1990, 1992) wideband model, and Lemmon's (1991, 1993) wideband model. None of these models are implemented. The current version of the Ⓢ**igns** toolbox implements a simple additive white-Gaussian noise model for the HF channel. Moreover, the carrier estimate block depicted in Figure 11.5 requires implementation. Additionally, only entropic distance, coherence, and the SNR estimator have are contained in the feature extraction module. Interfaces for the other parameters are provided in the module. Finally, an interface is provided by Ⓢ**igns** that enables a classification module to be added to the toolbox. The current Ⓢ**igns** toolbox does not include a classifier. A simple classification algorithm could include a majority logic algorithm and a thresholding algorithm.

So, with this in mind let us discuss the Ⓢ**igns** file structure and its installation in a **MATLAB**® environment, before attempting to describe the steps in the analysis process.

Ⓢigns Installation Procedure

Installation of the Ⓢ**igns** toolbox is simple. All that is required is that the directory structure of the toolbox remain on the platform to which the toolbox is copied and that the Ⓢ**igns** directory (and its subdirectories) be added to the **MATLAB**® path. The structure is shown in Figure G.4.

The Logs folder is meant to store log files of analysis runs and errors. The toolbox implements a logging function that will record errors and analysis information. Profiles, created from the configuration stage, are stored in the Profiles directory. The RealData directory is meant for the user to store recorded signals. It is not necessary that

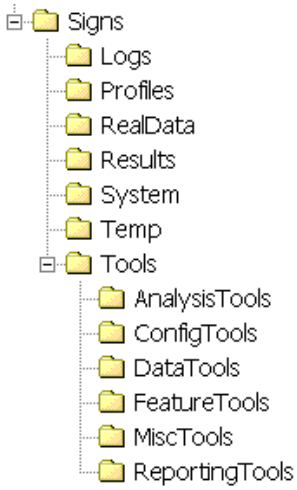


Figure G.4. Directory structure of the Ⓢigns Toolbox. The directory structure of the Ⓢ**igns** toolbox should remain on the platform to which the toolbox is copied. The Tools directory contains all toolbox functions. The System directory contains some executable files. The other folders contain configuration information, logs, real data files (if the user requires), results, and temporary information.

this folder be used to store the signals, but it is nonetheless convenient for the toolbox. The output of each stage is stored in a file in the Results folder. Each subsequent stage will read a file from the Results folder before continuing processing. The System folder contains dynamic link libraries created for the toolbox. The Temp folder is a temporary directory for the toolbox. It is primarily used by the **signs_lzw.m** function when the disk compression option is selected. Finally, the Tools folder contains all the toolbox functions in a series of appropriately named folders.

G.2 Description of the Analysis Process

Each of the following subsections focus on one stage of the analysis process and, collectively, are intended to demonstrate how to setup and run the Ⓢ**igns** toolbox. It is necessary to read this description in conjunction with the Ⓢ**igns** scripts contained on the CD-ROM (packaged with this thesis) or with the listings in Appendix H.

Ⓢigns Stage 1: Configuration

Configuration of the analysis environment involves setting global variables. Each configuration script contains groupings of global variables that correspond to stages in the analysis process. A global variable is easily recognized by its "CFG_" prefix. For example, the script **cfg2fsk.m** defines the global variables for a scenario involving an FSK signal. Other example configuration scripts are listed in Appendix H. In all the example configuration scripts, the definitions for all these global variables are clearly commented.

The output of the configuration script is an analysis profile with a .mat extension. This profile is loaded by each stage prior to execution of its tasks. The name of the profile and other file related variables are located at the end of the script.

At this point the toolbox user should run **cfg2fsk.m**, or any of the other configuration scripts, and choose an appropriate name for the profile when prompted by the save dialog box.

Initiating the analysis can follow two paths. The easy path is to use **sign_s.m**. This script will prompt the user to choose an appropriate profile and will then call each stage in sequence. If a stage reports an error, then analysis will stop. The user can fix the problem and continue the analysis by manually calling each stage, or by re-running **sign_s.m**. This manual method, is of course, the more difficult path.

If an error does occur and it is related to the configuration, the configuration script will need to be re-run. For minor errors, the analysis can continue with the manual method; significant errors will likely necessitate restarting the whole analysis process.

The analysis need not start with **sign_s.m**. Instead, the user may choose to run each module individually to maintain control of the analysis. The script for each stage takes, as an input parameter, a string indicating the path and filename of the analysis profile. We shall take the manual method here.

If the analysis profile is not passed to the **MATLAB**® module for the particular stage, the module will prompt the user for the profile through a load dialog box.

Configuration variables in the analysis profile are critical; it is important to understand the usage of these variables before continuing. One way to do this is to read the code for the configuration files **cfg2fsk.m**, **cfg2psk.m**, and **cfgstanag.m**. Most configuration variables are the same for each script, but a few are specific to the signals and scenarios outlined by each script.

Ⓢigns Stage 2: Data Generation

The **datagen.m** script is the next stage of the process; its purpose is to prepare a binary data stream for input to the transmitter. Symbols in the data stream can be distributed uniformly or normally. Actually, two data streams are created: one for the ideal/reference signal, X, in the common model (see Figure 10.1), and one for the signal passing over the propagation medium (Y in the common model). The latter data stream, by default, is the same as the former unless the analysis profile specifies a Hamming distance greater than zero between Y and X. In this case, **datagen.m** also randomly chooses a number of bits equal to the Hamming distance to invert.

The output of this stage is a .mat file containing all variables in the **datagen.m** having a "SIG_" prefix.

This is the same method that each stage uses to prepare an output file. The **MATLAB**® variables prefixed by "SIG_" are saved to a .mat file.

Key variables in the output file are:

SIG_N which is the number of samples at the sampling rate within the analysis time (this value should be large in order to have good frequency resolution);

SIG_SYM which contains the number of symbols in the analysis time;

SIG_HAM which is Hamming distance between X and Y;

SIG_DLY which is the number of additional symbols required to compensate for the transmission filter delays; and finally

SIG_MSG which contains the binary data streams for X and Y.

The toolbox user should now run **datagen.m** in the **MATLAB**® environment.

Ⓢigns Stage 3: Transmission

The purpose of this stage is to generate two baseband signals. One becomes an ideal reference, while the other is mixed up to the carrier frequency specified by the input profile. The transmission stage is implemented with the **txsim.m** script.

This script can, depending on the settings in the analysis profile, generate m-ary FSK, m-ary PSK, and Stanag 4285 signals. Close inspection of the code for **txsim.m** will reveal a branching structure to allow other modulation types such as AM, FM, static noise (white or coloured noise), or some other custom modulation type.

Key variables in the output file are:

SIG_REF which contains the baseband reference signal, X;

SIG_TxBS which contains the baseband signal to be transmitted over the propagation medium;

SIG_TxIF which is the upconverted signal, Y, prior to filtering; and

SIG_TxRF which is the upconverted signal, Y, after the transmission filters.

The toolbox user should now run **txsim.m** in the **MATLAB**® environment.

Ⓢigns Stage 4: HF Propagation

The purpose of this stage is to apply a model of the propagation medium to the pass-band signal created by the transmission stage. Currently only a white Gaussian noise model is implemented. Branching code in the **rfsim.m** script is available for additional

models. The additional models include the narrowband HF channel model by Watterson (1969, 1970), the wideband models by Vogler (1988, 1990, 1992) and Lemmon (1991, 1993), as well as incorporating the Bi-Kappa distribution for natural HF noise.

As in the previous stages, the output .mat file contains key variables from the propagation stage. However, in this case there is only one variable in the output file, **SIG_RF**, which represents the transmitted signal affected by the propagating medium.

The toolbox user should now run **rfsim.m** in the **MATLAB®** environment.

Ⓢigns Stage 5: Signal Reception

This stage has two functions: receiving the signal output from the previous stage, and formatting of data for the subsequent feature extraction stage. From the output of the propagation stage the **rxsim.m** script creates an ideal baseband signal and a life-like received baseband signal for the parameter extraction routine.

The script for this module, **rxsim.m**, deviates slightly from the staged approach in that it loads the output file from the data generation stage, the transmission stage, as well as the signal propagation stage. There is no other reason for this than to reduce the size of the output file from the propagation stage. The alternative is that all the key output variables from the data generation stage and transmission stage are incorporated in the output from the propagation stage. Depending on the analysis profile, this output file can be large.

The algorithm for the receiver stage follows Figure G.2. Unidentified in the figure is the ability of the mixer and final filter to extract the upper-, lower-, or double-sidebands of the received signal. Moreover, the carrier estimation block in the figure is not implemented. Branches in the script are available for estimation algorithms to be inserted.

Key variables in the output file are:

SIG_RxBS which contains the downconverted baseband signal;

SIG_RxIF which received signal after passband filtering prior to downconversion.

The toolbox user should now run **rxsim.m** in the **MATLAB**® environment.

The next part of the reception stage is the script, **xswitch.m**, that implements the switching depicted in Figure G.2 and formatting of data for the feature extraction stage. If synthetic signals generated by Ⓢ**igns** are not desired, other signals (real or synthetic) can be switched in. Program flow in this script is completely dictated by the analysis profile.

Key variables in the output file of **xswitch.m** are all prefixed by **SIG_**, including those loaded from output files of previous stages.

The script for this module, **xswitch.m**, deviates slightly from the staged approach in that it may load the output file from the transmission stage, as well as the output from **rxsim.m**. This is necessary to implement the signal switching arrangement.

The toolbox user should now run **xswitch.m** in the **MATLAB**® environment.

Ⓢigns Stage 6: Feature Extraction

The purpose of this stage is to extract features from the baseband signals output from the previous stage. Parameter extraction depends on configuration information stored in the analysis profile. Settings of the global variables determine which feature extraction algorithms are executed by **extract.m**.

Coherence, coherence-median-difference (CMD), entropic distance, and power-spectral density estimating algorithms are implemented in the script. There are also branches for instantaneous frequency, instantaneous phase, instantaneous amplitude, and carrier frequency. The intention is that **extract.m** contain a variety of algorithms for extracting signal features; this is a subject of further work.

The output of this stage is a file containing variables prefixed by "EXT_". The key variables in the current version of the script are:

EXT_COH which contains the coherence estimate;

EXT_CMD which is the coherence-median-difference result;

EXT_ENT which is the entropic distance measure; and

EXT_PSD which is the power-spectral density.

The toolbox user should now run **extract.m** in the **MATLAB**® environment.

Ⓢigns Stage 7: Modulation Recognition

Further work is required to implement the modulation recognition stage. A classification algorithm is required to analyze the features extracted from the previous stage. The intention is to include this algorithm in a script called, **express.m**.

Ⓢigns Stage 8: Reporting

The purpose of the reporting stage is to collate and present the results of the analysis. Numerous reporting functions are listed in Appendix H. The **report.m** script allows the toolbox user to select various reports from a selection window. All the reporting functions take the analysis profile as an input as well as select variables from output files from other stages. The variables of choice, of course, depend on the reporting function. The reporting stage does not generate an output file.

There are eleven implemented reporting functions:

Report 1 : **rpt_chhamf.m** produces a surface plot of coherence versus Hamming distance and frequency;

Report 2 : **rpt_chsnrf.m** generates a surface plot of coherence versus SNR and frequency;

Report 3 : **rpt_cmdfrq.m** plots the CMD versus frequency;

Report 4 : **rpt_cohfrq.m** displays a 2-D plot of coherence versus frequency;

Report 5 : **rpt_cohham.m** produces a graph of coherence versus Hamming distance

Report 6 : **rpt_cohsnr.m** provides a plot of coherence versus SNR

Report 7 : **rpt_ensamb.m** generates a surface plot of entropic distance versus number of samples in the main sequence versus number of samples in the appended sequence;

Report 8 : **rpt_entsam.m** graphs entropic distance versus sequence length;

Report 9 : **rpt_gcmdfr.m** provides a plot of generalized[42] coherence-median difference versus frequency;

Report 10 : **rpt_medsam.m** mean entropic distance versus segment length;

Report 11 : **rpt_psds.m** plots the power spectral densities of signals used in the analysis.

In its branching structure, however, the **report.m** script is written to accommodate many more reporting features. These additional features are subjects of further work.

The script for this module, **xswitch.m**, deviates slightly from the staged approach in that it may load the output file from the data generation stage, as well as the output from **extract.m**.

The toolbox user should now run **report.m** in the **MATLAB**® environment.

[42] The generalized CMD is the coherence at every frequency in a specified bandwidth less the median coherence across that bandwidth.

G.3 Ⓢigns Toolbox Reference

The following summarizes the contents of the Ⓢ**igns** toolbox. Appendix H contains program listings of the toolbox.

```
% SIGN(s) Toolbox    Version 1.0
%    Copyright(c)2003,2004,2005,2006,2007,2008 by James Giesbrecht
%
% General information.
%    SIGN(s) is a signal modulation recognition toolbox that can be used
%    to simulate the recognition of contrived signals, or can be used
%    to recognize real signals imported from a data collection system.
%
%    SIGN(s) is so named because the process of modulation recognition
%    is one of extracting %    (i.e. features) from an unknown received
%    signal and then attempting to signify (i.e. recognize) the modulation
%    from the extracted signs.  SIGN(s) is also an acronym for
%    SIGnal modulation aNalysis (SIGN).  The appended (s) signifies that
%    signal modulation analysis requires one or more signs (features) to
%    determine the modulation.
%
% GUI tool configuration.
%    config              - Brings up general GUI for creation of analysis profiles.
%    cfgtxsim            - Brings up GUI for configuration of transmitter simulator.
%    cfgrfsim            - Brings up GUI for configuration of RF channel simulator.
%    cfgrxsim            - Brings up GUI for configuration of receiver simulator.
%    cfgfeat             - Brings up GUI for configuration of feature extraction tool.
%    cfgexpr             - Brings up GUI for configuration of modulation recognition tool.
%
% Data preparation tools.
%    datagen             - Tool for preparation of data for input to an analysis tool.
%    xswitch             - Tool for switching data for the extraction tool.
%
% Analysis tools.
%    txsim               - Basic radio transmiter simulation tool.
%    rfsim               - RF channel simulation tool.
%    rxsim               - Basic radio receiver simulation tool (can accept real signals).
%    extract             - Feature extraction tool.
%    express             - Modulation recognition tool.
%
% Reporting tools.
%    report              - Tool for viewing analysis results
%
% Feature extraction tools.
%    ext_cohr            - Tool for computing the coherence of two signals.
%    ext_cmd             - Tool for extracting the coherence-median difference.
%    ext_psd             - Tool for determining the power-spectral density of a signal.
%    ext_entropy         - Tool for determining signal entropy
%
```

```
% Miscellaneous tools.
%   signs_aisbetthash   - Aisbett hash function (product of the time-based means less the time
    -based covariance)
%   signs_analogout     - Downconverts a signal from a data vector to baseband and creates an
    audio file
%   signs_apwin         - Sliding window tool
%   signs_bikappapdf    - Returns the Bi-kappa distribution
%   signs_cmhd          - Cross-Margenau-Hill distribution
%   signs_daubwavelet   - Returns Daubechies' basic wavelet
%   signs_daubwt        - Returns the Daubechies' wavelet transform of a data sequence
%   signs_decile        - Computes the nth decile
%   signs_dynrange      - Estimates the dynamic range of a vector or 2-D array
%   signs_editpath      - Modification of file paths in a Sign(s) configuration profile
%   signs_engunt        - Returns scaling factor for most appropriate engineering units
%   signs_fsk           - m-ary FSK modulator
%   signs_gausspdf      - Returns the Gaussian distribution
%   signs_haarwt        - 1-D Haar Wavelet Transform
%   signs_haarwt2       - 2-D Haar Wavelet Transform
%   signs_hamdist       - Returns the Hamming distance between two binary sequences
%   signs_hfnoise       - Returns the level of the HF environmental for a particular noise
    model
%   signs_hyper2f1.m    - Computes the confluent hypergeometric function
%   signs_ihaarwt       - 1-D Inverse Haar Wavelet Transform
%   signs_ihaarwt2      - 2-D Inverse Haar Wavelet Transform
%   signs_isdyadic      - Returns true if a number is dyadic
%   signs_log           - Tool for logging/reading messages
%   signs_lzw           - LZW compression tool
%   signs_mixer         - Mixer for communications
%   signs_nrz           - Tool for creating NRZ waveforms
%   signs_psk           - m-ary PSK modulator
%   signs_read          - Reads data from a file
%   signs_ricepdf       - Returns the Rice distribution
%   signs_rms           - Tool for computing RMS values
%   signs_rndbikappa    - Returns random numbers from a bi-kappa distribution
%   signs_round         - Rounds numeric elements to M significant digits
%   signs_sgzoom        - "zoom in" tool for generic signals
%   signs_snr           - SNR estimator using Aisbett's hash function
%   signs_st4285        - Stanag 4285 modulator
%   signs_tsallispdf    - Returns the Tsallis distribution
%   signs_varham        - Tool for varying hamming distance of digital signals
%   signs_vsigt         - Tool for viewing Tx and Rx baseband signals in the time domain
%   signs_wigner        - Wigner distribution
%
% Terms of use.
%   SIGN(s) may be used by anyone without license.  You may copy, modify,
%   improve, or use it for non-commercial purposes provided that you acknowledge
%   the original author in all copies and modified versions, that you extend the same
%   rights of use to others who may find this program useful, and that you
%   inform such others that this program is freeware.
```

Appendix H

Ⓢigns Toolbox Code

PROGRAMMING code for the Ⓢ**igns** toolbox is listed in this appendix. The Ⓢ**igns** toolbox is implemented in the **MATLAB**® and C environments. The listings are complete for all Ⓢ**igns** related modules. These modules are used to achieve some of the results presented in this thesis. However, the author has also created many other **MATLAB**® scripts and processes, not listed here, that are used to acquire, analyze, and present other results in the thesis. As far as possible, and within the constraints of the research work schedule, the **MATLAB**® and C code is written with software engineering principles in mind and is commented so that function processes are self-evident. Interested readers may find occassions where these principles are not followed, and indeed, may find better ways of implementing aspects of the Ⓢ**igns** toolbox. These are all areas for futher work.

H.1 Ⓢigns Configuration Modules

Configuration modules are user customizable **MATLAB**® scripts that contain definitions of key analysis variables. A configuration module is executed before the analysis modules. The output of a configuration module is a **MATLAB**® .mat file that is the profile of the analysis. Some example scripts are contained in this section. The first script sets up an analysis of 2-FSK signals (see CFG2FSK), the second prepares Ⓢ**igns** for the analysis of 2-PSK signals (see CFG2PSK), and the last sets up parameters for Ⓢ**igns** to analyze Stanag 4285 signals (see CFGSTANAG).

CFG2FSK

```
%CLEAR WORKSPACE
%-------------------------------------------------------------------------
clear all
close all
clc

%GENERAL SIMULATION PARAMETERS
%-------------------------------------------------------------------------
CFG_T=10;                   %simulation time in seconds
CFG_Fs=32768;               %sampling rate in Hz
CFG_BYPASS=0;               %set to 1 to bypass upconverter, channel simulator, and
    downconverter
CFG_REALRX=0;                 %set to 1 to use real baseband data for the rx signal fed to
    extract
CFG_REALREF=0;              %set to 1 to use real baseband data as the reference signal for
    extract


%TX DATA SOURCE PARAMETERS
%-------------------------------------------------------------------------
CFG_Dpdf='uniform';         %pdf of data source (options: 'uniform','normal')
CFG_Dstd=1;                 %std. dev. of data source if CFG_Dpdf='normal'
CFG_Dmu=0;                  %mean of data source if CFG_Dpdf='normal'
CFG_HAM=[1];          %hamming distances (fraction of symbol vector length) between reference
    & Tx signals

%MODULATION PARAMETERS OF BASEBAND SIGNAL
%-------------------------------------------------------------------------
CFG_MODL='m-fsk';           %modulation type
CFG_R=150;                  %symbol rate in Hz
CFG_Fdev=85;                %peak frequency deviation for FSK
CFG_Fctr=2000;              %centre frequency of baseband signal
```

```
CFG_LVL=2;                  %number of FSK levels

%TRANSMITTER CHARACTERISTICS
%-------------------------------------------------------------------------
CFG_TxFc=0;                 %carrier frequency
CFG_TxBPFBW=800;            %bandwidth in Hz of transmitter BPF
CFG_TxBPFOFF=2000;          %centre of BPF offset (in Hz) from Fc
CFG_TxBPForder=512;         %number of taps for BPF
CFG_TxBPFwin='@hamming';    %window type (can be any matlab window type - see HELP WINDOW)
CFG_TxSB='usb';             %sideband to include in signal (dsb, usb, or lsb)
CFG_STYP='int8';            %equivalent quantization level in double precision

%CHANNEL CHARACTERISTICS
%-------------------------------------------------------------------------
CFG_CHANMOD='none';         %can be one of (awgn,watterson,v&h...
CFG_CCIR='good';            %can be one of (good,moderate,poor) according to CCIR fading
    channel parameters (default is good)

CFG_NBW=1000;               %noise bandwidth in Hz (used only if CFG_NOISE='AWGN')
CFG_SNR=[0];        %chosen SNRs in RF passband (in dB & valid only if CFG_NOISE='AWGN')

%RECEIVER CHARACTERISTICS
%-------------------------------------------------------------------------
CFG_RxBPFBW=800;            %bandwidth in Hz of receiver BPF
CFG_RxBPFOFF=2000;              %centre of BPF offset (in Hz) from Fc
CFG_RxLPFBW=3000;           %bandwidth in Hz of receiver LPF
CFG_RxBPForder=512;         %number of taps for BPF
CFG_RxBPFwin='@hamming';    %window type (can be any matlab window type - see HELP WINDOW)
CFG_RxLPForder=512;         %number of taps for LPF
CFG_RxLPFwin='@hamming';    %window type (can be any matlab window type - see HELP WINDOW)
CFG_RxEstFcON=0;            %set to 1 to enable carrier estimation algorithm for Rx
CFG_RxSB='usb';             %sideband to extract (dsb, usb, or lsb)
CFG_RxFc=0;                 %carrier for Rx if CFG_RxEstFcON=0


%EXTRACT CHARACTERISTICS
%-------------------------------------------------------------------------
%appropriate features:
%'coherence','cmd','b-entropy','psd'
CFG_PARAMETER={'b-entropy'}; %feature(s) to extract (must be a cell array of strings)

%additional parameters for 'coherence','psd', and the psd computed by the reporting tool
CFG_NFFT=1024; %FFT size for psd parameters
CFG_SEG=128; %number of segments for spectral estimation
CFG_OVR=50; %percentage overlap of segments for spectral estimation using Welch's method

%addtional parameters for 'b-entropy'
CFG_bentAP=[100:25:1000]; %length of appending sequence (in percent of length of signal vector
    )
```

```
CFG_bentN=10.^[-1.8:.05:0]; %number of samples used in entropy calculations (in percent of
    length of signal vector)
CFG_bentNT=3; %number of trials per entropy calculation
CFG_bentCMP='lzw'; %can be 'zip23' or 'lzw' compression

%EXPRESS CHARACTERISTICS
%-------------------------------------------------------------------------------


%REPORT PARAMETERS
%-------------------------------------------------------------------------------
CFG_WINDOW='@rectwin,5'; %averaging window type (can be any matlab window type - see HELP
    WINDOW)
CFG_RPTTYP='scatter-solid';      %can be 'scatter','scatter-X', or 'X' where X is one of (
    solid, dotted, dashdot, dashed)
CFG_AZ=45;                  %azimuth for viewing angle of 3D plots
CFG_EL=30;                  %elevation for viewing angle of 3D plots

%Time Series: parameters for plotting
CFG_TMSPLOT=1; %set to one to plot time series waveforms
CFG_TMSRAT=.65; %aspect ratio (width to height ratio) for each time series plot (width is base
     measurement)
CFG_TMSSIG={'SIG_REF','SIG_TxRF','SIG_RxBS'}; %signals for time domain plots
%indices of plots (must be a cell array); the 1st element in row is the PSD for SIG_REF
%the remaining elements in each row are the PSDs for SIG_RxBS.  If CFG_PSDSEL='all' then all
%PSDs are plotted.
CFG_TMSSEL={[1,11];[2,11];[3,11]};

CFG_PLOTRAW=0; %set to one to plot all individual coherenceVSfrequency plots
CFG_SAVERAW=0; %set to one to save all individual coherenceVSfrequency plots
CFG_PLOTWIN=0; %set to one to plot all final windowed result plots
CFG_SAVEWIN=0; %set to one to save final windowed result plots
CFG_CLSPLTS=0; %set to one to close figures after simulation completes

%PATHS
%-------------------------------------------------------------------------------
wdir=cd; %get working directory
CFG_resultspath=[wdir,'\Results\'];
CFG_profilespath=[wdir,'\Profiles\'];
CFG_logpath=[wdir,'\Logs\'];
CFG_temppath=[wdir,'\Temp\'];
%addpath(CFG_resultspath,CFG_profilespath,CFG_realinputpath,CFG_logpath);

%FILE NAMES
%-------------------------------------------------------------------------------
CFG_realFileRX='d:\capture.4.0.bin'; %filename of real Rx data for input to extract (used if
    CFG_REALRX=1)
CFG_realFileREF='d:\capture.4.0.bin'; %filename of real reference data for input to extract (
    used if CFG_REALREF=1);
CFG_realPrecision='int32'; %precision for reading the real data
```

```
CFG_realMask='hFFFFF000'; %mask for the real data (see SIGNS_READ)
CFG_realComponent='i'; %get in-phase data (see SIGNS_READ)
CFG_datagen_outFile=[CFG_resultspath,'datagen.mat']; %filename for output from datagen for
    creation of new messages
CFG_txsim_outFile=[CFG_resultspath,'txsim.mat']; %filename for output from txsim
CFG_rfsim_outFile=[CFG_resultspath,'rfsim.mat']; %filename for output from rfsim
CFG_rxsim_outFile=[CFG_resultspath,'rxsim.mat']; %filename for output from rxsim
CFG_xswitch_outFile=[CFG_resultspath,'xswitch.mat']; %filename for output from xswitch
CFG_extract_outFile=[CFG_resultspath,'extract.mat']; %filename for output from extract
CFG_logfile=[CFG_logpath,'logfile.txt']; %filename for logfile output

%SAVE VARIABLES
%-------------------------------------------------------------------------
cd([CFG_profilespath]);
[filename,path2file]=uiputfile('profile.mat','SIGN(s) : Save Profile As');
if ~isequal(filename,0)&~isequal(path2file,0) %check for errors
    filename=strjust(filename,'left'); %remove possible white space
    idx=findstr(filename,'.');
    if idx
        save([path2file,filename(1:idx-1),'.mat'],'CFG_*');
    else
        save([path2file,filename,'.mat'],'CFG_*');
    end
else
    disp('SIGN(s) : File Save Error or Cancel Selected');
end
cd(wdir);
clear all
close all
```

CFG2PSK

```
%CLEAR WORKSPACE
%-------------------------------------------------------------------------
clear all
close all
clc

%GENERAL SIMULATION PARAMETERS
%-------------------------------------------------------------------------
CFG_T=10;                   %simulation time in seconds
CFG_Fs=32768;               %sampling rate in Hz
CFG_BYPASS=0;               %set to 1 to bypass upconverter, channel simulator, and
    downconverter
CFG_REALRX=0;                 %set to 1 to use real baseband data for the rx signal fed to
    extract
CFG_REALREF=0;              %set to 1 to use real baseband data as the reference signal for
    extract


%TX DATA SOURCE PARAMETERS
%-------------------------------------------------------------------------
CFG_Dpdf='uniform';         %pdf of data source (options: 'uniform','normal')
CFG_Dstd=1;                 %std. dev. of data source if CFG_Dpdf='normal'
CFG_Dmu=0;                  %mean of data source if CFG_Dpdf='normal'
CFG_HAM=[1];          %hamming distances (fraction of symbol vector length) between reference
    & Tx signals

%MODULATION PARAMETERS OF BASEBAND SIGNAL
%-------------------------------------------------------------------------
CFG_MODL='m-psk';           %modulation type
CFG_R=150;                  %symbol rate in Hz
CFG_Po=0;                   %phase offset from the abcissa for the signal constellation (
    radians)
CFG_Fctr=3000;              %centre frequency of baseband signal
CFG_LVL=2;                  %number of PSK levels

%TRANSMITTER CHARACTERISTICS
%-------------------------------------------------------------------------
CFG_TxFc=0;               %carrier frequency
CFG_TxBPFBW=800;            %bandwidth in Hz of transmitter BPF
CFG_TxBPFOFF=2000;          %centre of BPF offset (in Hz) from Fc
CFG_TxBPForder=512;         %number of taps for BPF
CFG_TxBPFwin='@hamming';    %window type (can be any matlab window type - see HELP WINDOW)
CFG_TxSB='usb';             %sideband to include in signal (dsb, usb, or lsb)
CFG_STYP='int8';            %equivalent quantization level in double precision

%CHANNEL CHARACTERISTICS
%-------------------------------------------------------------------------
CFG_CHANMOD='none';         %can be one of {awgn,watterson,v&h...
```

```
CFG_CCIR='good';              %can be one of {good,moderate,poor} according to CCIR fading
    channel parameters (default is good)

CFG_NBW=1000;                 %noise bandwidth in Hz (used only if CFG_NOISE='AWGN')
CFG_SNR=[0];          %chosen SNRs in RF passband (in dB & valid only if CFG_NOISE='AWGN')

%RECEIVER CHARACTERISTICS
%-------------------------------------------------------------------------
CFG_RxBPFBW=800;              %bandwidth in Hz of receiver BPF
CFG_RxBPFOFF=2000;                %centre of BPF offset (in Hz) from Fc
CFG_RxLPFBW=3000;             %bandwidth in Hz of receiver LPF
CFG_RxBPForder=512;           %number of taps for BPF
CFG_RxBPFwin='@hamming';      %window type (can be any matlab window type - see HELP WINDOW)
CFG_RxLPForder=512;           %number of taps for LPF
CFG_RxLPFwin='@hamming';      %window type (can be any matlab window type - see HELP WINDOW)
CFG_RxEstFcON=0;              %set to 1 to enable carrier estimation algorithm for Rx
CFG_RxSB='usb';               %sideband to extract (dsb, usb, or lsb)
CFG_RxFc=0;                 %carrier for Rx if CFG_RxEstFcON=0


%EXTRACT CHARACTERISTICS
%-------------------------------------------------------------------------
%appropriate features:
%'coherence','cmd','b-entropy','psd'
CFG_PARAMETER={'b-entropy'}; %feature(s) to extract (must be a cell array of strings)

%additional parameters for 'coherence','psd', and the psd computed by the reporting tool
CFG_NFFT=1024; %FFT size for psd parameters
CFG_SEG=128; %number of segments for spectral estimation
CFG_OVR=50; %percentage overlap of segments for spectral estimation using Welch's method

%addtional parameters for 'b-entropy'
CFG_bentAP=[100:25:1000]; %length of appending sequence (in percent of length of signal vector
    )
CFG_bentN=10.^[-1.8:.05:0]; %number of samples used in entropy calculations (in percent of
    length of signal vector)
CFG_bentNT=3; %number of trials per entropy calculation
CFG_bentCMP='lzw'; %can be 'zip23' or 'lzw' compression

%EXPRESS CHARACTERISTICS
%-------------------------------------------------------------------------


%REPORT PARAMETERS
%-------------------------------------------------------------------------
CFG_WINDOW='@rectwin,5'; %averaging window type (can be any matlab window type - see HELP
    WINDOW)
CFG_RPTTYP='scatter-solid';       %can be 'scatter','scatter-X', or 'X' where X is one of {
    solid, dotted, dashdot, dashed}
CFG_AZ=45;                  %azimuth for viewing angle of 3D plots
```

```
CFG_EL=30;                  %elevation for viewing angle of 3D plots

%Time Series: parameters for plotting
CFG_TMSPLOT=1; %set to one to plot time series waveforms
CFG_TMSRAT=.65; %aspect ratio (width to height ratio) for each time series plot (width is base
    measurement)
CFG_TMSSIG={'SIG_REF','SIG_TxRF','SIG_RxBS'}; %signals for time domain plots
%indices of plots (must be a cell array); the 1st element in row is the PSD for SIG_REF
%the remaining elements in each row are the PSDs for SIG_RxBS.  If CFG_PSDSEL='all' then all
%PSDs are plotted.
CFG_TMSSEL={[1,11];[2,11];[3,11]};

CFG_PLOTRAW=0; %set to one to plot all individual coherenceVSfrequency plots
CFG_SAVERAW=0; %set to one to save all individual coherenceVSfrequency plots
CFG_PLOTWIN=0; %set to one to plot all final windowed result plots
CFG_SAVEWIN=0; %set to one to save final windowed result plots
CFG_CLSPLTS=0; %set to one to close figures after simulation completes

%PATHS
%-------------------------------------------------------------------------------
wdir=cd; %get working directory
CFG_resultspath=[wdir,'\Results\'];
CFG_profilespath=[wdir,'\Profiles\'];
CFG_logpath=[wdir,'\Logs\'];
CFG_temppath=[wdir,'\Temp\'];
%addpath(CFG_resultspath,CFG_profilespath,CFG_realinputpath,CFG_logpath);

%FILE NAMES
%-------------------------------------------------------------------------------
CFG_realFileRX='d:\capture.4.0.bin'; %filename of real Rx data for input to extract (used if
    CFG_REALRX=1)
CFG_realFileREF='d:\capture.4.0.bin'; %filename of real reference data for input to extract (
    used if CFG_REALREF=1);
CFG_realPrecision='int32'; %precision for reading the real data
CFG_realMask='hFFFFF000'; %mask for the real data (see SIGNS_READ)
CFG_realComponent='i'; %get in-phase data (see SIGNS_READ)
CFG_datagen_outFile=[CFG_resultspath,'datagen.mat']; %filename for output from datagen for
    creation of new messages
CFG_txsim_outFile=[CFG_resultspath,'txsim.mat']; %filename for output from txsim
CFG_rfsim_outFile=[CFG_resultspath,'rfsim.mat']; %filename for output from rfsim
CFG_rxsim_outFile=[CFG_resultspath,'rxsim.mat']; %filename for output from rxsim
CFG_xswitch_outFile=[CFG_resultspath,'xswitch.mat']; %filename for output from xswitch
CFG_extract_outFile=[CFG_resultspath,'extract.mat']; %filename for output from extract
CFG_logfile=[CFG_logpath,'logfile.txt']; %filename for logfile output

%SAVE VARIABLES
%-------------------------------------------------------------------------------
cd([CFG_profilespath]);
[filename,path2file]=uiputfile('profile.mat','SIGN(s) : Save Profile As');
if ~isequal(filename,0)&~isequal(path2file,0) %check for errors
```

```
    filename=strjust(filename,'left'); %remove possible white space
    idx=findstr(filename,'.');
    if idx
        save([path2file,filename(1:idx-1),'.mat'],'CFG_*');
    else
        save([path2file,filename,'.mat'],'CFG_*');
    end
else
    disp('SIGN(s) : File Save Error or Cancel Selected');
end
cd(wdir);
clear all
close all
```

CFGSTANAG

```
%CLEAR WORKSPACE
%-------------------------------------------------------------------------
clear all
close all
clc

%SIMULATION PARAMETERS
%-------------------------------------------------------------------------
CFG_T=.5;                   %simulation time in seconds
CFG_Fs=50e3;                %sampling rate in Hz

%TX DATA SOURCE PARAMETERS
%-------------------------------------------------------------------------
CFG_Dpdf='uniform';         %pdf of data source (options: 'uniform','normal')
CFG_Dstd=1;                 %std. dev. of data source if CFG_Dpdf='normal'
CFG_Dmu=0;                  %mean of data source if CFG_Dpdf='normal'
CFG_HAM=[0,.5,1];           %hamming distances (% of symbol vector length) between reference &
    Tx signals

%MODULATION PARAMETERS OF BASEBAND SIGNAL
%-------------------------------------------------------------------------
CFG_MODL='stanag4285';      %modulation type
CFG_R=1200;                 %symbol rate in Hz



%TRANSMITTER CHARACTERISTICS
%-------------------------------------------------------------------------
CFG_TxFc=10e3;              %carrier frequency
CFG_TxBPFBW=3000;           %bandwidth in Hz of transmitter BPF
CFG_TxBPFOFF=1800;          %centre of BPF offset (in Hz) from Fc
CFG_TxBPForder=512;         %number of taps for BPF
CFG_TxBPFwin='@hamming';    %window type (can be any matlab window type - see HELP WINDOW)
CFG_TxSB='dsb';             %sideband to include in signal (dsb, usb, or lsb)

%CHANNEL CHARACTERISTICS
%-------------------------------------------------------------------------
CFG_CHANMOD='awgn';         %can be one of {awgn,watterson,v&h
CFG_CCIR='good';            %can be one of {good,moderate,poor} according to CCIR fading
    channel parameters (default is good)

CFG_NBW=7000;               %noise bandwidth in Hz (used only if CFG_NOISE='AWGN')
CFG_SNR=[-10:2:10];         %chosen SNRs in RF passband (in dB & valid only if CFG_NOISE='AWGN
    ')

%RECEIVER CHARACTERISTICS
%-------------------------------------------------------------------------
```

```
CFG_RxBPFBW=3000;           %bandwidth in Hz of receiver BPF
CFG_RxBPFOFF=1800;          %centre of BPF offset (in Hz) from Fc
CFG_RxLPFBW=3300;           %bandwidth in Hz of receiver LPF
CFG_RxBPForder=512;         %number of taps for BPF
CFG_RxBPFwin='@hamming';    %window type (can be any matlab window type - see HELP WINDOW)
CFG_RxLPForder=512;         %number of taps for LPF
CFG_RxLPFwin='@hamming';    %window type (can be any matlab window type - see HELP WINDOW)
CFG_RxEstFcON=0;            %set to 1 to enable carrier estimation algorithm for Rx
CFG_RxSB='usb';             %sideband to demodulate (dsb, usb, or lsb)
CFG_RxFc=10e3;              %carrier for Rx if CFG_RxEstFcON=0


%EXTRACT CHARACTERISTICS
%-------------------------------------------------------------------------
CFG_PARAMETER={'coherence',...
        'cmd'};             %feature(s) to extract (must be a cell array of strings)
CFG_REAL=0;                 %set to 1 if extract is to use real baseband data

%EXPRESS CHARACTERISTICS
%-------------------------------------------------------------------------


%REPORT PARAMETERS
%-------------------------------------------------------------------------
CFG_WINDOW='@rectwin,7'; %averaging window type (can be any matlab window type - see HELP
    WINDOW)
CFG_RPTTYP='line';      %can be 'scatter','scatter-line','line' (line is windowed by
    CFG_WINDOW)
CFG_REPORT={'cohsnr',... %coherence versus SNR
        'cmdsnr',...    %CMD versus SNR
        'chsnrf',...    %coherence versus SNR and frequency
        'cohham',...    %coherence versus Hamming distance
        'cmdham',...    %CMD versus Hamming distance
        'cohfrq',...    %coherence versus frequency
        'enzprb',...    %entropy versus probability of zero bit
        'enmsgb',...    %entropy versus length of message and length of appending sequence
        'clzprb',...    %compressed length of messeage versus actual message length and
            probability of zero bit
        'entmsg',...    %entropy versus length of message
        'entsam',...    %entropy versus number of samples
        'entsnr',...    %entropy versus SNR
        'psd'};         %psd of trials
CFG_AZ=45;              %azimuth for viewing angle of 3D plots
CFG_EL=45;              %elevation for viewing angle of 3D plots

%Time Series: parameters for plotting
CFG_TMSPLOT=0; %set to one to plot time series waveforms
CFG_TMSRAT=1; %aspect ratio (width to height ratio) for each psd plot (width is base
    measurement)
CFG_TMSSIG={'SIG_REF','SIG_RxBS'}; %signals for time domain plots
```

```
%indices of plots (must be a cell array); the 1st element in row is the PSD for SIG_REF
%the remaining elements in each row are the PSDs for SIG_RxBS.  If CFG_PSDSEL='all' then all
%PSDs are plotted.
CFG_TMSSEL={[1,11];[2,11];[3,11]};

%Power Spectral Density: parameters for plotting
%psd parameters used if CFG_PLOTPSD=1 (psd plots depend on
CFG_PSDPLOT=1; %set to one to plot psd of time series waveforms of SIG_REF and SIG_RxBS
CFG_PSDRAT=.6; %aspect ratio (width to height ratio) for each psd plot (width is base
    measurement)
CFG_PSDSIG={'SIG_REF','SIG_TxRF','SIG_RxBS'}; %signals for PSDs
%indices of PSDs to plot (must be a cell array); the 1st element in row is the PSD for SIG_REF
%the remaining elements in each row are the PSDs for SIG_RxBS.  If CFG_PSDSEL='all' then all
%PSDs are plotted.
CFG_PSDSEL={[1,11];[2,11];[3,11]};

CFG_PLOTRAW=0; %set to one to plot all individual coherenceVSfrequency plots
CFG_SAVERAW=0; %set to one to save all individual coherenceVSfrequency plots
CFG_PLOTWIN=0; %set to one to plot all final windowed result plots
CFG_SAVEWIN=0; %set to one to save final windowed result plots
CFG_CLSPLTS=0; %set to one to close figures after simulation completes

%PATHS
%-------------------------------------------------------------------------
wdir=cd; %get working directory
CFG_resultspath=[wdir,'\Results\'];
CFG_profilespath=[wdir,'\Profiles\'];
CFG_realinputpath=[wdir,'\RealData\'];
CFG_logpath=[wdir,'\Logs\'];
CFG_temppath=[wdir,'\Temp\'];
%addpath(CFG_resultspath,CFG_profilespath,CFG_realinputpath,CFG_logpath);

%FILE NAMES
%-------------------------------------------------------------------------
CFG_realFile=[CFG_realinputpath,'real.mat']; %filename of real data for input to extract (used
    if CFG_REAL=1)
CFG_datagen_outFile=[CFG_resultspath,'datagen.mat']; %filename for output from datagen for
    creation of new messages
CFG_txsim_outFile=[CFG_resultspath,'txsim.mat']; %filename for output from txsim
CFG_rfsim_outFile=[CFG_resultspath,'rfsim.mat']; %filename for output from rfsim
CFG_rxsim_outFile=[CFG_resultspath,'rxsim.mat']; %filename for output from rxsim
CFG_extract_outFile=[CFG_resultspath,'extract.mat']; %filename for output from extract
CFG_logfile=[CFG_logpath,'logfile.txt']; %filename for logfile output

%SAVE VARIABLES
%-------------------------------------------------------------------------
cd([CFG_profilespath]);
[filename,path2file]=uiputfile('profile.mat','SIGN(s) : Save Profile As');
if ~isequal(filename,0)&~isequal(path2file,0) %check for errors
    filename=strjust(filename,'left'); %remove possible white space
```

```
    idx=findstr(filename,'.');
    if idx
        save([path2file,filename(1:idx-1),'.mat'],'CFG_*');
    else
        save([path2file,filename,'.mat'],'CFG_*');
    end
else
    disp('SIGN(s) : File Save Error or Cancel Selected');
end
cd(wdir);
```

H.2 Ⓢigns Initiate Script

The Ⓢ**igns** initiation script is executed after at least one configuration script is executed. The output of the configuration script is a .mat file, which defines an analysis profile. The initiation script, **sign_s.m**, prompts the Ⓢ**igns** user to choose an analysis profile. The script then controls the entire analysis based on parameter values defined in the profile.

SIGN_S

```
%SIGN_S
%
%The purpose of this script is to initiate the analysis.  The sign_s
%module is to be executed after the user executes a configuration
%module.  The sign_s script will ask the user to choose an analysis
%profile (ie the output of a configuration script) prior to loading
%the profile and executing the analysis based on the key parameter
%values in the profile.
%
%Mandatory Input Parameters:
%    None
%
%Optional Input Parameters:
%    None
%
%Mandatory Output Parameters:
%    None
%
%Optional Output Parameters:
%    None
%
%Special Notes:
%
%
%Copyright (c) 2003 James Giesbrecht

%Company:
%
%Revision History:
%Date         Version   Editor             Changes Made
%-----------  --------  ---------------    ---------------------
%

%clean up workspace
clear all
```

```
close all
%get profile
[filename,path2file]=uigetfile('profile.mat','SIGN(s) : Open Profile');
if isequal(filename,0) | isequal(path2file,0) %check for errors
    disp('SIGN(s) : File Open Error or Cancel Selected');
else
    %prepare profile string
    filename=strjust(filename,'left'); %remove possible white space
    path2file=strjust(path2file,'left'); %remove possible white space
    profilename=[path2file,filename];
    %get key variables
    load(profilename,'CFG_logfile');
    %start processing
    if ~datagen(profilename)
        if ~txsim(profilename)
            if ~rfsim(profilename)
                if ~rxsim(profilename)
                    if ~xswitch(profilename)
                        if ~extract(profilename)
                            if ~report(profilename)
                                signs_log(CFG_logfile,'***SIGN(s): Successful completion of
                                    analysis');
                            else
                                signs_log(CFG_logfile,'***SIGN(s): Warning -- Unsuccesful
                                    completion of REPORT');
                            end
                        else
                            signs_log(CFG_logfile,'***SIGN(s): Warning -- Unsuccesful
                                completion of EXTRACT');
                        end
                    else
                        signs_log(CFG_logfile,'***SIGN(s): Warning -- Unsuccesful completion
                            of XSWITCH');
                    end
                else
                    signs_log(CFG_logfile,'***SIGN(s): Warning -- Unsuccesful completion of
                        RXSIM');
                end
            else
                signs_log(CFG_logfile,'***SIGN(s): Warning -- Unsuccesful completion of RFSIM'
                    );
            end
        else
            signs_log(CFG_logfile,'***SIGN(s): Warning -- Unsuccesful completion of TXSIM');
        end
    else
        signs_log(CFG_logfile,'***SIGN(s): Warning -- Unsuccesful completion of DATAGEN');
    end
end
```

H.3 Ⓢigns Analysis Modules

Analysis modules are part of the staged approach to the Ⓢ**igns** analysis package. Each module completes a specific set of related tasks of analysis, the output of which is critical input for the next stage.

EXTRACT

```
%Sign(s) Module: EXTRACT
%
%SUCC=EXTRACT(inputProfile)
%
%The purpose of this function is to extract features from two baseband signals (one which
%is affected by the HF channel). The parameter extraction depends on configuration
    information
%stored in the inputfile. So, the algorithm varies with this configuration information.
%
% Input Parameters:
%   inputProfile - string variable containing the name of the .mat file that contains
%                  the simulation profile. The input profile is a .mat file that contains
%                  details of the simulation. This file can be created with the profile tool
    .
%                  If inputProfile is empty, a profile selection window will appear to allow
%                  the user to select a profile.
% Output Parameters:
%   SUCC - 0 if the function completed successfully, otherwise 1.
% Input Files:
%   As specified by inputFile
% Output Files:
%   As specified by outputFile
% Special Notes:
%   See comments in code.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%-----------  --------  ----------------  --------------------

function SUCC=extract(inputProfile)
try
    %get profile
    if exist('inputProfile')
        load(inputProfile);
    else
        %if the profile wasn't passed as an argument, the use the gui
        [filename,path2file]=uigetfile('profile.mat','SIGN(s) : Open Profile');
```

```
    if isequal(filename,0) | isequal(path2file,0) %check for errors
        disp('SIGN(s) : File Open Error or Cancel Selected');
        SUCC=1;
        return
    else
        %prepare profile string
        filename=strjust(strjust(filename,'left'),'right'); %remove possible white space
        path2file=strjust(strjust(path2file,'left'),'right'); %remove possible white space
        inputProfile=[path2file,filename];

        %get key variables
        load(inputProfile);
    end
end

%assume successful operation
SUCC=0;

%get other inputfiles
load(CFG_xswitch_outFile);

%extract parameters of interest
sz=size(SIG_RxBS);
rows=sz(1);
cols=sz(2);
for i=1:length(CFG_PARAMETER)
    switch CFG_PARAMETER{i}

    %coherence
    case 'coherence',

        %create waitbar
        H=waitbar(0,'Extracting Coherence.  Please wait...');
        set(H,'Name','Sign(s):  EXTRACT');

        for k=1:rows
            for m=1:cols

                %extract parameter
                if ~isempty(CFG_WINDOW)
                    [EXT_COH(k,m),ERR]=ext_cohr(SIG_RxBS{k,m},SIG_REF{k},CFG_Fs,CFG_WINDOW
                        ,[CFG_SEG,CFG_OVR]);
                else
                    [EXT_COH(k,m),ERR]=ext_cohr(SIG_RxBS{k,m},SIG_REF{k},CFG_Fs,[CFG_SEG,
                        CFG_OVR]);
                end

                %check for errors
                if ~isempty(ERR)
                    signs_log(CFG_logfile,ERR{1});
```

```
                ERR=[];
                SUCC=1;
                break
            end

            %update waitbar
            waitbar(((k-1)*cols+m)/rows/cols,H)
        end
    end

    %close waitbar
    close(H);

%coherence-median difference
case 'cmd', %coherence-median difference
    if exist('EXT_COH')==1

        %create waitbar
        H=waitbar(0,'Extracting CMD.  Please wait...');
        set(H,'Name','Sign(s):  EXTRACT');

        for k=1:rows
            for m=1:cols

                %extract parameter
                switch CFG_MODL
                case 'm-fsk',
                    symbol_spacing=2*CFG_Fdev/(CFG_LVL-1);
                    Fsym=[CFG_Fctr-CFG_Fdev:symbol_spacing:CFG_Fctr+CFG_Fdev];
                    if ~isempty(CFG_WINDOW)
                        [EXT_CMDf(k,m),ERR]=ext_cmd(EXT_COH(k,m).Cxy,EXT_COH(k,m).Freq
                            ,Fsym,CFG_WINDOW); %determine the CMD
                    else
                        [EXT_CMDf(k,m),ERR]=ext_cmd(EXT_COH(k,m).Cxy,EXT_COH(k,m).Freq
                            ,Fsym); %determine the CMD
                    end
                otherwise
                    if ~isempty(CFG_WINDOW)
                        [EXT_CMDf(k,m),ERR]=ext_cmd(EXT_COH(k,m).Cxy,EXT_COH(k,m).Freq
                            ,CFG_WINDOW); %determine generalized CMD
                    else
                        [EXT_CMDf(k,m),ERR]=ext_cmd(EXT_COH(k,m).Cxy,EXT_COH(k,m).Freq
                            ); %determine generalized CMD
                    end
                end

                %check for errors
                if ~isempty(ERR)
                    signs_log(CFG_logfile,ERR{1});
                    ERR=[];
```

```
                    SUCC=1;
                    break
                end

                %update waitbar
                waitbar(((k-1)*cols+m)/rows/cols,H)
            end
        end

        %close waitbar
        close(H);
    else
        signs_log(CFG_logfile,'***SIGN(s): Warning -- Extraction of CMD ignored -->
            Coherence not in feature list');
    end
case 'instfreq',    %instantaneous frequency
case 'instphase',   %instantaneous phase
case 'instamp',     %instantaneous amplitude
case 'b-entropy',   %benedetto entropy

    %create waitbar
    H=waitbar(0,'Extracting Benedetto Entropy.  Please wait...');
    set(H,'Name','Sign(s):  EXTRACT');

    for k=1:rows
        for m=1:cols

            %extract parameter
            [EXT_ENT(k,m),ERR]=ext_entropy('benedetto',SIG_RxBS{k,m},SIG_REF{k},
                CFG_bentAP,...
                CFG_bentN,CFG_bentCMP,CFG_bentNT,CFG_WINDOW,CFG_temppath);

            %check for errors
            if ~isempty(ERR)
                signs_log(CFG_logfile,ERR{1});
                ERR=[];
                SUCC=1;
                break
            end

            %update waitbar
            waitbar(((k-1)*cols+m)/rows/cols,H)
        end
    end

    %close waitbar
    close(H);
case 'sincoeff',    %sinusoid coefficient (measure of how close a signal is to a
    sinusoid)
case 'carrier',     %estimate of carrier frequency
```

```
        %estimate psd of signal using Welch's method
        case 'psd',

            %create waitbar
            H=waitbar(0,'Extracting␣PSDs.␣␣Please␣wait...');
            set(H,'Name','Sign(s):␣␣EXTRACT');

            for k=1:rows
                for m=1:cols
                    %extract parameter
                    [EXT_PSD(k,m),ERR]=ext_psd(SIG_RxBS{k,m},CFG_NFFT,CFG_Fs,CFG_OVR);

                    %check for errors
                    if ~isempty(ERR)
                        signs_log(CFG_logfile,ERR{1});
                        ERR=[];
                        SUCC=1;
                        break
                    end

                    %update waitbar
                    waitbar(((k-1)*cols+m)/rows/cols,H)
                end
            end

            %close waitbar
            close(H);
        end
    end

    %save data
    save(CFG_extract_outFile,'EXT_*');
catch
    %log error condition
    signs_log(CFG_logfile,dbstack);
    SUCC=1;
    close all;
end
return
```

RFSIM

```
%Sign(s) Module: RFSIM
%
%SUCC=RFSIM(inputProfile)
%
%The purpose of this function is to apply a transmission medium model to a
%passband signal created with TXSIM.
%
%This routine expects a .mat configuration file as an input and the output is
%a .mat file containing the RF signal modified by the transmission medium.
%
%The routine is completely controlled by the contents of the input configuration file.
%See the example configuration files for more information.  The variables defined in the
%input configuration file depend on the type of modulation for the hf signal, the
%simulation run parameters, and output control parameters.
%
%   Permitted Channel Models:
%     AWGN          - gaussian noise only added to RF signal
%     NBHF          - narrowband HF channel model by Watterson/CCIR
%     vhWBHFd       - deterministic wideband HF channel model by Vogler and Hoffmeyer
%     vhWBHFs       - stochastic wideband HF channel model by Vogler and Hoffmeyer
%     vhWBHFt       - wideband transfer function of HF channel by Vogler and Hoffmeyer
%     lbWBHFf       - wideband HF channel model with first-order stats. from Lemmon and Behm
%     lbWBHFh       - wideband HF channel model with higher-order stats. from Lemmon and Behm
%
%  Input Parameters:
%    inputProfile - string variable containing the name of the .mat file that contains
%                   the simulation profile.  The input profile is a .mat file that contains
%                   details of the simulation.  This file can be created with the profile tool
    .
%                   If inputProfile is empty, a profile selection window will appear to allow
%                   the user to select a profile.
%  Output Parameters:
%    SUCC - 0 if the function completed successfully, otherwise 1.
%  Input Files:
%    As specified by inputProfile
%  Output Files:
%    As specified by inputProfile
%  Special Notes:
%    The output file contains one variable representing the RF signal:
%       SIG_RF      -   RF signal modified by the transmission medium
%    See additional comments in code.
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%-----------  --------  ----------------  --------------------
```

```
function SUCC=rfsim(inputProfile)
try
    %get profile
    if exist('inputProfile')
        load(inputProfile);
    else
        %if the profile wasn't passed as an argument, the use the gui
        [filename,path2file]=uigetfile('profile.mat','SIGN(s) : Open Profile');
        if isequal(filename,0) | isequal(path2file,0) %check for errors
            disp('SIGN(s) : File Open Error or Cancel Selected');
            SUCC=1;
            return
        else
            %prepare profile string
            filename=strjust(strjust(filename,'left'),'right'); %remove possible white space
            path2file=strjust(strjust(path2file,'left'),'right'); %remove possible white space
            inputProfile=[path2file,filename];

            %get key variables
            load(inputProfile);
        end
    end

    %assume successful operation
    SUCC=0;

    %get other inputfiles
    load(CFG_txsim_outFile,'SIG_TxRF');

    %create waitbar
    H=waitbar(0,'Working.  Please wait...');
    set(H,'Name','Sign(s):  RFSIM');

    %loop for each transmitted signal
    Ks=length(SIG_TxRF);
    for k=1:Ks

        %check bypass variable
        if ~CFG_BYPASS
            %apply HF channel
            switch lower(CFG_CHANMOD)
                %     AWGN          - gaussian noise only added to RF signal
            case 'awgn',
                %given
                N=length(SIG_TxRF{k});
                Psignal=20*log10(signs_rms(SIG_TxRF{k}));      %get signal power (in dB)
                %loop for each SNR
                %
                %                                              Ps
                %note that SNR in passband = 10*log10(----------------)
```

```
        %                                       Pn*Nbw/Fs*2
        %where Ps is the signal power in the passband, Pn is the noise power across
            the Nyquist band,
        %and Nbw is the noise bandwidth required for the passband signal.
        %
        for m=1:length(CFG_SNR)
            %SNR(dB) = 10log10(Psignal)-10log10(Pnoise in passband)
            Pnoise=Psignal-CFG_SNR(m); %get noise power of awgn in signal passband
            %Pnoise = (BW of signal)x(Noise Density)
            %Noise Density = 10log10[(var of AWGN)] / (Fs/2)
            %therefore (var of AWGN)=10^((Pnoise/(BW of signal)/2*Fs)/10)
            Nstd=10^((Pnoise/CFG_NBW*(CFG_Fs/2))/20); %get std. of awgn
            %Ndensity=20*log10(Nstd)*2/CFG_Fs; %get the noise density (dB/Hz)
            Nsignal=randn(1,N)*Nstd;            %adjust sigma of noise pdf
            SIG_RF{k,m}=SIG_TxRF{k}+Nsignal;    %add in the zero-mean gaussian noise
        end
        %     NBHF          - narrowband HF channel model by Watterson/CCIR
    case 'watterson',
        %read in CCIR fading channel parameters
        %row 1 is delay spread (sec); row2 is doppler spread (Hz)
        %col 1 is a "good" channel; col 2 is a "moderate" channel; col 3 is a "poor"
            channel
        P=dlmread('signs_ccir.dat',',',1,1);

        %get parameter index
        switch lower(CFG_CCIR)
        case 'good',
            col=1;
        case 'moderate',
            col=2;
        case 'poor',
            col=3;
        otherwise
            col=1; %defaul
        end

        %get delay spread and doppler spread
        DLYSPD=P(col,1);
        DOPSPD=P(col,2);

        %     vhWBHFd       - deterministic wideband HF channel model by Vogler and
            Hoffmeyer
    case 'vhWBHFd',
        %     vhWBHFs       - stochastic wideband HF channel model by Vogler and
            Hoffmeyer
    case 'vhWBHFs',
        %     vhWBHFt       - wideband transfer function of HF channel by Vogler and
            Hoffmeyer
    case 'vhWBHFt',
```

```
                %       lbWBHFf         - wideband HF channel model with first-order stats. from
                    Lemmon and Behm
            case 'lbWBHFf',
                %       lbWBHFh         - wideband HF channel model with higher-order stats. from
                    Lemmon and Behm
            case 'lbWBHFh',
            otherwise,
                %do nothing to the signal
                for m=1:length(CFG_SNR)
                    SIG_RF{k,m}=SIG_TxRF{k};
                end
            end
        else
            %do nothing to the signal
            for m=1:length(CFG_SNR)
                SIG_RF{k,m}=[];
            end
        end

        %update waitbar
        waitbar(k/Ks,H);
    end

    %save signals
    save(CFG_rfsim_outFile,'SIG_RF');

    %close waitbar
    close(H);
catch
    %log error condition
    signs_log(CFG_logfile,dbstack);
    SUCC=1;
    close all;
end
return
```

RXSIM

```
%Sign(s) Module: RXSIM
%
%SUCC=RXSIM(inputProfile)
%
%The purpose of this function is to generate two baseband signals.  The model
%implemented by this function is described on page 72 of Book I.  One signal
%is unaffected by the transmission channel, the other is affected by the
%transmission channel.  This function stops short of parameter extraction.
%Rather, it creates an ideal baseband signal and a life-like received
%baseband signal for the parameter extraction routine to determine
%modulation recognition parameters.
%
%This routine expects a .m configuration file as an input and the output is
%a .mat file containing variables of configuration information and baseband
%signals.
%
%The routine is completely controlled by the contents of the input configuration file.
%See the example configuration files for more information.  The variables defined in the
%input configuration file depend on the type of modulation for the hf signal, the
%simulation run parameters, and output control parameters.
%
%Permitted Modulation Types:
%   m-fsk
%   m-psk
%   stanag4285
%
%Permitted HF Channel Models:
%   TBD
%
%  Input Parameters:
%    inputProfile - string variable containing the name of the .mat file that contains
%                   the simulation profile.  The input profile is a .mat file that contains
%                   details of the simulation.  This file can be created with the profile tool
    .
%                   If inputProfile is empty, a profile selection window will appear to allow
%                   the user to select a profile.
%  Output Parameters:
%    SUCC - 0 if the function completed successfully, otherwise 1.
%  Input Files:
%    As specified by inputFile
%  Output Files:
%    As specified by outputFile
%  Special Notes:
%    See comments in code.
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
```

```
%Date          Version   Editor            Changes Made
%-------------  --------  ----------------  --------------------
function SUCC=rxsim(inputProfile)
try
    %assume successful operation
    SUCC=0;

    %get profile
    if exist('inputProfile')
        load(inputProfile);
    else
        %if the profile wasn't passed as an argument, the use the gui
        [filename,path2file]=uigetfile('profile.mat','SIGN(s): Open Profile');
        if isequal(filename,0) | isequal(path2file,0) %check for errors
            disp('SIGN(s): File Open Error or Cancel Selected');
            SUCC=1;
            return
        else
            %prepare profile string
            filename=strjust(strjust(filename,'left'),'right'); %remove possible white space
            path2file=strjust(strjust(path2file,'left'),'right'); %remove possible white space
            inputProfile=[path2file,filename];

            %get key variables
            load(inputProfile);
        end
    end

    %create waitbar
    H=waitbar(0,'Working.  Please wait...');
    set(H,'Name','Sign(s):  RXSIM');

    %get other inputfiles
    load(CFG_datagen_outFile);
    load(CFG_rfsim_outFile);

    %compute FIR and introduced symbol delays through simulator
    FIRdelay=ceil((CFG_TxBPForder+CFG_RxBPForder+CFG_RxLPForder+3)/2);
    SYMdelay=ceil(SIG_DLY/CFG_R*CFG_Fs);

    %loop for each RF signal
    sz=size(SIG_RF);
    rows=sz(1);
    cols=sz(2);

    %check bypass variable
    if ~CFG_BYPASS
        for k=1:rows
            for m=1:cols
                %determine mixer tuning frequency
```

```
RxTune=CFG_RxFc; %default tuning frequency
if CFG_RxEstFcON
    %work on the appropriate sideband to estimate carrier
    switch CFG_RxSB
    case 'dsb',
    case 'usb',
    case 'lsb',
    otherwise
    end
end

%apply input BPF
sig=filter(fir1(CFG_RxBPForder,[(RxTune+CFG_RxBPFOFF-CFG_RxBPFBW/2)/CFG_Fs*2,(
    RxTune+CFG_RxBPFOFF+CFG_RxBPFBW/2)/CFG_Fs*2],...
    'bandpass',eval(['window(',CFG_RxBPFwin,',',num2str(CFG_RxBPForder+1),')'
        ])),1,SIG_RF{k,m});
SIG_RxIF{k,m}=sig;

%mix down received signal
switch CFG_RxSB
case 'dsb',
    %perform mix
    Rx=signs_mixer(SIG_RxIF{k,m},RxTune/CFG_Fs*2,0);
    %apply baseband LPF
    sig=filter(fir1(CFG_RxLPForder,CFG_RxLPFBW/CFG_Fs*2,'low',...
        eval(['window(',CFG_RxLPFwin,',',num2str(CFG_RxLPForder+1),')'])),1,Rx
            );
case 'usb',
    %perform mix
    I=signs_mixer(SIG_RxIF{k,m},RxTune/CFG_Fs*2,0);
    Q=signs_mixer(SIG_RxIF{k,m},RxTune/CFG_Fs*2,-pi/2);
    %apply baseband LPF
    Ib=filter(fir1(CFG_RxLPForder,CFG_RxLPFBW/CFG_Fs*2,'low',...
        eval(['window(',CFG_RxLPFwin,',',num2str(CFG_RxLPForder+1),')'])),1,I)
            ;
    Qb=filter(fir1(CFG_RxLPForder,CFG_RxLPFBW/CFG_Fs*2,'low',...
        eval(['window(',CFG_RxLPFwin,',',num2str(CFG_RxLPForder+1),')'])),1,Q)
            ;
    sig=Ib+imag(hilbert(Qb));
case 'lsb',
    %perform mix
    I=signs_mixer(SIG_RxIF{k,m},RxTune/CFG_Fs*2,0);
    Q=signs_mixer(SIG_RxIF{k,m},RxTune/CFG_Fs*2,-pi/2);
    %apply baseband LPF
    Ib=filter(fir1(CFG_RxLPForder,CFG_RxLPFBW/CFG_Fs*2,'low',...
        eval(['window(',CFG_RxLPFwin,',',num2str(CFG_RxLPForder+1),')'])),1,I)
            ;
    Qb=filter(fir1(CFG_RxLPForder,CFG_RxLPFBW/CFG_Fs*2,'low',...
        eval(['window(',CFG_RxLPFwin,',',num2str(CFG_RxLPForder+1),')'])),1,Q)
            ;
```

```
                    sig=Ib-imag(hilbert(Qb));
                otherwise
                end

                %compensate for filter delays
                SIG_RxBS{k,m}=sig(SYMdelay+FIRdelay:end-SYMdelay+FIRdelay);
            end
            %update waitbar
            waitbar(((k-1)*cols+m)/rows/cols,H);
        end
    else
        %get txsim data
        load(CFG_txsim_outFile)
        for k=1:rows
            for m=1:cols
                SIG_RxIF{k,m}=[];
                SIG_RxBS{k,m}=SIG_TxBS{k}; %set received signal to tx baseband signal
                %update waitbar
                waitbar(((k-1)*cols+m)/rows/cols,H);
            end
        end
    end

    %clean up
    clear SIG_RF

    %save signals
    save(CFG_rxsim_outFile,'SIG_RxIF','SIG_RxBS');

    %close waitbar
    close(H);
catch
    %log error condition
    signs_log(CFG_logfile,dbstack);
    SUCC=1;
    close all;
end
return
```

TXSIM

```
%Sign(s) Module: TXSIM
%
%SUCC=TXSIM(inputProfile)
%
%The purpose of this function is to generate two baseband signals.  One becomes
%an ideal reference, while the other is mixed up to the carrier frequency
%specified by the input profile.
%
%This routine expects a .mat configuration file as an input and the output is
%a .mat file containing the ideal reference signal, the filtered Tx baseband
%signal, and the filtered Tx transmit signal.
%
%The algorithm for this routine is simply
%
%   for each message created by datagen
%      determine what the ideal received baseband signal should be
%      determine the baseband signal to be transmitted
%      apply low-pass filter to baseband signal to be transmitted
%      mix up the baseband Tx signal to the carrier frequency
%      band-pass filter the Tx signal at transmission frequency
%
%The routine is completely controlled by the contents of the input configuration file
%messages created by datagen.  See the example configuration files for more information.
%The variables defined in the input configuration file depend on the type of modulation
%for the hf signal, the simulation run parameters, and output control parameters.
%
%   Permitted Modulation Types:
%     m-fsk          - M-ary continuous phase FSK
%     m-psk          - M-ary PSK
%     stanag4285     - 8-PSK modulation method defined by NATO Standardization Agreement 4285
%     am             - amplitude modulated audio
%     fm             - frequency modulated audio
%     static         - no message transmitted ("just static" or white noise)
%
%  Input Parameters:
%    inputProfile - string variable containing the name of the .mat file that contains
%                   the simulation profile.  The input profile is a .mat file that contains
%                   details of the simulation.  This file can be created with the profile tool
    .
%                   If inputProfile is empty, a profile selection window will appear to allow
%                   the user to select a profile.
%  Output Parameters:
%    SUCC - 0 if the function completed successfully, otherwise 1.
%  Input Files:
%    As specified by inputProfile
%  Output Files:
%    As specified by inputProfile
%  Special Notes:
```

```
%    The output file contains three variables representing the generated signals:
%       SIG_REF      -    ideal baseband reference signal
%       SIG_TxBS     -    filtered baseband transmit signal
%       SIG_TX       -    filtered passband transmit signal at Fc
%    See additional comments in code.
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
%Date         Version   Editor             Changes Made
%----------   -------   ---------------    ---------------------

function SUCC=txsim(inputProfile)


try
    %get profile
    if exist('inputProfile')
        load(inputProfile);
    else
        %if the profile wasn't passed as an argument, the use the gui
        [filename,path2file]=uigetfile('profile.mat','SIGN(s): Open Profile');
        if isequal(filename,0) | isequal(path2file,0) %check for errors
            disp('SIGN(s): File Open Error or Cancel Selected');
            SUCC=1;
            return
        else
            %prepare profile string
            filename=strjust(strjust(filename,'left'),'right'); %remove possible white space
            path2file=strjust(strjust(path2file,'left'),'right'); %remove possible white space
            inputProfile=[path2file,filename];

            %get key variables
            load(inputProfile);
        end
    end

    %assume successful operation
    SUCC=0;

    %get other input files
    load(CFG_datagen_outFile);

    %create waitbar
    H=waitbar(0,'Working...  Please wait...');
    set(H,'Name','Sign(s):  TXSIM');

    %create signals
    Ks=size(SIG_MSG,1);
    for k=1:Ks
```

```
%get the message symbols created by datagen
sym1=SIG_MSG{k,1};
sym2=SIG_MSG{k,2};

%create signals
switch CFG_MODL
case 'm-fsk',
    %modulate the bits of the ideal reference signal
    SIG_REF{k,1}=signs_fsk(sym1,CFG_R,CFG_Fctr,CFG_Fs,CFG_Fdev,CFG_LVL,CFG_STYP);

    %modulate the bits of the baseband signal
    SIG_TxBS{k,1}=signs_fsk(sym2,CFG_R,CFG_Fctr,CFG_Fs,CFG_Fdev,CFG_LVL,CFG_STYP);
case 'm-psk',
    %modulate the bits of the ideal reference signal
    SIG_REF{k,1}=signs_psk(sym1,CFG_R,CFG_Fctr,CFG_Fs,CFG_Po,CFG_LVL,CFG_STYP);

    %modulate the bits of the baseband signal
    SIG_TxBS{k,1}=signs_psk(sym2,CFG_R,CFG_Fctr,CFG_Fs,CFG_Po,CFG_LVL,CFG_STYP);
case 'stanag4285',
    %modulate the bits of the ideal reference signal
    SIG_REF{k,1}=signs_st4285(sym1,CFG_R,CFG_Fs,CFG_STYP);

    %modulate the bits of the baseband signal
    SIG_TxBS{k,1}=signs_st4285(sym2,CFG_R,CFG_Fs,CFG_STYP);
case 'am',
case 'fm',
case 'static',
case 'custom',
otherwise,
    %return an error
    SUCC=1;
    return
end

%check bypass variable
if ~CFG_BYPASS
    %choose sidebands
    switch CFG_TxSB
    case 'dsb',
        %don't do anything for double sideband
    case 'usb',
        %generate USB
        SIG_TxBS{k}=SIG_TxBS{k}+j*imag(hilbert(SIG_TxBS{k}));
    case 'lsb',
        %generate LSB
        SIG_TxBS{k}=SIG_TxBS{k}-j*imag(hilbert(SIG_TxBS{k}));
    end

    %mix up transmitted signal
    SIG_TxIF{k,1}=signs_mixer(SIG_TxBS{k},CFG_TxFc/CFG_Fs*2,0);
```

```
            %apply output BPF
            switch CFG_TxSB
            case 'dsb',
                %keep double sideband (ignore CFG_TxBPFOFF)
                sig=filter(fir1(CFG_TxBPForder,[(CFG_TxFc-CFG_TxBPFOFF-CFG_TxBPFBW/2)/CFG_Fs
                    *2,(CFG_TxFc+CFG_TxBPFOFF+CFG_TxBPFBW/2)/CFG_Fs*2],...
                    'bandpass',eval(['window(',CFG_TxBPFwin,',',num2str(CFG_TxBPForder+1),')'
                        ])),1,SIG_TxIF{k});
            case 'usb',
                %filter off lower sideband (i.e. keep upper sideband)
                sig=filter(fir1(CFG_TxBPForder,[(CFG_TxFc+CFG_TxBPFOFF-CFG_TxBPFBW/2)/CFG_Fs
                    *2,(CFG_TxFc+CFG_TxBPFOFF+CFG_TxBPFBW/2)/CFG_Fs*2],...
                    'bandpass',eval(['window(',CFG_TxBPFwin,',',num2str(CFG_TxBPForder+1),')'
                        ])),1,SIG_TxIF{k});
            case 'lsb',
                %filter off upper sideband (i.e. keep lower sideband)
                sig=filter(fir1(CFG_TxBPForder,[(CFG_TxFc-CFG_TxBPFOFF-CFG_TxBPFBW/2)/CFG_Fs
                    *2,(CFG_TxFc-CFG_TxBPFOFF+CFG_TxBPFBW/2)/CFG_Fs*2],...
                    'bandpass',eval(['window(',CFG_TxBPFwin,',',num2str(CFG_TxBPForder+1),')'
                        ])),1,SIG_TxIF{k});
            end
            SIG_TxRF{k,1}=sig;
        else
            SIG_TxIF{k,1}=[];
            SIG_TxRF{k,1}=[];
        end

        %update waitbar
        waitbar(k/Ks,H);
    end

    %save signals
    save(CFG_txsim_outFile,'SIG_REF','SIG_TxBS','SIG_TxIF','SIG_TxRF');

    %close waitbar
    close(H);
catch
    %log error condition
    signs_log(CFG_logfile,dbstack);
    SUCC=1;
    close all;
end
return
```

XSWITCH

```
%Sign(s) Module: XSWITCH
%
%SUCC=XSWITCH(inputProfile)
%
%The purpose of this function is to correctly format and arrange data from either the
%synthetic source (i.e. txsim) or alternate data source (i.e. data stored in a file).
%
%  Input Parameters:
%    inputProfile - string variable containing the name of the .mat file that contains
%                   the simulation profile.  The input profile is a .mat file that contains
%                   details of the simulation.  This file can be created with the profile tool
    .
%                   If inputProfile is empty, a profile selection window will appear to allow
%                   the user to select a profile.
%  Output Parameters:
%    SUCC - 0 if the function completed successfully, otherwise 1.
%  Input Files:
%    As specified by inputFile
%  Output Files:
%    As specified by outputFile
%  Special Notes:
%    See comments in code.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%-----------  -------  ---------------    ---------------------

function SUCC=xswitch(inputProfile)
try
    %get profile
    if exist('inputProfile')
        load(inputProfile);
    else
        %if the profile wasn't passed as an argument, the use the gui
        [filename,path2file]=uigetfile('profile.mat','SIGN(s): Open Profile');
        if isequal(filename,0) | isequal(path2file,0) %check for errors
            disp('SIGN(s): File Open Error or Cancel Selected');
            SUCC=1;
            return
        else
            %prepare profile string
            filename=strjust(strjust(filename,'left'),'right'); %remove possible white space
            path2file=strjust(strjust(path2file,'left'),'right'); %remove possible white space
            inputProfile=[path2file,filename];

            %get key variables
```

```
        load(inputProfile);
    end
end

%assume successful operation
SUCC=0;

%create waitbar
H=waitbar(0,'Preparing data.  Please wait...');
set(H,'Name','Sign(s):  XSWITCH');

%check for real data
if CFG_REALRX | CFG_REALREF

    if CFG_REALRX
        %read the real data for SIG_RxBS
        [data,ERR]=signs_read(CFG_realFileRX,CFG_realPrecision,CFG_realMask,
            CFG_realComponent);
        realRxBS={data};
        SIG_realRxBS=realRxBS;
    end

    %update waitbar
    waitbar(.25,H);

    if CFG_REALREF
        %read real data for SIG_REF
        [data,ERR]=signs_read(CFG_realFileREF,CFG_realPrecision,CFG_realMask,
            CFG_realComponent);
        realREF={data};
        SIG_realREF=realREF;
    end

    %update waitbar
    waitbar(.5,H);

    %align data vectors
    if CFG_REALRX & ~CFG_REALREF %align SIG_REF with real rx data

        %get other synthetic reference signal
        load(CFG_txsim_outFile,'SIG_REF');

        %align data vectors
        for k=1:length(SIG_REF)
            if length(SIG_REF{k})>length(realRxBS{1})
                SIG_REF{k}=SIG_REF{k}(1:length(realRxBS{1}));
                SIG_RxBS{k,1}=realRxBS{1};
            else
                data=realRxBS{1};
```

```
                SIG_RxBS{k,1}=reshape(data(1:length(SIG_REF{1})),size(SIG_REF{1},1),size(
                    SIG_REF{1},2));
            end
        end
    elseif ~CFG_REALRX & CFG_REALREF %align synthetic rx data with real reference

        %get other synthetic reference signal
        load(CFG_rxsim_outFile,'SIG_RxBS');

        %align data vectors
        for k=1:size(SIG_RxBS,1)
            for m=1:size(SIG_RxBS,2)
                if length(SIG_RxBS{k})>length(realREF{1})
                    SIG_RxBS{k,m}=SIG_RxBS{k,m}(1:length(realREF{1}));
                    SIG_REF{k,1}=realREF{1};
                else
                    data=realREF{1};
                    SIG_REF{k,1}=reshape(data(1:length(SIG_RxBS{1,1})),size(SIG_RxBS
                        {1,1},1),size(SIG_RxBS{1,1},2));
                end
            end
        end
    else %align both real rx data and real reference
        SIG_RxBS=realRxBS;
        SIG_REF=realREF;
        %truncate whichever vector is longest
        if length(SIG_REF{1})>length(SIG_RxBS{1})
            SIG_REF{1}=SIG_REF{1}(1:length(SIG_RxBS{1}));
        else
            SIG_RxBS{1}=SIG_RxBS{1}(1:length(SIG_REF{1}));
        end
    end

    %update waitbar
    waitbar(1,H);

else %if no real data then align the synthetic rx data and reference

    %load the output of rxsim
    load(CFG_txsim_outFile,'SIG_REF');
    load(CFG_rxsim_outFile,'SIG_RxBS');

    %truncate whichever vector is longest
    sz=size(SIG_RxBS);
    for k=1:sz(1)
        for m=1:sz(2)
            if length(SIG_REF{k})>length(SIG_RxBS{k,m})
                SIG_REF{k}=SIG_REF{k}(1:length(SIG_RxBS{k,m}));
            else
                SIG_RxBS{k,m}=SIG_RxBS{k,m}(1:length(SIG_REF{k}));
```

```
                end

                %update waitbar
                waitbar(((k-1)*sz(2)+m)/sz(1)/sz(2),H);
            end
        end
    end

    %save data
    save(CFG_xswitch_outFile,'SIG_*');

    %cleanup
    close(H)
catch
    %log error condition
    signs_log(CFG_logfile,dbstack);
    SUCC=1;
    close all;
end
return
```

H.4 Ⓢigns Data Generation

The data generation tool is used to create binary strings for the Ⓢ**igns** transmit module.

DATAGEN

```
%Sign(s) Module: DATAGEN
%
%SUCC=DATAGEN(inputProfile)
%
%The purpose of this function is to prepare data for input to an analysis tool.
%The function provides a method for generating/modifying data prior to its
%inclusion in a subsequent stage of analysis by the analysis tools.  For example,
%DATAGEN can read the output file from TXSIM and modify the data before RXSIM reads
%that data from the TXSIM output file.
%
%This routine expects a .mat configuration file as an input and the output is
%a .mat file appropriate for the next analysis tool.
%
%  Input Parameters:
%    inputProfile - string variable containing the name of the .mat file that contains
%                   the simulation profile.  The input profile is a .mat file that contains
%                   details of the simulation.  This file can be created with the profile tool
    .
%                   If inputProfile is empty, a profile selection window will appear to allow
%                   the user to select a profile.
%  Output Parameters:
%    SUCC - 0 if the function completed successfully, otherwise 1.
%  Input Files:
%    As specified by inputProfile
%  Output Files:
%    As specified by inputProfile
%  Special Notes:
%    Currently, datagen creates signals based on variations in hamming distance.
%    See additional comments in code.
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
%Date         Version   Editor              Changes Made
%-----------  --------  ----------------    --------------------

function SUCC=datagen(inputProfile)
try
    %assume successful operation
    SUCC=0;
```

```
%get profile
if exist('inputProfile')
    load(inputProfile);
else
    %if the profile wasn't passed as an argument, the use the gui
    [filename,path2file]=uigetfile('profile.mat','SIGN(s) : Open Profile');
    if isequal(filename,0) | isequal(path2file,0) %check for errors
        disp('SIGN(s) : File Open Error or Cancel Selected');
        SUCC=1;
        return
    else
        %prepare profile string
        filename=strjust(strjust(filename,'left'),'right'); %remove possible white space
        path2file=strjust(strjust(path2file,'left'),'right'); %remove possible white space
        inputProfile=[path2file,filename];

        %get key variables
        load(inputProfile);
    end
end

%compute key values
Ts=1/CFG_Fs; %sampling period
SIG_N=round(CFG_T/Ts); %number of samples at sampling rate (needs to be large to have good
    frequency resolution)
SIG_SYM=fix(CFG_R*CFG_T); %number of bits
SIG_HAM=unique(fix(CFG_HAM*SIG_SYM)); %determine specific hamming distances

%compensate for filter delays by adding training periods
DLYTx=CFG_R*CFG_TxBPForder/CFG_Fs; %get additional symbols for tx BPF
DLYRxbpf=CFG_R*CFG_RxBPForder/CFG_Fs; %get additional symbols for rx BPF
DLYRxlpf=CFG_R*CFG_RxLPForder/CFG_Fs; %get additional symbols for rx LPF
SIG_DLY=ceil(DLYTx+DLYRxbpf+DLYRxlpf); %get number of additional symbols

for k=1:length(SIG_HAM)

    %generate data symbols for SIG1
    switch CFG_Dpdf,
    case 'uniform',
        SIG_MSG{k,1}=floor(rand(1,SIG_SYM)*2);
    case 'normal',
        SIG_MSG{k,1}=floor((randn(1,SIG_SYM)*CFG_Dstd+CFG_Dmu)*2);
    end

    %generate data symbols for SIG2
    sig=signs_varham(SIG_MSG{k,1},SIG_HAM(k));
    SIG_MSG{k,2}=sig;

    %check bypass variable
```

```
        if ~CFG_BYPASS
            %compensate for filter delays
            SIG_MSG{k,2}=[zeros(1,SIG_DLY),sig,zeros(1,SIG_DLY)];
        end
    end

    %save data
    save(CFG_datagen_outFile,'SIG_*');
catch
    %log error condition
    signs_log(CFG_logfile,dbstack);
    SUCC=1;
    close all;
end
return
```

H.5 Ⓢigns Feature Extraction Tools

Feature extraction tools are functions that can be used to form feature vectors. There are only four tools in this section corresponding to the coherence-median-difference (CMD), coherence, entropic distance, and power spectral density. No extraction module exists for the SNR estimator. A miscellaneous function, **signs_snr.m**, exists for this signal feature.

EXT_CMD

```
%Signs Feature:  EXT_CMD
%
%[S,ERR]=EXT_CMD(Cxy,F,Fi,WIN)
%
%The purpose of this function is to compute the coherence-median difference (CMD).
%The CMD is defined in Giesbrecht, et. al; "Modulation Recognition for HF Signals";
%2004, SPIE 5649-75.  The CMD is a measure of the dominance of the coherence at one
%or more frequencies over the coherence at all other frequencies in a bandwidth.
%
%The generalized CMD is defined as
%
%   gCMD = Cxy(for all f in a specified bandwidth) - median(Cxy(for all f in a specified
    bandwidth)).
%
%The specific CMD is defined as
%
%   CMD = mean(Cxy(for specific frequencies)) - median(Cxy(for all f in a specified bandwidth)
    ).
%
%Essentially, the CMD for specific frequencies is also
%
%   CMD = mean(gCMD(for specific frequencies)).
%
%Consequently, this function computes gCMD and CMD from gCMD.
%
%Mandatory Input Parameters:
%   Cxy -       coherence vector
%   F -         frequencies for each coherence value
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   Fi -        vector of specific frequencies for the specific CMD.  If Fi is not
%               specified the output CMD will be the generalized CMD.  If Fi is
%               specified the output CMD will be the specific CMD.  EXT_CMD uses
%               linear interpolation to determine the coherence at Fi not equal to
%               any element of F.
```

```
%   WIN -       averaging window for output CMD (e.g. a 7 sample boxcar window is
%               specified by WIN = '@rectwin,7'; see Matlab's window command for
%               more window options).  If WIN is specified the specific CMD and
%               parameters in S are computed on the windowed gCMD.
%
%Mandatory Output Parameters:
%   S -         Matlab structure containing the following fields:
%                   Parameter           - parameter extracted (in this case 'cmd')
%                   gCMD                - general CMD values for all frequencies
%                   Freq                - evaluation frequencies for gCMD
%                   CMD                 - specific CMD (null if Fi not specified)
%                   Fspc                - evaluation frequencies for specific CMD (equal to Fi
    )
%                   MxgCMD              - peak value of gCMD
%                   MngCMD              - minimum value of gCMD
%                   MxgCMDFreq          - frequencies of peak gCMD
%                   MngCMDFreq          - frequencies of minimum gCMD
%                   Median              - median gCMD
%                   Mean                - average gCMD
%                   Std                 - standard deviation of gCMD
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   If an error occurs in the function S will be unity.
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
%Date        Version   Editor          Changes Made
%-----------  -------  -------------   ---------------------

function [S,ERR]=ext_cmd(Cxy,F,varargin)

ERR=[];
try
    CMD=[]; %default CMD
    Fi=[]; %default specific frequencies for CMD

    %get optional parameters Fi and WIN
    if nargin>2 & nargin<5
        for i=1:length(varargin)
            arg=varargin{i};
            %check for window paramater
            if ischar(arg)
                WIN=arg;
            %check for Fi
```

```
        elseif isnumeric(arg)
            Fi=arg; %get specific frequencies
        end
    end
end

%compute the generalized coherence-median difference
gCMD=Cxy-median(Cxy);

%apply averaging window if required
if exist('WIN')==1
    [gCMD,ERR]=signs_apwin(gCMD,WIN);
end

%check for errors
if ~isempty(ERR)
    S=[];
    return
end

%make gCMD a 1xN array
gCMD=reshape(gCMD,1,prod(size(gCMD)));

%if specific frequencies are of interest compute the specific CMD
if exist('Fi')
    %get mean of gCMD at all Fi
    CMD=0;
    delF=F(2)-F(1);
    for i=1:length(Fi);
        %find adjacent frequency bins of the symbol frequency
        idx_lo=max(find(F<=Fi(i)));
        idx_hi=min(find(F>=Fi(i)));
        if idx_lo~=idx_hi
            idx=[idx_lo,idx_hi];
            %interpolate to get CMD at mark and space frequencies with
            %                        |  *  -----
            % ------------------     *  .    ^
            %     ^                  |  .    |
            %     |    -----    *    |  .    |
            %     |       ^     .    |  .    y1
            %     y       |     .    |  .    |
            %     |       y2    .    |  .    |
            %     |       |     .    |  .    |
            % ---------------------------------------
            %                  f2   f   f1
            %
            %      {(f-f2)*y1 + (f1-f)*y2}
            % y = ------------------------
            %          (f-f2) + (f1-f)
            Fdiff=[Fi(i)-min(F(idx)),max(F(idx))-Fi(i)];
```

```
                coh_freq=Fdiff*gCMD(idx)'/(Fdiff*[1,1]');
                CMD=coh_freq/length(Fi)+CMD;
            else
                %Fi is right on a frequency bin
                CMD=gCMD(idx_lo)/length(Fi)+CMD;
            end
        end
    end

    %find CMD statistics
    maxgCMD=max(max(gCMD));                 %peak gCMD value
    mingCMD=min(min(gCMD));                 %minimum gCMD value
    maxgCMDf=F(find(gCMD>=maxgCMD));        %frequencies of peak gCMD
    mingCMDf=F(find(gCMD<=mingCMD));        %frequencies of minimum gCMD
    medgCMD=median(gCMD);                   %median of gCMD values
    avggCMD=mean(gCMD);                     %average of gCMD values
    stdgCMD=std(gCMD);                      %standard deviation of gCMD values

    %return the results
    S=struct(...
        'Parameter','cmd',...               %parameter type
        'gCMD',gCMD,...                     %CMD values for specific frequencies
        'Freq',F,...                        %evaluation frequencies for gCMD
        'CMD',CMD,...                       %CMD for FSK signals
        'Fspc',Fi,...                       %specific evaluation frequencies for CMD
        'MxgCMD',maxgCMD,...                %peak value of gCMD
        'MngCMD',mingCMD,...                %minimum value of gCMD
        'MxgCMDFreq',maxgCMDf,...           %frequencies of peak gCMD
        'MngCMDFreq',mingCMDf,...           %frequencies of minimum gCMD
        'Median',medgCMD,...                %median gCMD
        'Mean',avggCMD,...                  %average gCMD
        'Std',stdgCMD);                     %standard deviation of gCMD
catch
    %capture error condition
    ERR={dbstack,lasterr};
    S=[];
end
return
```

EXT_COHR

```
%Sign(s) Feature:  EXT_COHR
%
%[S,ERR]=EXT_COHR(SIG,SIGREF,Fs,WIN,[K OVERLAP])
%
%The purpose of this function is to compute the coherence between a reference
%signal and a signal of interest.  Coherence is computed using Welch's periodgram
%method and Carter's considerations on coherence estimations.
%
%Coherence is defined as the ratio of cross-power spectrum squared to the product of
%two auto-power spectrums, such that
%
%            | Pxy (f) | ^2
%    Cxy = ------------------- , where 0 < Cxy < 1.
%            Pxx(f) Pyy(f)
%
%As Carter points out, this function generally requres estimation and therefore requires
%sub-dividing the time-series data into overlapping segments.  In so doing, the bias on the
%coherence estimate decreases with increasing number of overlapped sections and also
%decreases with increasing amount of overlap of sections up to approx. 50% overlap.  The
%variance on the coherence estimate also decreases with increasing number of overlapped
    sections
%and the mean square error also descreases with increasing number of sections (see  Carter,
%G. Clifford; COHERENCE AND TIME DELAY ESTIMATION; 1992, Ieee Press).  Hence the segment size,
%FFT size, and dataset size must be chosen so that the coherence estimate is accurate.
%
%Mandatory Input Parameters:
%    SIG -        vector representing a received signal
%    SIGREF -     vector representing the signal to which SIG is compared
%    Fs -         sampling rate in Hz
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%    WIN -        averaging window for coherence output (e.g. a 7 sample boxcar window is
%                 specified by WIN = '@rectwin,7'; see Matlab's window command for more window
    options).
%                 If WIN is specified, parameters in S are computed on the windowed coherence.
%    [K OVERLAP]-two-element vector indicating the number of desired segments and the percent
%                 overlap of the segments.  The first element, K, is the number of segments and
    the
%                 second element, OVERLAP, is the percentage overlap.  If K and OVERLAP are not
%                 specified the default is K=64 and OVERLAP=50%.
%
%Mandatory Output Parameters:
%    S -          Matlab structure containing the following fields:
%                     Parameter           - parameter extracted (in this case 'coherence')
%                     Cxy                 - coherence values for each frequency
%                     Freq                - frequency for each coherence value
%                     MxCxy               - maximum coherence
%                     MnCxy               - minimum coherence
```

```
%                   MxCxyFreq       - frequency corresponding to the maximum coherence
%                   MnCxyFreq       - frequency corresponding to the minimum coherence
%                   Median          - median coherence
%                   Mean            - average coherence
%                   Std             - standard deviation of coherence
%                   SegSize         - number of samples in each overlapping segment
%                   OverLap         - number of samples of overlap in each segment
%                   NFFT            - size of FFT for each segment
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   If an error occurs in the function the output will be unity.
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
%Date         Version   Editor          Changes Made
%-----------  --------  ------------    ---------------------
%

function [S,ERR]=ext_cohr(SIG,SIGREF,Fs,varargin)

ERR=[];
try
    %set segmenting and overlap defaults
    K=64;
    alpha=0.5;

    %get optional parameters
    if nargin>3 & nargin<6
        for i=1:length(varargin)
            arg=varargin{i};
            %check for window paramater
            if ischar(arg)
                WIN=arg;
            %check for [K OVERLAP]
            elseif isnumeric(arg) & prod(size(arg))==2
                K=arg(1);           %get number of disjoint segments
                alpha=arg(2)/100;   %get percent overlap
            end
        end
    end

    %coherence with Welch's Method
    %bias on the coherence estimate decreases with increasing number of overlapped sections
```

```
%bias also decreases with increasing amount of overlap of sections up to approx. 50%
    overlap
%variance on the coherence estimate also decreases with increasing number of overlapped
    sections
%and the mean square error also descreases with increasing number of sections
%(ref:  Carter, G. Clifford; COHERENCE AND TIME DELAY ESTIMATION; 1992, Ieee Press)
%hence the WINDOWsize, FFTsize, and dataset size must be chosen so that the coherence
    estimate
%is accurate.  The number of overlapped sections in the cohere.m function is computed
    based on
%
%K = (N-NOVERLAP)/(WINDOWsize-NOVERLAP)
%
%The size of the FFT should be increased beyond the length of the window to ensure
    resolvability
%of the coherence estimate (see Carter).
%
X=SIGREF;
Y=SIG;

%Choose number of disjoint segments (i.e. K), NOVERLAP=X% of WINDOWsize (i.e. alpha), and
    FFTsize=2^nextpow2(WINDOWsize)
WINDOWsize=round(length(X)/(K*(1-alpha)+alpha));
FFTsize=2^(nextpow2(WINDOWsize)+2);
NOVERLAP=round(WINDOWsize*alpha);
DFLAG='none';

%calculate coherence
[Cxy,F]=cohere(X,Y,FFTsize,Fs,WINDOWsize,NOVERLAP,DFLAG);

%apply windowing function if necessary
if exist('WIN')==1
    [Cxy,ERR]=signs_apwin(Cxy,WIN);

    %check for errors
    if ~isempty(ERR)
        S=[];
        return
    end

    %adjust length of frequency vector to align with Cxy
    Lwin=length(eval(['window(',WIN,')']));
    if mod(Lwin,2)==0
        skip=Lwin/2;
        delF=F(2)-F(1);
        F=F(skip:length(Cxy)+skip-1)+delF/2; %adjust the value of F to account for even
            length window
    else
        skip=(Lwin-1)/2;
        F=F(skip+1:length(Cxy)+skip);
```

```
        end
    end

    %find coherence statistics
    maxCOH=max(max(Cxy));              %peak coherence value
    minCOH=min(min(Cxy));              %minimum coherence value
    maxCOHf=F(find(Cxy>=maxCOH));      %frequencies of peak coherence
    minCOHf=F(find(Cxy<=minCOH));      %frequencies of minimum coherence
    medCOH=median(Cxy);                %median of coherence values
    avgCOH=mean(Cxy);                  %average of coherence values
    stdCOH=std(Cxy);                   %standard deviation of coherence values

    %return the results
    S=struct(...
        'Parameter','coherence',...    %parameter type
        'Cxy',Cxy,...                  %coherence values for each frequency
        'Freq',F,...                   %evaluation frequencies
        'MxCxy',maxCOH,...             %peak value of coherence
        'MnCxy',minCOH,...             %minimum value of coherence
        'MxCxyFreq',maxCOHf,...        %frequencies of peak coherence
        'MnCxyFreq',minCOHf,...        %frequencies of minimum coherence
        'Median',medCOH,...            %median coherence
        'Mean',avgCOH,...              %average coherence
        'Std',stdCOH,...               %standard deviation of coherence
        'SegSize',WINDOWsize,...       %number of samples in each segment
        'OverLap',NOVERLAP,...         %number of samples of overlap in each segment
        'NFFT',FFTsize);               %number of FFT bins in each segment
catch
    %capture error condition
    ERR={dbstack,lasterr};
    S=[];
end
return
```

EXT_ENTROPY

```
%Sign(s) Feature:  EXT_ENTROPY
%
%[S,ERR]=EXT_ENTROPY(METH,A,P1,P2,...)
%
%The purpose of this function is to compute the entropy of a sequence A.
%
%Mandatory Input Parameters:
%   METH -      string indicating the method for entropy calculation
%               METH can be any one of the following strings
%                   'shannon'       - compute using C. E. Shannon's formula
%                   'benedetto'     - compute using Benedetto, Caglioti, & Loreto
%                                     compression method (see Special Notes)
%               Some methods may require additional parameters.
%   A -         arbitrary data vector on which entropy is to be computed
%
%Additional Input Parameters:
%   P1, P2, ... additional parameters as required based on METH:
%               METH        Extra Parameters
%               ---------   ------------------------------
%               shannon     P1=pdf of A.  If P1 isn't specified, the pdf is estimated
%               benedetto   P1=B where B is another sequence of equal length to A
%                           P2=L where L is the length of each appended sequence.  If 0<=L<=1
 then
%                              L is considered a percentage of the shorter of A and B;
%                              if however, L>1 then L is considered as an integer number of
 samples
%                              to append.  L can be a vector of percentages or integers (not
 both)
%                              and values of L<=0 are ignored.
%                           P3=Nmsg where Nmsg is the length of each signal sequence.  If 0<=
 Nmsg<=1 then
%                              Nmsg is considered a percentage of the shorter of A and B;
%                              if however, Nmsg>1 then Nmsg is considered as an integer
 number of samples.
%                              Nmsg can be a vector of percentages or integers (not both)
%                              and values of Nmsg<=0 are ignored.
%                           P4='lzw' for Lempel-Ziv Welch compression with 12-bit codes
%                                OR
%                           P4='zip23' for the Info-Zip Zip 2.3 compression algorithm that
 comes
%                              with Win2K. EXT_ENTROPY is not gauranteed to work with other
 zip versions.
%                           P5=NT where NT is the number of trials per calculation of entropy
%                           P6=optional sliding window to apply to entropy
%                           values (see HELP WINDOW), which is ignored if
%                           the length of the window exceeds the longest
%                           dimension of the resulting entropy arrays.
```

```
%                           P7=optional path for temporary files created if P4='zip23'
    otherwise ignore P7
%                           P8=optional type modifier 'int8','uint8','int16','uint16', or '
    double'.  This parameter
%                              directs EXT_ENTROPY to convert all elements of A to an 8- or
    16-bit integer or
%                              to leave each element as a double float.  The default is a
    double float.
%                              If the default is chosen, each byte of the 8-byte floats of A
    are compressed
%                              indepedently of the other bytes in the element.  If this is
    not the desired
%                              behaviour, then P8 must be one of the other options.
    Converting elements of
%                             A to an integer prior to compression ensures that all bits
    associated with
%                              any one element are compressed together.
%
%Mandatory Output Parameters:
%   S - will contain a measure of the entropy of A in a Matlab structure with the following
    fields
%
%           'Parameter'       -parameter type (always 'entropy')
%           'Method'          -entropy method (either 'shannon' or 'benedetto')
%           'Comp'            -compression method (either 'lzw' or 'zip23')
%           'S'               -total entropy for the 'shannon' method (will be a single
    value) OR
%                              entropic distance for each appending sequence length (rows)
    and
%                              length of each signal vector segment (columns)
%           'S_AA'            -benedetto method only: self-entropy of A for each appending
    sequence length (rows) and
%                              length of each signal vector segment (columns)
%           'S_BB'            -benedetto method only: self-entropy of B for each appending
    sequence length (rows) and
%                              length of each signal vector segment (columns)
%           'S_AB'            -benedetto method only: relative-entropy of A wrt B for each
    appending sequence length
%                             (rows) and length of each signal vector segment (columns)
%           'S_BA'            -benedetto method only: relative-entropy of B wrt A for each
    appending sequence length
%                             (rows) and length of each signal vector segment (columns)
%           'R_AB'            -benedetto method only: relative entropic distance of A wrt B
    for each appending sequence length
%                             (rows) and length of each signal vector segment (columns)
%           'R_BA'            -benedetto method only: relative entropic distance of B wrt A
    for each appending sequence length
%                             (rows) and length of each signal vector segment (columns)
%           'd_Aa'            -benedetto method only: differential length of A wrt a
%           'd_Ab'            -benedetto method only: differential length of A wrt b
```

```
%             'd_Ba'              -benedetto method only: differential length of B wrt a
%             'd_Bb'              -benedetto method only: differential length of B wrt b
%             'Lapp'              -benedetto method only: length of appending sequences
%             'Nmsg'              -benedetto method only: length of each signal vector segment
%             'Win'               -sliding window applied to data
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   See Physical Review Letters, Vol. 88 No. 4, 28 January 2002
%   for more information on the Benedetto method.
%   EXT_ENTROPY creates and deletes temporary data files if METH='benedetto' and P3='zip23'.
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
%Date        Version   Editor          Changes Made
%---------   -------   ------------    ---------------------
%17/03/05    1.01      JEG             Added description of output structure
%08/07/05    1.02      JEG             All data segments for Benedetto's entropy are now
    selected at random
%13/02/08    1.03      JEG             Fixed error in test for length of
%                                      main sequence, and shortend L by the
%                                      window length. Also tested for case
%                                      where the window is longer than data
function [S,ERR]=ext_entropy(METH,A,varargin)

ERR=[];
try
    switch METH
    case 'shannon',
        %get or estimate pdf
        if nargin>2
            %get pdf
            pdf=varargin{1};
        else
            %estimate pdf
            H=hist(A,length(A));
            idx=find(H~=0);
            pdf=H(idx)/length(idx);
        end
        pdf=reshape(pdf,1,prod(size(pdf)));

        %compute self-information

        I=-log(pdf)./log(length(pdf));
        I=reshape(I,prod(size(I)),1);
```

```
    %compute entropy
    TotEnt=pdf*I;

    %return the results
    S=struct(...
        'Parameter','entropy',...    %parameter type
        'Method','shannon',...       %entropy method
        'Comp',[],...                %compression method
        'S',TotEnt,...               %total entropy or entropic distance
        'S_AA',[],...                %self-entropy of A
        'S_BB',[],...                %self-entropy of B
        'S_AB',[],...                %relative entropy A wrt B
        'S_BA',[],...                %relative entropy B wrt A
        'd_Aa',[],...                %delta length of A wrt a
        'd_Ab',[],...                %delta length of A wrt b
        'd_Ba',[],...                %delta length of B wrt a
        'd_Bb',[],...                %delta length of B wrt b
        'Lapp',[],...                %length of appending sequences (rows)
        'Nmsg',[],...                %length of each message (columns)
        'Win',[]);                   %window type
case 'benedetto',

    %ensure first sequence is a horizontal vector
    A=reshape(A,1,length(A));

    %get second sequence
    B=varargin{1}; %get the second sequence
    B=reshape(B,1,length(B)); %ensure a horizontal vector

    %get the length of appending sequence
    if max(varargin{2})<=1 & min(varargin{2})>=0
        L=fix(varargin{2}*min(length(B),length(A))); %get percentage length of appending
            sequences
    else
        L=fix(varargin{2}); %get integer number of samples to append
    end
    L=L(find(L>0)); %remove zero and negative elements

    %determine the number of samples for each main sequence (i.e. A or B)
    if max(varargin{3})<=1 & min(varargin{3})>=0
        Nmsg=fix(varargin{3}*min(length(B),length(A))); %get percentage length of
            sequences
    else
        Nmsg=fix(varargin{3}); %get integer number of samples
    end
    Nmsg=Nmsg(find(Nmsg>0)); %remove zero and negative elements

    %get compression method
    CompressMeth=varargin{4}; %get compression method
```

```
%get number of trials
if isnumeric(varargin{5}) & size(varargin{5})==[1,1]
    NT=varargin{5}; %get number of trials per entropy calculation
else
    NT=1;
end

%parse optional parameters
win=''; %default sliding window
tempath=''; %default temporary path
typemod='double'; %default datatype modifier
if nargin>=8
    for j=6:min(nargin-2,8)
        if ischar(varargin{j})
            switch varargin{j}
            case {'int8','uint8','int16','uint16','double'},
                typemod=varargin{j}; %get data type modifier
            otherwise
                if varargin{j}(1)=='@'
                    win=varargin{j}; %get sliding window string
                else
                    tempath=varargin{j}; %get path for temporary files
                    if ~isempty(tempath)
                        if tempath(end)~='\'
                            tempath=[tempath,'\'];
                        end
                    end
                end
            end
        end
    end
end

%initialize arrays
L_A=zeros(length(L),length(Nmsg));
L_Aa=L_A;
L_Ab=L_A;
L_B=L_A;
L_Ba=L_A;
L_Bb=L_A;

%compute entropy for each length of appending sequence and each message length
for j=1:length(L)
    for k=1:length(Nmsg)
        for trial=1:NT

            %get random segments
            idx_start=round(rand(1)*length(B)+1);
```

```
b=B(mod([idx_start-1:idx_start+L(j)-2],length(B))+1); %get a small random
    sequence from B
idx_start=round(rand(1)*length(A)+1);
a=A(mod([idx_start-1:idx_start+L(j)-2],length(A))+1); %get a small random
    sequence from A
idx_start=round(rand(1)*length(B)+1);
SegB=B(mod([idx_start-1:idx_start+Nmsg(k)-2],length(B))+1); %get the main
    segment from B
idx_start=round(rand(1)*length(A)+1);
SegA=A(mod([idx_start-1:idx_start+Nmsg(k)-2],length(A))+1); %get the main
    segment from A

%choose correct datatype modification
switch typemod
case 'int8',
    b=int8(b);
    a=int8(a);
    SegB=int8(SegB);
    SegA=int8(SegA);
    SYM=8;
case 'uint8',
    b=uint8(b);
    a=uint8(a);
    SegB=uint8(SegB);
    SegA=uint8(SegA);
    SYM=8;
case 'int16',
    b=int16(b);
    a=int16(a);
    SegB=int16(SegB);
    SegA=int16(SegA);
    SYM=16;
case 'uint16',
    b=uint16(b);
    a=uint16(a);
    SegB=uint16(SegB);
    SegA=uint16(SegA);
    SYM=16;
case 'double',
    %do nothing if 'double'
    SYM=8;
otherwise
    %default is to do nothing
    SYM=8;
end

%perform the compression
switch CompressMeth
case 'lzw',    %compress using LZW algorithm
    %get compressed lengths
```

```
                L_A(j,k)=length(signs_lzw(SegA,'c',0,SYM))+L_A(j,k);
                L_Aa(j,k)=length(signs_lzw([SegA,a],'c',0,SYM))+L_Aa(j,k);
                L_Ab(j,k)=length(signs_lzw([SegA,b],'c',0,SYM))+L_Ab(j,k);
                L_B(j,k)=length(signs_lzw(SegB,'c',0,SYM))+L_B(j,k);
                L_Ba(j,k)=length(signs_lzw([SegB,a],'c',0,SYM))+L_Ba(j,k);
                L_Bb(j,k)=length(signs_lzw([SegB,b],'c',0,SYM))+L_Bb(j,k);
            case 'zip23', %compress using Info-ZIP's Zip 2.3 algorithm in Win2K
                %save data sequences
                fid=fopen([tempath,'L_A.tmp'],'w');
                fwrite(fid,SegA,typemod);
                fclose(fid);
                fid=fopen([tempath,'L_Aa.tmp'],'w');
                fwrite(fid,[SegA,a],typemod);
                fclose(fid);
                fid=fopen([tempath,'L_Ab.tmp'],'w');
                fwrite(fid,[SegA,b],typemod);
                fclose(fid);
                fid=fopen([tempath,'L_B.tmp'],'w');
                fwrite(fid,SegB,typemod);
                fclose(fid);
                fid=fopen([tempath,'L_Ba.tmp'],'w');
                fwrite(fid,[SegB,a],typemod);
                fclose(fid);
                fid=fopen([tempath,'L_Bb.tmp'],'w');
                fwrite(fid,[SegB,b],typemod);
                fclose(fid);

                %read length of zip files

                L_A(j,k)=L_A(j,k)+zippedlength(tempath,'L_A.tmp');

                L_Aa(j,k)=L_Aa(j,k)+zippedlength(tempath,'L_Aa.tmp');

                L_Ab(j,k)=L_Ab(j,k)+zippedlength(tempath,'L_Ab.tmp');

                L_B(j,k)=L_B(j,k)+zippedlength(tempath,'L_B.tmp');

                L_Ba(j,k)=L_Ba(j,k)+zippedlength(tempath,'L_Ba.tmp');

                L_Bb(j,k)=L_Bb(j,k)+zippedlength(tempath,'L_Bb.tmp');

                %clean up temporary files
                eval(['!del ',tempath,'L_*.*']);
            end
        end
    end
end

%find deltas
del_Ab=(L_Ab-L_A)/NT;
```

```
del_Aa=(L_Aa-L_A)/NT;
del_Bb=(L_Bb-L_B)/NT;
del_Ba=(L_Ba-L_B)/NT;

%find self entropies
S_AA=inv(diag(L))*del_Aa;
S_BB=inv(diag(L))*del_Bb;

%find relative entropies
S_AB=inv(diag(L))*(del_Ab-del_Bb);
S_BA=inv(diag(L))*(del_Ba-del_Aa);

%entropic distance
S_Dist=(del_Ab-del_Bb)./del_Bb+(del_Ba-del_Aa)./del_Aa;

%relative entropic distances
R_AB=S_AB./S_BB;
R_BA=S_BA./S_AA;

%compute windowed entropy
if ~isempty(win) & length(win)<=length(L) & length(win)<=length(Nmsg)
    %windowed entropic distance
    S_Dist=signs_apwin(S_Dist,win);

    %windowed relative entropies
    S_AB=signs_apwin(S_AB,win);
    S_BA=signs_apwin(S_BA,win);

    %windowed self-entropies
    S_AA=signs_apwin(S_AA,win);
    S_BB=signs_apwin(S_BB,win);

    %windowed deltas
    del_Ab=signs_apwin(del_Ab,win);
    del_Aa=signs_apwin(del_Aa,win);
    del_Ba=signs_apwin(del_Ba,win);
    del_Bb=signs_apwin(del_Bb,win);

    %shorten Nmsg and L to compensate for windowing
    sz=length(eval(['window(',win,')']));
    if mod(sz,2)==0
        st=sz/2;
    else
        st=(sz-1)/2;
    end
    Nmsg=Nmsg(st+1:end-st);
    L=L(st+1:end-st);
end

%return the results
```

```
        S=struct(...
            'Parameter','entropy',...    %parameter type
            'Method','benedetto',...     %entropy method
            'Comp',CompressMeth,...      %compression method
            'S',S_Dist,...               %total entropic (R_AB+R_BA) distance between A and B
            'R_AB',R_AB,...              %relative entropic distance from A to B
            'R_BA',R_BA,...              %relative entropic distance from B to A
            'S_AA',S_AA,...              %self-entropy of A
            'S_BB',S_BB,...              %self-entropy of B
            'S_AB',S_AB,...              %relative entropy A wrt B
            'S_BA',S_BA,...              %relative entropy B wrt A
            'd_Aa',del_Aa,...            %delta length of A wrt a
            'd_Ab',del_Ab,...            %delta length of A wrt b
            'd_Ba',del_Ba,...            %delta length of B wrt a
            'd_Bb',del_Bb,...            %delta length of B wrt b
            'Lapp',L,...                 %length of appending sequences (rows)
            'Nmsg',Nmsg,...              %length of each message (columns)
            'Win',win);                  %window type
    end
catch
    %close all open files
    fclose('all');

    %capture error condition
    ERR={dbstack,lasterr};
    S=[];
end
return

%ZIPPED LENGTH returns the length of a single compressed file in a zip archive.
%This function is guaranteed to work with Info-Zip's Zip 2.3 algorithm.
%It depends on the verbose option to extract the compressed length of the file.
%Other versions of zip may not have the same verbose output.
function [Lx,ERR]=zippedlength(tempath,FILE)
try
    %default value for Lx
    Lx=0;

    %generate zip file name
    idx=strfind(FILE,'.');
    ZIPFILE=[tempath,FILE(1:idx),'zip'];

    %compress the file and get output
    string=evalc(['!zip␣-q␣-m␣-j␣-9␣-D␣-J␣-X␣-v␣',ZIPFILE,'␣',tempath,FILE]);

    %extract compressed length from string output
  idx1=strfind(string,'out='); %find location of compressed length in output string
    if ~isempty(idx1)
        idx2=strfind(string(idx1(1)+4:end),')'); %find location of end of compressed length in
            output string
```

```
        if ~isempty(idx2)
            Lx=str2num(string(idx1(1)+4:idx1(1)+4-1+idx2(1)-1));
        else
            signs_log('stdout','***SIGN(s): Warning -- Cannot extract compressed length from
                zipped file');
        end
    else
        signs_log('stdout','***SIGN(s): Warning -- Cannot extract compressed length from
            zipped file');
    end
    disp(string)
    disp(['Lx=',num2str(Lx)])
    disp(' ')
catch
    %close all open files
    fclose('all');

    %capture error condition
    ERR={dbstack,lasterr};
    Lx=0;
end
return
```

EXT_PSD

```
%Signs Feature:  EXT_PSD
%
%[S,ERR]=EXT_PSD(SIG,NFFT,Fs,OVERLAP);
%
%The purpose of this function is to compute the power spectral density of a
%signal.  The PSD is computed using Welch's periodgram method.
%
%Mandatory Input Parameters:
%   SIG -       real or complex vector representing the signal on which to compute the psd
%   NFFT -      size of FFT for each overlapping segment of SIG
%   Fs -        sampling rate in Hz
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   OVERLAP -   indicates the percent overlap of the segments.  The default is 0.
%
%Mandatory Output Parameters:
%   S -         Matlab structure containing the following fields:
%                   Parameter           - parameter extracted (in this case 'psd')
%                   Pxx                 - psd values for each frequency
%                   Freq                - frequency for each psd value
%                   MxPxx               - maximum psd
%                   MnPxx               - minimum psd
%                   MxPxxFreq           - frequency corresponding to the maximum psd
%                   MnPxxFreq           - frequency corresponding to the minimum psd
%                   Median              - median psd
%                   Mean                - average psd
%                   Std                 - standard deviation of psd
%                   SegSize             - number of samples in each overlapping segment
%                   OverLap             - number of samples of overlap in each segment
%                   NFFT                - size of FFT for each segment
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   If an error occurs in the function the output will be unity.
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%----------  --------  ----------------  --------------------
%

function [S,ERR]=ext_psd(SIG,NFFT,Fs,varargin)
```

```
ERR=[];
try
    %get optional parameters
    if nargin==4
        arg=varargin{1};
        if isnumeric(arg)
            alpha=arg(1)/100;   %get percent overlap
        else
            %set overlap default
            alpha=0;
        end

        %psd with Welch's Method
        %The number of overlapped sections in the psd.m function is computed based on
        %
        %K = (N-NOVERLAP)/(WINDOWsize-NOVERLAP)
        %
        %Choose number of disjoint segments (i.e. K), NOVERLAP=X% of WINDOWsize (i.e. alpha),
            and FFTsize=2^nextpow2(WINDOWsize)
        WINDOWsize=NFFT; %number of elements in each window
        NOVERLAP=round(WINDOWsize*alpha);
        DFLAG='none';
    else
        WINDOWsize=NFFT;
        NOVERLAP=0;
        DFLAG='none';
    end

    [Pxx,F]=psd(SIG,NFFT,Fs,WINDOWsize,NOVERLAP,DFLAG);

    %find psd statistics
    maxPSD=max(max(Pxx));              %peak psd value
    minPSD=min(min(Pxx));              %minimum psd value
    maxPSDf=F(find(Pxx>=maxPSD));      %frequencies of peak psd
    minPSDf=F(find(Pxx<=minPSD));      %frequencies of minimum psd
    medPSD=median(Pxx);                %median of psd values
    avgPSD=mean(Pxx);                  %average of psd values
    stdPSD=std(Pxx);                   %standard deviation of psd values

    %return the results
    S=struct(...
        'Parameter','psd',... %parameter type
        'Pxx',Pxx,...                  %psd values for each frequency
        'Freq',F,...                   %evaluation frequencies
        'MxPxx',maxPSD,...             %peak value of psd
        'MnPxx',minPSD,...             %minimum value of psd
        'MxPxxFreq',maxPSDf,...        %frequencies of peak psd
        'MnPxxFreq',minPSDf,...        %frequencies of minimum psd
        'Median',medPSD,...            %median psd
        'Mean',avgPSD,...              %average psd
```

```
        'Std',stdPSD,...                %standard deviation of psd
        'SegSize',WINDOWsize,...        %number of samples in each segment
        'OverLap',NOVERLAP,...          %number of samples of overlap in each segment
        'NFFT',NFFT);                   %number of FFT bins in each segment
catch
    %display the error condition
    ERR={dbstack,lasterr};
    S=[];
end
return
```

H.6 Ⓢigns Reporting Tools

The reporting tools provide the ability for the Ⓢ**igns** package to display analysis results. Some of the reporting modules are used this thesis; others are implemented to provide useful reporting facilities for the toolbox.

REPORT

```
%Sign(s) Module: REPORT
%
%SUCC=REPORT(inputProfile)
%
%The purpose of this function is to report results of extracted features and/or
%results of modulation recognition algorithms.
%
%  Input Parameters:
%    inputProfile - string variable containing the name of the .mat file that contains
%                   the simulation profile.  The input profile is a .mat file that contains
%                   details of the simulation.  This file can be created with the profile tool
    .
%                   If inputProfile is empty, a profile selection window will appear to allow
%                   the user to select a profile.
%  Output Parameters:
%    SUCC - 0 if the function completed successfully, otherwise 1.
%  Special Notes:
%    See comments in code.
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
%Date        Version   Editor          Changes Made
%---------   -------   ------------    --------------------

function SUCC=report(inputProfile)
try
    %get profile
    if exist('inputProfile')
        load(inputProfile);
    else
        %if the profile wasn't passed as an argument, the use the gui
        [filename,path2file]=uigetfile('profile.mat','SIGN(s) : Open Profile');
        if isequal(filename,0) | isequal(path2file,0) %check for errors
            disp('SIGN(s) : File Open Error or Cancel Selected');
            SUCC=1;
            return
        else
            %prepare profile string
```

```
        filename=strjust(strjust(filename,'left'),'right'); %remove possible white space
        path2file=strjust(strjust(path2file,'left'),'right'); %remove possible white space
        inputProfile=[path2file,filename];

        %get key variables
        load(inputProfile);
    end
end

%assume successful operation
SUCC=0;

%load other profiles
load(CFG_datagen_outFile);

%read in report types
[RptDes,Rpt]=textread('signs_rpttypes.dat','%s%s','delimiter',',');
[SELECTION,OK]=listdlg('ListString',RptDes,'SelectionMode','multiple','ListSize'
    ,[460,400],...
    'Name','Reports','PromptString','Select␣one␣or␣more␣reports');

%execute the reports
for rptidx=1:length(SELECTION)
    switch Rpt{SELECTION(rptidx)}
    %Coherence versus SNR
    case 'cohsnr',
        if ~CFG_REALRX
            SUCC=rpt_cohsnr(inputProfile);
        end

    %Coherence versus SNR versus Frequency
    case 'chsnrf',
        if ~CFG_REALRX
            SUCC=rpt_chsnrf(inputProfile);
        end
    case 'cmdsnr',
        if ~CFG_REALRX
            SUCC=rpt_cmdsnr(inputProfile);
        end

    %Coherence versus Hamming distance
    case 'cohham',
        SUCC=rpt_cohham(inputProfile);

    %Coherence versus Hamming distance and Frequency
    case 'chhamf',
        SUCC=rpt_chhamf(inputProfile);
    case 'cmdham',
        SUCC=rpt_cmdham(inputProfile);
```

```
        %CMD versus Frequency
        case 'cmdfrq',
            SUCC=rpt_cmdfrq(inputProfile);

        %Generalized CMD versus Frequency
        case 'gcmdfr',
            SUCC=rpt_gcmdfr(inputProfile);

        %Coherence versus Frequency
        case 'cohfrq',
            SUCC=rpt_cohfrq(inputProfile);

        case 'enzprb',
            SUCC=rpt_enzprb(inputProfile);

        %Entropy versus Appending Length versus Number of Samples
        case 'ensamb',
            SUCC=rpt_ensamb(inputProfile);

        case 'clzprb',
            SUCC=rpt_clzprb(inputProfile);

        %Entropy versus sample number
        case 'entsam',
            SUCC=rpt_entsam(inputProfile);

        %Mean Entropic Distance versus Sample Number
        case 'medsam',
            SUCC=rpt_medsam(inputProfile);

        case 'entmsg',
            SUCC=rpt_entmsg(inputProfile);

        case 'entsnr',
            if ~CFG_REALRX
                SUCC=rpt_entsnr(inputProfile);
            end

        %Power spectrum
        case 'psds',
            SUCC=rpt_psds(inputProfile);
        otherwise
        end
    end
catch
    signs_log(CFG_logfile,dbstack); %log error condition
    SUCC=1;
end
return
```

RPT_CHHAMF

```
%Sign(s) Report: RPT_CHHAMF
%
%SUCC=RPT_CHHAMF(inputProfile)
%
%The purpose of this function is to report results of coherence versus Hamming distance and
    frequency.
%
%  Input Parameters:
%    inputProfile - string variable containing the name of the .mat file that contains
%                   the simulation profile.  The input profile is a .mat file that contains
%                   details of the simulation.  This file can be created with the profile tool
    .
%  Output Parameters:
%    SUCC - 0 if the function completed successfully, otherwise 1.
%  Special Notes:
%    See comments in code.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%---------   -------   ------------      --------------------

function SUCC=rpt_chhamf(inputProfile)
try
    %assume successful operation
    SUCC=0;

    %get profile
    load(inputProfile);

    %get other inputfiles
    load(CFG_extract_outFile)
    load(CFG_datagen_outFile);

    %get hamming distances at which to plot coherence versus snr versus frequency
    prompt={'Enter␣SNRs␣of␣interest␣(one␣plot␣for␣each␣SNR):'};
    def={num2str(CFG_SNR)};
    dlgTitle='Coherence␣vs␣Hamming␣Distance␣vs␣Frequency';
    lineNo=1;
    SNR=[];
    answer={'dummy'};
    while isempty(SNR) & ~isempty(answer)
        answer=inputdlg(prompt,dlgTitle,lineNo,def);
        if ~isempty(answer)
            [SNR,dummy,snridx]=intersect(str2num(answer{1}),CFG_SNR);
            if isempty(SNR) | length(str2num(answer{1}))>length(SNR)
                waitfor(errordlg({'SNR␣must␣be␣of␣the␣following␣set:',num2str(CFG_SNR)},...
```

```
                    'Invalid SNR','on'));
                def={answer{1}};
                SNR=[];
            end
        end
    end
    if ~isempty(answer)
        %plot results
        for j=1:length(snridx)
            figure
            surf(SIG_HAM,EXT_COH(1,1).Freq/signs_engunt(EXT_COH(1,1).Freq),[EXT_COH(:,snridx(j
                )).Cxy])
            xlabel('Hamming Distance');
            ylabel(['Frequency (',signs_engunt(EXT_COH(1,1).Freq,'prefix'),'Hz)']);
            zlabel('Coherence')
            view(CFG_AZ,CFG_EL);
            title(['Coherence between Received and Ideal Signals (HD=',num2str(CFG_SNR(snridx(
                j))),')'])
        end
    else
        SUCC=1;
    end
catch
    %log error condition
    signs_log(CFG_logfile,dbstack);
    SUCC=1;
end
return
```

RPT_CHSNRF

```
%Sign(s) Report: RPT_CHSNRF
%
%SUCC=RPT_CHSNRF(inputProfile)
%
%The purpose of this function is to report results of coherence versus SNR and frequency.
%
%  Input Parameters:
%    inputProfile - string variable containing the name of the .mat file that contains
%                   the simulation profile.  The input profile is a .mat file that contains
%                   details of the simulation.  This file can be created with the profile tool
    .
%  Output Parameters:
%    SUCC - 0 if the function completed successfully, otherwise 1.
%  Special Notes:
%    See comments in code.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%-----------  --------  ----------------  ----------------------

function SUCC=rpt_chsnrf(inputProfile)
try
    %assume successful operation
    SUCC=0;

    %get profile
    load(inputProfile);

    %get other inputfiles
    load(CFG_extract_outFile)
    load(CFG_datagen_outFile);

    %get hamming distances at which to plot coherence versus snr versus frequency
    prompt={'Enter Hamming distances (one plot for each Hamming distance):'};
    def={num2str(SIG_HAM)};
    dlgTitle='Coherence vs SNR vs Frequency';
    lineNo=1;
    HAM=[];
    answer={'dummy'};
    while isempty(HAM) & ~isempty(answer)
        answer=inputdlg(prompt,dlgTitle,lineNo,def);
        if ~isempty(answer)
            [HAM,dummy,hamidx]=intersect(str2num(answer{1}),SIG_HAM);
            if isempty(HAM) | length(str2num(answer{1}))>length(HAM)
                waitfor(errordlg({'Hamming distances must be of the following set:',num2str(
                    SIG_HAM)},...
```

```
                    'Incorrect hamming distance','on'));
                def={answer{1}};
                HAM=[];
            end
        end
    end
    if ~isempty(answer)
        %plot results
        for j=1:length(hamidx)
            figure
            surf(CFG_SNR,EXT_COH(1,1).Freq/signs_engunt(EXT_COH(1,1).Freq),[EXT_COH(hamidx(j)
                ,:).Cxy])
            xlabel('SNR (dB)');
            ylabel(['Frequency (',signs_engunt(EXT_COH(1,1).Freq,'prefix'),'Hz)']);
            zlabel('Coherence')
            view(CFG_AZ,CFG_EL);
            title(['Coherence between Received and Ideal Signals (HD=',num2str(SIG_HAM(hamidx(
                j))),')'])
        end
    else
        SUCC=1;
    end
catch
    %log error condition
    signs_log(CFG_logfile,dbstack);
    SUCC=1;
end
return
```

RPT_CMDFRQ

```
%Sign(s) Report: RPT_CMDFRQ
%
%SUCC=RPT_CMDFRQ(inputProfile)
%
%The purpose of this function is to report results of coherence-median difference versus
    frequency.
%
%  Input Parameters:
%   inputProfile - string variable containing the name of the .mat file that contains
%                  the simulation profile.  The input profile is a .mat file that contains
%                  details of the simulation.  This file can be created with the profile tool
    .
%
%  Output Parameters:
%    SUCC - 0 if the function completed successfully, otherwise 1.
%  Special Notes:
%    See comments in code.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%-----------  -------  ------------      --------------------

function SUCC=rpt_cmdfrq(inputProfile)
try
    %assume successful operation
    SUCC=0;

    %get profile
    load(inputProfile);

    %get other inputfiles
    load(CFG_extract_outFile)
    load(CFG_datagen_outFile);

    %get hamming distances and SNRs at which to plot CMD versus frequency
    prompt={'Enter␣SNR␣of␣interest␣(one␣plot␣for␣each␣SNR):',...
            'Enter␣Hamming␣distances␣of␣interest␣to␣appear␣on␣each␣plot:'};
    def={num2str(CFG_SNR),num2str(SIG_HAM)};
    dlgTitle='CMD␣vs␣Frequency';
    lineNo=1;
    SNR=[];
    HD=[];
    answer={'dummy'};
    while (isempty(SNR) | isempty(HD)) & ~isempty(answer)
        answer=inputdlg(prompt,dlgTitle,lineNo,def);
        if ~isempty(answer)
```

```
            [SNR,dummy,snridx]=intersect(str2num(answer{1}),CFG_SNR);
            [HD,dummy,hamidx]=intersect(str2num(answer{2}),SIG_HAM);
            if isempty(SNR) | length(str2num(answer{1}))>length(SNR)
                waitfor(errordlg({'SNR must be of the following set:',num2str(CFG_SNR)},...
                    'Incorrect SNR','on'));
                def={answer{1},answer{2}};
                SNR=[];
            end
            if isempty(HD) | find(HD<min(SIG_HAM) | HD>max(SIG_HAM))
                waitfor(errordlg({['Hamming distances must be within the range of ',num2str(
                    min(CFG_HAM)),...
                            ' and ',num2str(max(SIG_HAM))]},'Incorrect Hamming distances','on'
                                ));
                def={answer{1},answer{2}};
                HD=[];
            end
        end
    end
    if ~isempty(answer)
        %get slice of CMD vs snr vs frequency vs hamming distance surface based on selection
            of hamming distance
        sz=size(EXT_CMDf);
        for i=1:min(length(snridx),sz(2))
            %prepare figure
            Lstring=[];
            figure
            hold on
            for j=1:min(length(hamidx),sz(1))

                %plot result
                plot(EXT_CMDf(hamidx(j),i).Freq,EXT_CMDf(hamidx(j),i).CMD,getnextlinetype(
                    CFG_RPTTYP))

                %prepare legend string
                if isempty(Lstring)
                    Lstring={['HD=',num2str(HD(j))]};
                else
                    Lstring={Lstring{:},['HD=',num2str(HD(j))]};
                end
            end
            xlabel('Frequency (Hz)');
            ylabel('Coherence-Median Difference');
            title(['CMD between Received and Reference Signals (SNR=',num2str(CFG_SNR(snridx(i
                ))),')'])
            grid on
            legend(Lstring,0)
        end
    else
        SUCC=1;
    end
```

```
catch
    %log error condition
    signs_log(CFG_logfile,dbstack);
    SUCC=1;
end
return

%function to return a line type for plots with multiple curves
function y=getnextlinetype(TYP)
persistent count
LineColour='bgrcmykbgrcmykbgrcmykbgrcmykbgrcmykbgrcmyk';
LineSample='ox+*sdvx+*sdvo+*sdvox*sdvox+sdvox+*dvox+*svox+*sd';
if isempty(count)
    count=1;
end
%choose plot type
switch TYP
case 'scatter',
    y=[LineColour(count),LineSample(count)];
case 'scatter-solid',
    y=[LineColour(count),LineSample(count),'-'];
case 'scatter-dotted',
    y=[LineColour(count),LineSample(count),':'];
case 'scatter-dashdot',
    y=[LineColour(count),LineSample(count),'-.'];
case 'scatter-dashed',
    y=[LineColour(count),LineSample(count),'--'];
case 'solid',
    y=[LineColour(count),'-'];
case 'dotted',
    y=[LineColour(count),':'];
case 'dashdot',
    y=[LineColour(count),'-.'];
case 'dashed',
    y=[LineColour(count),'--'];
end
count=mod(count-1,length(LineColour))+2;
return
```

RPT_COHFRQ

```
%Sign(s) Report: RPT_COHFRQ
%
%SUCC=RPT_COHFRQ(inputProfile)
%
%The purpose of this function is to report results of coherence versus frequency.
%
%  Input Parameters:
%   inputProfile - string variable containing the name of the .mat file that contains
%                  the simulation profile.  The input profile is a .mat file that contains
%                  details of the simulation.  This file can be created with the profile tool
    .
%
%  Output Parameters:
%    SUCC - 0 if the function completed successfully, otherwise 1.
%  Special Notes:
%    See comments in code.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%----------  --------  ----------        --------------------

function SUCC=rpt_cohfrq(inputProfile)
try
    %assume successful operation
    SUCC=0;

    %get profile
    load(inputProfile);

    %get other inputfiles
    load(CFG_extract_outFile)
    load(CFG_datagen_outFile);

    %get frequencies and SNRs at which to plot coherence versus Hamming Distance
    prompt={'Enter SNR of interest (one plot for each SNR):',...
            'Enter Hamming distances of interest to appear on each plot:'};
    def={num2str(CFG_SNR),num2str(SIG_HAM)};
    dlgTitle='Coherence vs Frequency';
    lineNo=1;
    SNR=[];
    HD=[];
    answer={'dummy'};
    while (isempty(SNR) | isempty(HD)) & ~isempty(answer)
        answer=inputdlg(prompt,dlgTitle,lineNo,def);
        if ~isempty(answer)
            [SNR,dummy,snridx]=intersect(str2num(answer{1}),CFG_SNR);
```

```
            [HD,dummy,hamidx]=intersect(str2num(answer{2}),SIG_HAM);
            if isempty(SNR) | length(str2num(answer{1}))>length(SNR)
                waitfor(errordlg({'SNR must be of the following set:',num2str(CFG_SNR)},...
                    'Incorrect SNR','on'));
                def={answer{1},answer{2}};
                SNR=[];
            end
            if isempty(HD) | find(HD<min(SIG_HAM) | HD>max(SIG_HAM))
                waitfor(errordlg({['Hamming distances must be within the range of ',num2str(
                    min(SIG_HAM)),...
                            ' and ',num2str(max(SIG_HAM))]},'Incorrect Hamming distances','on'
                                ));
                def={answer{1},answer{2}};
                HD=[];
            end
        end
    end
    if ~isempty(answer)
        %get slice of coherence vs snr vs frequency vs hamming distance surface based on
            selection of hamming distance
        sz=size(EXT_COH);
        for i=1:min(length(snridx),sz(2))
            %prepare figure
            Lstring=[];
            figure
            hold on
            for j=1:min(length(hamidx),sz(1))

                %plot result
                plot(EXT_COH(hamidx(j),i).Freq,EXT_COH(hamidx(j),i).Cxy,getnextlinetype(
                    CFG_RPTTYP))

                %prepare legend string
                if isempty(Lstring)
                    Lstring={['HD=',num2str(HD(j))]};
                else
                    Lstring={Lstring{:},['HD=',num2str(HD(j))]};
                end
            end
            xlabel('Frequency (Hz)');
            ylabel('Coherence');
            title(['Coherence between Received and Reference Signals (SNR=',num2str(CFG_SNR(
                snridx(i))),')'])
            grid on
            legend(Lstring,0)
        end
    else
        SUCC=1;
    end
catch
```

```
    %log error condition
    signs_log(CFG_logfile,dbstack);
    SUCC=1;
end
return

%function to return a line type for plots with multiple curves
function y=getnextlinetype(TYP)
persistent count
LineColour='bgrcmykbgrcmykbgrcmykbgrcmykbgrcmykbgrcmykbgrcmyk';
LineSample='ox+*sdvx+*sdvo+*sdvox*sdvox+sdvox+*dvox+*svox+*sd';
if isempty(count)
    count=1;
end
%choose plot type
switch TYP
case 'scatter',
    y=[LineColour(count),LineSample(count)];
case 'scatter-solid',
    y=[LineColour(count),LineSample(count),'-'];
case 'scatter-dotted',
    y=[LineColour(count),LineSample(count),':'];
case 'scatter-dashdot',
    y=[LineColour(count),LineSample(count),'-.'];
case 'scatter-dashed',
    y=[LineColour(count),LineSample(count),'--'];
case 'solid',
    y=[LineColour(count),'-'];
case 'dotted',
    y=[LineColour(count),':'];
case 'dashdot',
    y=[LineColour(count),'-.'];
case 'dashed',
    y=[LineColour(count),'--'];
end
count=mod(count-1,length(LineColour))+2;
return
```

RPT_COHHAM

```
%Sign(s) Report: RPT_COHHAM
%
%SUCC=RPT_COHHAM(inputProfile)
%
%The purpose of this function is to report results of coherence versus Hamming distance.
%
%  Input Parameters:
%   inputProfile - string variable containing the name of the .mat file that contains
%                  the simulation profile.  The input profile is a .mat file that contains
%                  details of the simulation.  This file can be created with the profile tool
    .
%
%  Output Parameters:
%    SUCC - 0 if the function completed successfully, otherwise 1.
%  Special Notes:
%    See comments in code.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%-----------  --------  ----------------  --------------------

function SUCC=rpt_cohham(inputProfile)
try
    %assume successful operation
    SUCC=0;

    %get profile
    load(inputProfile);

    %get other inputfiles
    load(CFG_extract_outFile)
    load(CFG_datagen_outFile);

    %get frequencies and SNRs at which to plot coherence versus Hamming Distance
    prompt={'Enter SNR of interest (one plot for each SNR):',...
            'Enter frequencies of interest to appear on each plot:'};
    def={num2str(CFG_SNR),''};
    dlgTitle='Coherence vs Hamming Distance';
    lineNo=1;
    SNR=[];
    FV=[];
    answer={'dummy'};
    while (isempty(SNR) | isempty(FV)) & ~isempty(answer)
        answer=inputdlg(prompt,dlgTitle,lineNo,def);
        if ~isempty(answer)
            [SNR,dummy,snridx]=intersect(str2num(answer{1}),CFG_SNR);
```

```
        FV=str2num(answer{2});
        if isempty(SNR) | length(str2num(answer{1}))>length(SNR)
            waitfor(errordlg({'SNR must be of the following set:',num2str(CFG_SNR)},...
                'Incorrect SNR','on'));
            def={answer{1},answer{2}};
            SNR=[];
        end
        if isempty(FV) | find(FV<min(EXT_COH(1,1).Freq) | FV>max(EXT_COH(1,1).Freq))
            waitfor(errordlg({['Frequencies must be within the range of ',num2str(min(
                EXT_COH(1,1).Freq)),...
                        ' Hz and ',num2str(max(EXT_COH(1,1).Freq)),' Hz']},'Incorrect
                            frequencies','on'));
            def={answer{1},answer{2}};
            FV=[];
        end
    end
end
if ~isempty(answer)
    %get slice of coherence vs snr vs frequency vs hamming distance surface based on
        selection of frequencies
    sz=size(EXT_COH);
    for i=1:min(length(snridx),sz(2)) %need to choose minimum in case of a vector of real
        data
        Cxy=[EXT_COH(:,snridx(i)).Cxy];

        %prepare figure
        Lstring=[];
        figure
        hold on
        for j=1:length(FV)
            %get indices for use in interpolation
            idx=max(find(EXT_COH(1,1).Freq<=abs(FV(j))));
            %get adjacent (high and low frequencies for interpolation)
            F=EXT_COH(1,1).Freq(idx:idx+1);
            %get interpolated coherence slice
            C=[Cxy(idx,:);Cxy(idx+1,:)];
            Co=diff(C)/diff(F).*(FV(j)+[F(2),-F(1)]*C./diff(C));

            %plot result
            plot(SIG_HAM,Co,getnextlinetype(CFG_RPTTYP))

            %prepare legend string
            if isempty(Lstring)
                Lstring={[num2str(FV(j)),' Hz']};
            else
                Lstring={Lstring{:},[num2str(FV(j)),' Hz']};
            end
        end
        xlabel('Hamming Distance');
        ylabel('Coherence');
```

```
            title(['Coherence␣between␣Received␣and␣Ideal␣Reference␣Signals␣(SNR=',num2str(
                CFG_SNR(snridx(i))),')'])
            grid on
            legend(Lstring,0)
        end
    else
        SUCC=1;
    end
catch
    %log error condition
    signs_log(CFG_logfile,dbstack);
    SUCC=1;
end
return

%function to return a line type for plots with multiple curves
function y=getnextlinetype(TYP)
persistent count
LineColour='bgrcmykbgrcmykbgrcmykbgrcmykbgrcmykbgrcmykbgrcmyk';
LineSample='ox+*sdvx+*sdvo+*sdvox*sdvox+sdvox+*dvox+*svox+*sd';
if isempty(count)
    count=1;
end
%choose plot type
switch TYP
case 'scatter',
    y=[LineColour(count),LineSample(count)];
case 'scatter-solid',
    y=[LineColour(count),LineSample(count),'-'];
case 'scatter-dotted',
    y=[LineColour(count),LineSample(count),':'];
case 'scatter-dashdot',
    y=[LineColour(count),LineSample(count),'-.'];
case 'scatter-dashed',
    y=[LineColour(count),LineSample(count),'--'];
case 'solid',
    y=[LineColour(count),'-'];
case 'dotted',
    y=[LineColour(count),':'];
case 'dashdot',
    y=[LineColour(count),'-.'];
case 'dashed',
    y=[LineColour(count),'--'];
end
count=mod(count-1,length(LineColour))+2;
return
```

RPT_COHSNR

```
%Sign(s) Report: RPT_COHSNR
%
%SUCC=RPT_COHSNR(inputProfile)
%
%The purpose of this function is to report results of coherence versus SNR.
%
%  Input Parameters:
%   inputProfile - string variable containing the name of the .mat file that contains
%                  the simulation profile.  The input profile is a .mat file that contains
%                  details of the simulation.  This file can be created with the profile tool
    .
%
%  Output Parameters:
%    SUCC - 0 if the function completed successfully, otherwise 1.
%  Special Notes:
%    See comments in code.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%---------   -------   ------------      ---------------------

function SUCC=rpt_cohsnr(inputProfile)
try
    %assume successful operation
    SUCC=0;

    %get profile
    load(inputProfile);

    %get other inputfiles
    load(CFG_extract_outFile)
    load(CFG_datagen_outFile);

    %get frequencies and hamming distances at which to plot coherence versus snr
    prompt={'Enter Hamming distances (one plot for each Hamming distance):',...
            'Enter frequencies (in Hz) of interest to appear on each plot:'};
    def={num2str(SIG_HAM),''};
    dlgTitle='Coherence vs SNR';
    lineNo=1;
    HAM=[];
    FV=[];
    answer={'dummy'};
    while (isempty(HAM) | isempty(FV)) & ~isempty(answer)
        answer=inputdlg(prompt,dlgTitle,lineNo,def);
        if ~isempty(answer)
            [HAM,dummy,hamidx]=intersect(str2num(answer{1}),SIG_HAM);
```

```
            FV=str2num(answer{2});
            if isempty(HAM) | length(str2num(answer{1}))>length(HAM)
                waitfor(errordlg({'Hamming distances must be of the following set:',num2str(
                    SIG_HAM)},...
                    'Incorrect hamming distance','on'));
                def={answer{1},answer{2}};
                HAM=[];
            end
            if isempty(FV) | find(FV<min(EXT_COH(1,1).Freq) | FV>max(EXT_COH(1,1).Freq))
                waitfor(errordlg({['Frequencies must be within the range of ',num2str(min(
                    EXT_COH(1,1).Freq)),...
                            ' Hz and ',num2str(max(EXT_COH(1,1).Freq)),' Hz']},'Incorrect
                                frequencies','on'));
                def={answer{1},answer{2}};
                FV=[];
            end
        end
    end
    if ~isempty(answer)
        %get slice of coherence vs snr vs frequency surface based on selection of frequencies
        for i=1:length(hamidx)
            Cxy=[EXT_COH(hamidx(i),:).Cxy];

            %prepare figure
            Lstring=[];
            figure
            hold on
            for j=1:length(FV)
                %get indices for use in interpolation
                idx=max(find(EXT_COH(1,1).Freq<=abs(FV(j))));
                %get adjacent (high and low frequencies for interpolation)
                F=EXT_COH(1,1).Freq(idx:idx+1);
                %get interpolated coherence slice
                C=[Cxy(idx,:);Cxy(idx+1,:)];
                Co=diff(C)/diff(F).*(FV(j)+[F(2),-F(1)]*C./diff(C));

                %plot result
                plot(CFG_SNR,Co,getnextlinetype(CFG_RPTTYP))

                %prepare legend string
                if isempty(Lstring)
                    Lstring={[num2str(FV(j)),' Hz']};
                else
                    Lstring={Lstring{:},[num2str(FV(j)),' Hz']};
                end
            end
            xlabel('SNR (dB)');
            ylabel('Coherence');
            title(['Coherence between Received and Ideal Signals (HD=',num2str(SIG_HAM(hamidx(
                i))),')'])
```

```
            grid on
            legend(Lstring,0)
        end
    else
        SUCC=1;
    end
catch
    %log error condition
    signs_log(CFG_logfile,dbstack);
    SUCC=1;
end
return

%function to return a line type for plots with multiple curves
function y=getnextlinetype(TYP)
persistent count
LineColour='bgrcmykbgrcmykbgrcmykbgrcmykbgrcmykbgrcmyk';
LineSample='ox+*sdvx+*sdvo+*sdvox*sdvox+sdvox+*dvox+*svox+*sd';
if isempty(count)
    count=1;
end
%choose plot type
switch TYP
case 'scatter',
    y=[LineColour(count),LineSample(count)];
case 'scatter-solid',
    y=[LineColour(count),LineSample(count),'-'];
case 'scatter-dotted',
    y=[LineColour(count),LineSample(count),':'];
case 'scatter-dashdot',
    y=[LineColour(count),LineSample(count),'-.'];
case 'scatter-dashed',
    y=[LineColour(count),LineSample(count),'--'];
case 'solid',
    y=[LineColour(count),'-'];
case 'dotted',
    y=[LineColour(count),':'];
case 'dashdot',
    y=[LineColour(count),'-.'];
case 'dashed',
    y=[LineColour(count),'--'];
end
count=mod(count-1,length(LineColour))+2;
return
```

RPT_ENSAMB

```
%Sign(s) Report: RPT_ENSAMB
%
%SUCC=RPT_ENSAMB(inputProfile)
%
%The purpose of this function is to report results of Benedetto's entropy versus appending
    length
%versus number of samples
%
%  Input Parameters:
%    inputProfile - string variable containing the name of the .mat file that contains
%                   the simulation profile.  The input profile is a .mat file that contains
%                   details of the simulation.  This file can be created with the profile tool
    .
%  Output Parameters:
%    SUCC - 0 if the function completed successfully, otherwise 1.
%  Special Notes:
%    See comments in code.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%----------  --------  ----------        --------------------

function SUCC=rpt_ensamb(inputProfile)
try
    %assume successful operation
    SUCC=0;

    %get profile
    load(inputProfile);

    %get other inputfiles
    load(CFG_extract_outFile)
    load(CFG_datagen_outFile);

    %get Hamming Distances and SNRs at which to plot Benedetto's entropy versus
    %appending sequence length and number of samples of signal vector
    prompt={'Enter␣SNR␣of␣interest:',...
            'Enter␣Hamming␣distances␣of␣interest:'};
    def={num2str(CFG_SNR),num2str(SIG_HAM)};
    dlgTitle='Benedetto''s␣Entropy␣vs␣Appended␣Sequence␣Length␣vs␣Number␣of␣Signal␣Samples';
    lineNo=1;
    SNR=[];
    HD=[];
    answer={'dummy'};
    while (isempty(SNR) | isempty(HD)) & ~isempty(answer)
        answer=inputdlg(prompt,dlgTitle,lineNo,def);
```

```
        if ~isempty(answer)
            [SNR,dummy,snridx]=intersect(str2num(answer{1}),CFG_SNR);
            [HD,dummy,hamidx]=intersect(str2num(answer{2}),SIG_HAM);
            if isempty(SNR) | length(str2num(answer{1}))>length(SNR)
                waitfor(errordlg({'SNR must be of the following set:',num2str(CFG_SNR)},...
                    'Incorrect SNR','on'));
                def={answer{1},answer{2}};
                SNR=[];
            end
            if isempty(HD) | find(HD<min(SIG_HAM) | HD>max(SIG_HAM))
                waitfor(errordlg({['Hamming distances must be within the range of ',num2str(
                    min(SIG_HAM)),...
                            ' and ',num2str(max(SIG_HAM))]},'Incorrect Hamming distances','on'
                                ));
                def={answer{1},answer{2}};
                HD=[];
            end
        end
    end
    if ~isempty(answer)
        %get slice of entropic distance vs snr vs hamming distance surface based on selection
            of hamming distance
        sz=size(EXT_ENT);
        for i=1:min(length(hamidx),sz(1))
            for j=1:min(length(snridx),sz(2))

                %plot entropic distance
                figure
                surf(EXT_ENT(hamidx(i),snridx(j)).Nmsg,EXT_ENT(hamidx(i),snridx(j)).Lapp,
                    EXT_ENT(hamidx(i),snridx(j)).S)
                view(CFG_AZ,CFG_EL);
                grid on
                xlabel('Number of Samples');
                ylabel('Length of Appended Sequence')
                zlabel('Entropic Distance')
                title(['Entropic Distance by Benedetto''s Method for Received & Reference
                    Signals (SNR=',num2str(CFG_SNR(snridx(j))),...
                        ';HD=',num2str(SIG_HAM(hamidx(i))),')'])

                %plot relative entropy of A wrt B
                figure
                surf(EXT_ENT(hamidx(i),snridx(j)).Nmsg,EXT_ENT(hamidx(i),snridx(j)).Lapp,
                    EXT_ENT(hamidx(i),snridx(j)).S_AB)
                view(CFG_AZ,CFG_EL);
                grid on
                xlabel('Number of Samples');
                ylabel('Length of Appended Sequence')
                zlabel('Relative Entropy')
                title(['Relative Entropy of Received wrt Reference by Benedetto''s Method (SNR
                    =',num2str(CFG_SNR(snridx(j))),...
```

```
                    ';HD=',num2str(SIG_HAM(hamidx(i))),')']);
            end
        end
    else
        SUCC=1;
    end
catch
    %log error condition
    signs_log(CFG_logfile,dbstack);
    SUCC=1;
end
return
```

RPT_ENTSAM

```
%Sign(s) Report: RPT_ENTSAM
%
%SUCC=RPT_ENTSAM(inputProfile)
%
%The purpose of this function is to report results of Benedetto's entropy versus number of
    samples
%
%  Input Parameters:
%    inputProfile - string variable containing the name of the .mat file that contains
%                   the simulation profile.  The input profile is a .mat file that contains
%                   details of the simulation.  This file can be created with the profile tool
    .
%  Output Parameters:
%    SUCC - 0 if the function completed successfully, otherwise 1.
%  Special Notes:
%    See comments in code.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor          Changes Made
%---------   -------   ------------    ---------------------
%31/03/05    1.01      JEG             Added plots of relative entropic distance

function SUCC=rpt_entsam(inputProfile)
try
    %assume successful operation
    SUCC=0;

    %get profile
    load(inputProfile);

    %get other inputfiles
    load(CFG_extract_outFile)
    load(CFG_datagen_outFile);

    %get Hamming Distances and SNRs at which to plot Benedetto's entropy versus
    %appending sequence length and number of samples of signal vector
    prompt={'Enter␣SNR␣of␣interest:',...
            'Enter␣Hamming␣distances␣of␣interest:',...
            'Enter␣Appended␣Sequence␣lengths␣of␣interest:'};
    def={num2str(CFG_SNR),num2str(SIG_HAM),num2str(EXT_ENT(1,1).Lapp)};
    dlgTitle='Benedetto''s␣Entropy␣vs␣Appended␣Sequence␣Length␣vs␣Number␣of␣Signal␣Samples';
    lineNo=1;
    SNR=[];
    HD=[];
    B=[];
    answer={'dummy'};
```

```
while (isempty(SNR) | isempty(HD) | isempty(B)) & ~isempty(answer)
    answer=inputdlg(prompt,dlgTitle,lineNo,def);
    if ~isempty(answer)
        [SNR,dummy,snridx]=intersect(str2num(answer{1}),CFG_SNR);
        [HD,dummy,hamidx]=intersect(str2num(answer{2}),SIG_HAM);
        [B,dummy,bidx]=intersect(str2num(answer{3}),EXT_ENT(1,1).Lapp);
        if isempty(SNR) | length(str2num(answer{1}))>length(SNR)
            waitfor(errordlg({'SNR must be of the following set:',num2str(CFG_SNR)},...
                'Incorrect SNR','on'));
            def={answer{1},answer{2},answer{3}};
            SNR=[];
        end
        if isempty(HD) | find(HD<min(SIG_HAM) | HD>max(SIG_HAM))
            waitfor(errordlg({['Hamming distance must be of the following set:',num2str(
                SIG_HAM)]},...
                'Incorrect Hamming Distance','on'));
            def={answer{1},answer{2},answer{3}};
            HD=[];
        end
        if isempty(B) | find(B<min(EXT_ENT(1,1).Lapp) | B>max(EXT_ENT(1,1).Lapp))
            waitfor(errordlg({['Appended Sequence length must be of the following set:'
                ,...
                    num2str(EXT_ENT(1,1).Lapp)]},'Incorrect Appended Sequence length',
                        'on'));
            def={answer{1},answer{2},answer{3}};
            B=[];
        end
    end
end
if ~isempty(answer)

    %prepare notes for plots
    if CFG_REALRX
        RXNote=['Rx Source: ',strrep(CFG_realFileRX(max(findstr(CFG_realFileRX,'\'))+1:end
            ),'_','\_')];
    else
        RXNote=['Rx Source: synthetic'];
    end
    if CFG_REALREF
        REFNote=['Ref Source: ',strrep(CFG_realFileREF(max(findstr(CFG_realFileREF,'\'))
            +1:end),'_','\_')];
    else
        REFNote=['Ref Source: synthetic'];
    end

    %get slice of entropy vs snr vs hamming distance for various appended lengths
    sz=size(EXT_ENT);
    for i=1:min(length(hamidx),sz(1))
        for j=1:min(length(snridx),sz(2))
```

```
%plot entropic distance versus samples of received signal
figure
semilogx(EXT_ENT(hamidx(i),snridx(j)).Nmsg,EXT_ENT(hamidx(i),snridx(j)).S(bidx
    (1),:),getnextlinetype(CFG_RPTTYP))
hold on
for k=2:length(bidx)
    semilogx(EXT_ENT(hamidx(i),snridx(j)).Nmsg,EXT_ENT(hamidx(i),snridx(j)).S(
        bidx(k),:),getnextlinetype(CFG_RPTTYP))
end
grid on
xlabel('Number␣of␣Samples␣in␣Received␣Signal␣Vector');
ylabel('Entropic␣Distance')
title(['Entropic␣Distance␣between␣Received␣&␣Reference␣Signals␣by␣Benedetto''s
    ␣Method␣(SNR=',num2str(CFG_SNR(snridx(j))),...
        ';HD=',num2str(SIG_HAM(hamidx(i))),')'])
V=axis;
Ymax=V(4);
Ymin=V(3);
Xmax=V(2);
Xmin=V(1);
X1=1.05*Xmin;
X2=X1;
Y1=.95*(Ymax-Ymin)+Ymin;
Y2=.90*(Ymax-Ymin)+Ymin;
text([X1;X2],[Y1;Y2],{RXNote;REFNote})

%prepare legend
Lstring={['|b|=',num2str(EXT_ENT(hamidx(i),snridx(j)).Lapp(bidx(1)))]};
for k=2:length(bidx)
    Lstring={Lstring{:},['|b|=',num2str(EXT_ENT(hamidx(i),snridx(j)).Lapp(bidx
        (k)))]};
end
legend(Lstring,0)

%compute relative entropic distances if not already computed
if exist('R_AB')~=1 & exist('R_BA')~=1
    R_AB=EXT_ENT(hamidx(i),snridx(j)).S_AB./EXT_ENT(hamidx(i),snridx(j)).S_BB;
    R_BA=EXT_ENT(hamidx(i),snridx(j)).S_BA./EXT_ENT(hamidx(i),snridx(j)).S_AA;
end

%plot relative entropic distances
figure
semilogx(EXT_ENT(hamidx(i),snridx(j)).Nmsg,R_AB(bidx(1),:),getnextlinetype(
    CFG_RPTTYP))
hold on
semilogx(EXT_ENT(hamidx(i),snridx(j)).Nmsg,R_BA(bidx(1),:),getnextlinetype(
    CFG_RPTTYP))
for k=2:length(bidx)
    semilogx(EXT_ENT(hamidx(i),snridx(j)).Nmsg,R_AB(bidx(k),:),getnextlinetype
        (CFG_RPTTYP))
```

```
    semilogx(EXT_ENT(hamidx(i),snridx(j)).Nmsg,R_BA(bidx(k),:),getnextlinetype
        (CFG_RPTTYP))
end
grid on
xlabel('Number of Samples in Received Signal Vector');
ylabel('Relative Entropic Distances')
title(['Relative Entropic Distances of Received wrt Reference by Benedetto''s
    Method (SNR=',num2str(CFG_SNR(snridx(j))),...
        ';HD=',num2str(SIG_HAM(hamidx(i))),')'])
V=axis;
Ymax=V(4);
Ymin=V(3);
Xmax=V(2);
Xmin=V(1);
X1=1.05*Xmin;
X2=X1;
Y1=.95*(Ymax-Ymin)+Ymin;
Y2=.90*(Ymax-Ymin)+Ymin;
text([X1;X2],[Y1;Y2],{RXNote;REFNote})

%prepare legend
Lstring={['R_A_B:|b|=',num2str(EXT_ENT(hamidx(i),snridx(j)).Lapp(bidx(1)))
    ],...
    ['R_B_A:|b|=',num2str(EXT_ENT(hamidx(i),snridx(j)).Lapp(bidx(1)))]};
for k=2:length(bidx)
    Lstring={Lstring{:},['R_A_B:|b|=',num2str(EXT_ENT(hamidx(i),snridx(j)).
        Lapp(bidx(k)))],...
    ['R_B_A:|b|=',num2str(EXT_ENT(hamidx(i),snridx(j)).Lapp(bidx(k)))]};
end
legend(Lstring,0)

%plot relative entropy of received signal versus reference
figure
semilogx(EXT_ENT(hamidx(i),snridx(j)).Nmsg,EXT_ENT(hamidx(i),snridx(j)).S_AB(
    bidx(1),:),getnextlinetype(CFG_RPTTYP))
hold on
for k=2:length(bidx)
    semilogx(EXT_ENT(hamidx(i),snridx(j)).Nmsg,EXT_ENT(hamidx(i),snridx(j)).
        S_AB(bidx(k),:),getnextlinetype(CFG_RPTTYP))
end
grid on
xlabel('Number of Samples');
ylabel('Relative Entropy')
title(['Relative Entropy of Received wrt Reference (S_A_B) by Benedetto''s
    Method (SNR=',num2str(CFG_SNR(snridx(j))),...
        ';HD=',num2str(SIG_HAM(hamidx(i))),')'])
V=axis;
Ymax=V(4);
Ymin=V(3);
Xmax=V(2);
```

```
                Xmin=V(1);
                X1=1.05*Xmin;
                X2=X1;
                Y1=.95*(Ymax-Ymin)+Ymin;
                Y2=.90*(Ymax-Ymin)+Ymin;
                text([X1;X2],[Y1;Y2],{RXNote;REFNote})

                %prepare legend
                Lstring={['|b|=',num2str(EXT_ENT(hamidx(i),snridx(j)).Lapp(bidx(1)))]};
                for k=2:length(bidx)
                    Lstring={Lstring{:},['|b|=',num2str(EXT_ENT(hamidx(i),snridx(j)).Lapp(bidx
                        (k)))]};
                end
                legend(Lstring,0)

            end
        end
    else
        SUCC=1;
    end
catch
    %log error condition
    signs_log(CFG_logfile,dbstack);
    SUCC=1;
end
return

%function to return a line type for plots with multiple curves
function y=getnextlinetype(TYP)
persistent count
LineColour='bgrcmykbgrcmykbgrcmykbgrcmykbgrcmykbgrcmykbgrcmyk';
LineSample='ox+*sdvx+*sdvo+*sdvox*sdvox+sdvox+*dvox+*svox+*sd';
if isempty(count) | count>length(LineColour)
    count=1;
end
%choose plot type
switch TYP
case 'scatter',
    y=[LineColour(count),LineSample(count)];
case 'scatter-solid',
    y=[LineColour(count),LineSample(count),'-'];
case 'scatter-dotted',
    y=[LineColour(count),LineSample(count),':'];
case 'scatter-dashdot',
    y=[LineColour(count),LineSample(count),'-.'];
case 'scatter-dashed',
    y=[LineColour(count),LineSample(count),'--'];
case 'solid',
    y=[LineColour(count),'-'];
case 'dotted',
```

```
    y=[LineColour(count),':'];
case 'dashdot',
    y=[LineColour(count),'-.'];
case 'dashed',
    y=[LineColour(count),'--'];
end
count=mod(count-1,length(LineColour))+2;
return
```

RPT_GCMDFR

```
%Sign(s) Report: RPT_GCMDFR
%
%SUCC=RPT_GCMDFR(inputProfile)
%
%The purpose of this function is to report results of generalized coherence-median difference
%versus frequency.
%
%  Input Parameters:
%   inputProfile - string variable containing the name of the .mat file that contains
%                  the simulation profile.  The input profile is a .mat file that contains
%                  details of the simulation.  This file can be created with the profile tool
    .
%
%  Output Parameters:
%    SUCC - 0 if the function completed successfully, otherwise 1.
%  Special Notes:
%    See comments in code.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%----------  --------  ----------        --------------------

function SUCC=rpt_gcmdfr(inputProfile)
try
    %assume successful operation
    SUCC=0;

    %get profile
    load(inputProfile);

    %get other inputfiles
    load(CFG_extract_outFile)
    load(CFG_datagen_outFile);

    %get hamming distances and SNRs at which to plot CMD versus frequency
    prompt={'Enter SNR of interest (one plot for each SNR):',...
            'Enter Hamming distances of interest to appear on each plot:'};
    def={num2str(CFG_SNR),num2str(SIG_HAM)};
    dlgTitle='Generalized CMD vs Frequency';
    lineNo=1;
    SNR=[];
    HD=[];
    answer={'dummy'};
    while (isempty(SNR) | isempty(HD)) & ~isempty(answer)
        answer=inputdlg(prompt,dlgTitle,lineNo,def);
        if ~isempty(answer)
```

```
        [SNR,dummy,snridx]=intersect(str2num(answer{1}),CFG_SNR);
        [HD,dummy,hamidx]=intersect(str2num(answer{2}),SIG_HAM);
        if isempty(SNR) | length(str2num(answer{1}))>length(SNR)
            waitfor(errordlg({'SNR must be of the following set:',num2str(CFG_SNR)},...
                'Incorrect SNR','on'));
            def={answer{1},answer{2}};
            SNR=[];
        end
        if isempty(HD) | find(HD<min(SIG_HAM) | HD>max(SIG_HAM))
            waitfor(errordlg({['Hamming distances must be within the range of ',num2str(
                min(CFG_HAM)),...
                    ' and ',num2str(max(SIG_HAM))]},'Incorrect Hamming distances','on'
                        ));
            def={answer{1},answer{2}};
            HD=[];
        end
    end
end
if ~isempty(answer)
    %get slice of CMD vs snr vs frequency vs hamming distance surface based on selection
        of hamming distance
    sz=size(EXT_CMDf);
    for i=1:min(length(snridx),sz(2))
        %prepare figure
        Lstring=[];
        figure
        hold on
        for j=1:min(length(hamidx),sz(1))

            %plot result
            plot(EXT_CMDf(hamidx(j),i).Freq,EXT_CMDf(hamidx(j),i).gCMD,getnextlinetype(
                CFG_RPTTYP))

            %prepare legend string
            if isempty(Lstring)
                Lstring={['HD=',num2str(HD(j))]};
            else
                Lstring={Lstring{:},['HD=',num2str(HD(j))]};
            end
        end
        xlabel('Frequency (Hz)');
        ylabel('Generalized Coherence-Median Difference');
        title(['Generalized CMD between Received and Reference Signals (SNR=',num2str(
            CFG_SNR(snridx(i))),')'])
        grid on
        legend(Lstring,0)
    end
else
    SUCC=1;
end
```

```
catch
    %log error condition
    signs_log(CFG_logfile,dbstack);
    SUCC=1;
end
return

%function to return a line type for plots with multiple curves
function y=getnextlinetype(TYP)
persistent count
LineColour='bgrcmykbgrcmykbgrcmykbgrcmykbgrcmykbgrcmykbgrcmyk';
LineSample='ox+*sdvx+*sdvo+*sdvox*sdvox+sdvox+*dvox+*svox+*sd';
if isempty(count)
    count=1;
end
%choose plot type
switch TYP
case 'scatter',
    y=[LineColour(count),LineSample(count)];
case 'scatter-solid',
    y=[LineColour(count),LineSample(count),'-'];
case 'scatter-dotted',
    y=[LineColour(count),LineSample(count),':'];
case 'scatter-dashdot',
    y=[LineColour(count),LineSample(count),'-.'];
case 'scatter-dashed',
    y=[LineColour(count),LineSample(count),'--'];
case 'solid',
    y=[LineColour(count),'-'];
case 'dotted',
    y=[LineColour(count),':'];
case 'dashdot',
    y=[LineColour(count),'-.'];
case 'dashed',
    y=[LineColour(count),'--'];
end
count=mod(count-1,length(LineColour))+2;
return
```

RPT_MEDSAM

```
%Sign(s) Report: RPT_MEDSAM
%
%SUCC=RPT_MEDSAM(inputProfile)
%
%The purpose of this function is to report results of mean entropic distance (via Benedetto's
%method) versus number of samples
%
%  Input Parameters:
%    inputProfile - string variable containing the name of the .mat file that contains
%                   the simulation profile.  The input profile is a .mat file that contains
%                   details of the simulation.  This file can be created with the profile tool
    .
%  Output Parameters:
%    SUCC - 0 if the function completed successfully, otherwise 1.
%  Special Notes:
%    See comments in code.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%----------  --------  ----------        --------------------

function SUCC=rpt_medsam(inputProfile)
try
    %assume successful operation
    SUCC=0;

    %get profile
    load(inputProfile);

    %get other inputfiles
    load(CFG_extract_outFile)
    load(CFG_datagen_outFile);

    %get Hamming Distances and SNRs at which to plot Benedetto's entropy versus
    %appending sequence length and number of samples of signal vector
    prompt={'Enter␣SNR␣of␣interest:',...
            'Enter␣Hamming␣distances␣of␣interest:'};
    def={num2str(CFG_SNR),num2str(SIG_HAM)};
    dlgTitle='Benedetto''s␣Entropic␣Distance␣vs␣Number␣of␣Samples';
    lineNo=1;
    SNR=[];
    HD=[];
    answer={'dummy'};
    while (isempty(SNR) | isempty(HD)) & ~isempty(answer)
        answer=inputdlg(prompt,dlgTitle,lineNo,def);
        if ~isempty(answer)
```

```
            [SNR,dummy,snridx]=intersect(str2num(answer{1}),CFG_SNR);
            [HD,dummy,hamidx]=intersect(str2num(answer{2}),SIG_HAM);
            if isempty(SNR) | length(str2num(answer{1}))>length(SNR)
                waitfor(errordlg({'SNR must be of the following set:',num2str(CFG_SNR)},...
                    'Incorrect SNR','on'));
                def={answer{1},answer{2},answer{3}};
                SNR=[];
            end
            if isempty(HD) | find(HD<min(SIG_HAM) | HD>max(SIG_HAM))
                waitfor(errordlg({['Hamming distance must be of the following set:',num2str(
                    SIG_HAM)]},...
                    'Incorrect Hamming Distance','on'));
                def={answer{1},answer{2},answer{3}};
                HD=[];
            end
        end
    end
    if ~isempty(answer)

        %prepare notes for plots
        if CFG_REALRX
            RXNote=['Rx Source: ',strrep(CFG_realFileRX(max(findstr(CFG_realFileRX,'\'))+1:end
                ),'_','\_')];
        else
            if exist('CFG_LVL')==1
                RXNote=['Rx Source: synthetic ',CFG_MODL,'(',num2str(CFG_LVL),' level ; ',
                    num2str(CFG_R),' bps)'];
            else
                RXNote=['Rx Source: synthetic ',CFG_MODL,'(',num2str(CFG_R),' bps)'];
            end
        end
        if CFG_REALREF
            REFNote=['Ref Source: ',strrep(CFG_realFileREF(max(findstr(CFG_realFileREF,'\'))
                +1:end),'_','\_')];
        else
            if exist('CFG_LVL')==1
                REFNote=['Rx Source: synthetic ',CFG_MODL,'(',num2str(CFG_LVL),' level ; ',
                    num2str(CFG_R),' bps)'];
            else
                REFNote=['Rx Source: synthetic ',CFG_MODL,'(',num2str(CFG_R),' bps)'];
            end
        end

        %get slice of entropy vs snr vs hamming distance for various appended lengths
        sz=size(EXT_ENT);
        for i=1:min(length(hamidx),sz(1))
            for j=1:min(length(snridx),sz(2))

                %plot mean entropic distance vs appending sequence
                figure
```

```
S=EXT_ENT(hamidx(i),snridx(j)).S;
Su=mean(S'); %get mean entropic distance for each different appending sequence
     length
Sq=std(S'); %get standard deviation for each different appending sequence
    length
errorbar(EXT_ENT(hamidx(i),snridx(j)).Lapp,Su,Sq);
grid on
xlabel('Length␣of␣Appended␣Sequence␣(samples)')
ylabel('Mean␣Entropic␣Distance,␣S_\mu␣(error␣bars␣show␣␣␣\sigma)')
title('Mean␣Entropic␣Distance␣versus␣Length␣of␣Appended␣Sequence')
axis([min(EXT_ENT(hamidx(i),snridx(j)).Lapp) max(EXT_ENT(hamidx(i),snridx(j)).
    Lapp) 0 max(1,2*max(Su))])
V=axis;
Ymax=V(4);
Ymin=V(3);
Xmax=V(2);
Xmin=V(1);
X1=1.05*Xmin;
X2=X1;
Y1=.95*(Ymax-Ymin)+Ymin;
Y2=.90*(Ymax-Ymin)+Ymin;
text([X1;X2],[Y1;Y2],{RXNote;REFNote})

%compute stats for mean entropic distance versus appended sequence length
meanSu=mean(Su);
stdSu=std(Su);
Y3=.80*(Ymax-Ymin)+Ymin;
Y4=.70*(Ymax-Ymin)+Ymin;
text([X1;X2],[Y3;Y4],{['E[{S_\mu}]␣=␣',num2str(meanSu)];['\sigma_{S_\mu}␣=␣',
    num2str(stdSu)]});

%plot mean entropic distance vs number of samples in received signal vector
figure
S=EXT_ENT(hamidx(i),snridx(j)).S;
Su=mean(S); %get mean entropic distance for each different appending sequence
    length
Sq=std(S); %get standard deviation for each different appending sequence
    length
errorbar(EXT_ENT(hamidx(i),snridx(j)).Nmsg,Su,Sq);
grid on
xlabel('Number␣of␣Samples␣in␣Received␣Signal␣Vector');
ylabel('Mean␣Entropic␣Distance,␣S_\mu␣(error␣bars␣show␣␣␣\sigma)')
title('Mean␣Entropic␣Distance␣vs␣Length␣of␣Received␣Signal␣Vector')
axis([min(EXT_ENT(hamidx(i),snridx(j)).Nmsg) max(EXT_ENT(hamidx(i),snridx(j)).
    Nmsg) 0 max(1,2*max(Su))])
V=axis;
Ymax=V(4);
Ymin=V(3);
Xmax=V(2);
Xmin=V(1);
```

```
X1=1.05*Xmin;
X2=X1;
Y1=.95*(Ymax-Ymin)+Ymin;
Y2=.90*(Ymax-Ymin)+Ymin;
text([X1;X2],[Y1;Y2],{RXNote;REFNote})

%compute stats for mean entropic distance versus number of samples
meanSu=mean(Su);
stdSu=std(Su);
Y3=.80*(Ymax-Ymin)+Ymin;
Y4=.70*(Ymax-Ymin)+Ymin;
text([X1;X2],[Y3;Y4],{['E[{S_\mu}] = ',num2str(meanSu)];['\sigma_{S_\mu} = ',
    num2str(stdSu)]});

%compute relative entropic distances if not already computed
if exist('R_AB')~=1 & exist('R_BA')~=1
    R_AB=EXT_ENT(hamidx(i),snridx(j)).S_AB./EXT_ENT(hamidx(i),snridx(j)).S_BB;
    R_BA=EXT_ENT(hamidx(i),snridx(j)).S_BA./EXT_ENT(hamidx(i),snridx(j)).S_AA;
end

%plot mean relative entropic distance vs appending sequence
figure
R_ABu=mean(R_AB'); %get mean relative entropic distance for each different
    appending sequence length
R_ABq=std(R_AB'); %get standard deviation for each different appending
    sequence length
R_BAu=mean(R_BA'); %get mean relative entropic distance for each different
    appending sequence length
R_BAq=std(R_BA'); %get standard deviation for each different appending
    sequence length
errorbar(EXT_ENT(hamidx(i),snridx(j)).Lapp,R_ABu,R_ABq,'bo-');
hold on
errorbar(EXT_ENT(hamidx(i),snridx(j)).Lapp,R_BAu,R_BAq,'k*-');
legend('R_A_B \sigma Bars','Mean R_A_B','R_B_A \sigma Bars','Mean R_B_A',0)
grid on
xlabel('Length of Appended Sequence (samples)')
ylabel('Mean Relative Entropic Distance (error bars show \sigma)')
title('Mean Relative Entropic Distance versus Length of Appended Sequence')
axis([min(EXT_ENT(hamidx(i),snridx(j)).Lapp) max(EXT_ENT(hamidx(i),snridx(j)).
    Lapp) 0 max(1,2*max([R_ABu,R_BAu]))])
hold off
V=axis;
Ymax=V(4);
Ymin=V(3);
Xmax=V(2);
Xmin=V(1);
X1=1.05*Xmin;
X2=X1;
Y1=.95*(Ymax-Ymin)+Ymin;
Y2=.90*(Ymax-Ymin)+Ymin;
```

```
                text([X1;X2],[Y1;Y2],{RXNote;REFNote})

                %plot mean relative entropic distance vs number of samples in received signal
                    vector
                figure
                R_ABu=mean(R_AB); %get mean relative entropic distance for each different
                    vector length
                R_ABq=std(R_AB); %get standard deviation for each different appending vector
                    length
                R_BAu=mean(R_BA); %get mean relative entropic distance for each different
                    vector length
                R_BAq=std(R_BA); %get standard deviation for each different appending vector
                    length
                errorbar(EXT_ENT(hamidx(i),snridx(j)).Nmsg,R_ABu,R_ABq,'bo-');
                hold on
                errorbar(EXT_ENT(hamidx(i),snridx(j)).Nmsg,R_BAu,R_BAq,'k*-');
                legend('R_A_B \sigma Bars','Mean R_A_B','R_B_A \sigma Bars','Mean R_B_A',0)
                grid on
                xlabel('Number of Samples in Received Signal Vector');
                ylabel('Mean Relative Entropic Distance (error bars show \sigma)')
                title('Mean Relative Entropic Distance vs Length of Received Signal Vector')
                axis([min(EXT_ENT(hamidx(i),snridx(j)).Nmsg) max(EXT_ENT(hamidx(i),snridx(j)).
                    Nmsg) 0 max(1,2*max([R_ABu,R_BAu]))])
                hold off
                V=axis;
                Ymax=V(4);
                Ymin=V(3);
                Xmax=V(2);
                Xmin=V(1);
                X1=1.05*Xmin;
                X2=X1;
                Y1=.95*(Ymax-Ymin)+Ymin;
                Y2=.90*(Ymax-Ymin)+Ymin;
                text([X1;X2],[Y1;Y2],{RXNote;REFNote})

            end
        end
    else
        SUCC=1;
    end
catch
    %log error condition
    signs_log(CFG_logfile,dbstack);
    SUCC=1;
end
return

%function to return a line type for plots with multiple curves
function y=getnextlinetype(TYP)
persistent count
```

```
LineColour='bgrcmykbgrcmykbgrcmykbgrcmykbgrcmykbgrcmyk';
LineSample='ox+*sdvx+*sdvo+*sdvox*sdvox+sdvox+*dvox+*svox+*sd';
if isempty(count) | count>length(LineColour)
    count=1;
end
%choose plot type
switch TYP
case 'scatter',
    y=[LineColour(count),LineSample(count)];
case 'scatter-solid',
    y=[LineColour(count),LineSample(count),'-'];
case 'scatter-dotted',
    y=[LineColour(count),LineSample(count),':'];
case 'scatter-dashdot',
    y=[LineColour(count),LineSample(count),'-.'];
case 'scatter-dashed',
    y=[LineColour(count),LineSample(count),'--'];
case 'solid',
    y=[LineColour(count),'-'];
case 'dotted',
    y=[LineColour(count),':'];
case 'dashdot',
    y=[LineColour(count),'-.'];
case 'dashed',
    y=[LineColour(count),'--'];
end
count=mod(count-1,length(LineColour))+2;
return
```

RPT_PSDS

```
%Sign(s) Report: RPT_PSDS
%
%SUCC=RPT_PSDS(inputProfile)
%
%The purpose of this function is to report the power-spectral densities of signals
%along the simulation path.
%
%  Input Parameters:
%   inputProfile - string variable containing the name of the .mat file that contains
%                  the simulation profile.  The input profile is a .mat file that contains
%                  details of the simulation.  This file can be created with the profile tool
    .
%  Output Parameters:
%    SUCC - 0 if the function completed successfully, otherwise 1.
%  Special Notes:
%    See comments in code.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor          Changes Made
%----------  --------  ------------    --------------------
%13/02/08    1.01      JEG             Fixed syntax error in SIG_TxIF/RF
%                                      scenario

function SUCC=rpt_psds(inputProfile)
try
    %assume successful operation
    SUCC=0;

    %get profile
    load(inputProfile);

    %get other inputfiles
    load(CFG_datagen_outFile);
    load(CFG_txsim_outFile);
    load(CFG_xswitch_outFile);
    load(CFG_extract_outFile)

    %get signals to compute and/or plot already extracted PSDs
    [SigDes,Sigs]=textread('signs_sigtypes.dat','%s%s','delimiter',',');
    if CFG_REALRX
        SigDes{length(SigDes)+1}='Real Received Baseband (all data points)';
        Sigs{length(Sigs)+1}='SIG_realRxBS';
    end
    if CFG_REALREF
        SigDes{length(SigDes)+1}='Real Reference Baseband (all data points)';
        Sigs{length(Sigs)+1}='SIG_realREF';
```

```
end
if exist('EXT_PSD')
    SigDes{length(SigDes)+1}='Extracted␣PSD␣of␣Rx␣Baseband';
    Sigs{length(Sigs)+1}='EXT_PSD';
end
[SELECTION,OK]=listdlg('ListString',SigDes,'SelectionMode','multiple','ListSize'
    ,[460,400],...
    'Name','Signals','PromptString','Select␣one␣or␣more␣signals␣to␣plot␣PSDs:');

%if a valid selection then execute user instructions
if OK
    %get the signals to plot
    count=0;
    for rptidx=1:length(SELECTION)
        %prepare counters, size measures, and strings
        switch Sigs{SELECTION(rptidx)}
        case {'SIG_TxBS'},
            load(CFG_txsim_outFile,Sigs{SELECTION(rptidx)}); %get variable
            count=count+1;
            SIGNAL{count}=[Sigs{SELECTION(rptidx)},'{k}'];
            SZ(count,1:2)=size(eval(Sigs{SELECTION(rptidx)})); %array to keep track of
                array size of signals being plotted
            sigidx(count)=rptidx; %array to keep track of which signals can be plotted
        case {'SIG_REF'},
            load(CFG_xswitch_outFile,Sigs{SELECTION(rptidx)}); %get variable
            count=count+1;
            SIGNAL{count}=[Sigs{SELECTION(rptidx)},'{k}'];
            SZ(count,1:2)=size(eval(Sigs{SELECTION(rptidx)})); %array to keep track of
                array size of signals being plotted
            sigidx(count)=rptidx; %array to keep track of which signals can be plotted
        case {'SIG_realRxBS','SIG_realREF'},
            load(CFG_xswitch_outFile,Sigs{SELECTION(rptidx)}); %get variable
            count=count+1;
            SIGNAL{count}=[Sigs{SELECTION(rptidx)},'{k}'];
            SZ(count,1:2)=size(eval(Sigs{SELECTION(rptidx)})); %array to keep track of
                array size of signals being plotted
            sigidx(count)=rptidx; %array to keep track of which signals can be plotted
        case {'SIG_RxBS'},
            load(CFG_xswitch_outFile,Sigs{SELECTION(rptidx)}); %get variable
            count=count+1;
            SIGNAL{count}=[Sigs{SELECTION(rptidx)},'{k,m}'];
            SZ(count,1:2)=size(eval(Sigs{SELECTION(rptidx)})); %array to keep track of
                array size of signals being plotted
            sigidx(count)=rptidx; %array to keep track of which signals can be plotted
        case {'EXT_PSD'},
            load(CFG_extract_outFile,Sigs{SELECTION(rptidx)}); %get variable
            count=count+1;
            SIGNAL{count}=[Sigs{SELECTION(rptidx)},'{k,m}'];
            SZ(count,1:2)=size(eval(Sigs{SELECTION(rptidx)})); %array to keep track of
                array size of signals being plotted
```

```
            sigidx(count)=rptidx; %array to keep track of which signals can be plotted
        case {'SIG_TxIF','SIG_TxRF'},
            if ~CFG_BYPASS %don't put in plot list if bypass enabled
                load(CFG_txsim_outFile,Sigs{SELECTION(rptidx)}); %get variable
                count=count+1;
                SIGNAL{rptidx}=[Sigs{SELECTION(rptidx)},'{k}'];
                SZ(count,1:2)=size(eval(Sigs{SELECTION(rptidx)})); %array to keep track of
                    array size of signals being plotted
                sigidx(count)=rptidx; %array to keep track of which signals can be plotted
            end
        case {'SIG_RF'},
            if ~CFG_BYPASS %don't put in plot list if bypass enabled
                load(CFG_rfsim_outFile,Sigs{SELECTION(rptidx)}); %get variable
                count=count+1;
                SIGNAL{rptidx}=[Sigs{SELECTION(rptidx)},'{k,m}'];
                SZ(count,1:2)=size(eval(Sigs{SELECTION(rptidx)})); %array to keep track of
                    array size of signals being plotted
                sigidx(count)=rptidx; %array to keep track of which signals can be plotted
            end
        case {'SIG_RxIF'},
            if ~CFG_BYPASS %don't put in plot list if bypass enabled
                load(CFG_rxsim_outFile,Sigs{SELECTION(rptidx)}); %get variable
                count=count+1;
                SIGNAL{rptidx}=[Sigs{SELECTION(rptidx)},'{k,m}'];
                SZ(count,1:2)=size(eval(Sigs{SELECTION(rptidx)})); %array to keep track of
                    array size of signals being plotted
                sigidx(count)=rptidx; %array to keep track of which signals can be plotted
            end
        otherwise
        end
    end

    %get details about each plot
    for rptidx=1:count
        %get trials to plot
        prompt={'Enter row indices of trials:',...
                'Enter column indices of trials:'};
        def={num2str(1:min(length(CFG_HAM),SZ(rptidx,1))),num2str(1:min(length(CFG_SNR),SZ
            (rptidx,2)))};
        dlgTitle=['Power Spectrum of ',Sigs{SELECTION(sigidx(rptidx))}];
        lineNo=1;
        row=[];
        col=[];
        answer={'dummy'};
        while (isempty(row) | isempty(col)) & ~isempty(answer)
            answer=inputdlg(prompt,dlgTitle,lineNo,def);
            if ~isempty(answer)
                row=intersect(str2num(answer{1}),1:min(length(CFG_HAM),SZ(rptidx,1)));
                col=intersect(str2num(answer{2}),1:min(length(CFG_SNR),SZ(rptidx,2)));
                if isempty(row)
```

```
                waitfor(errordlg({'Row indices must be of:',num2str(1:min(length(
                    CFG_HAM),SZ(rptidx,1)))},...
                    'Incorrect Row Indices','on'));
                def={answer{1},answer{2}};
                row=[];
            end
            if isempty(col)
                waitfor(errordlg({'Column indices must be of:',num2str(1:min(length(
                    CFG_SNR),SZ(rptidx,2)))},...
                    'Incorrect Column Indices','on'));
                def={answer{1},answer{2}};
                col=[];
            end
        end
    end

    %if there is valid input plot the psd
    if ~isempty(row) & ~isempty(col)
        %check if extracted PSDs need to be plotted
        if strcmp(Sigs{SELECTION(sigidx(rptidx))},'EXT_PSD')
            %plot extracted PSDs
            for r=1:length(row)
                for c=1:length(col)
                    %get row anc column index
                    k=row(r); %do not delete this line:  required for the eval
                        function
                    m=col(c); %do not delete this line:  required for the eval
                        function

                    %generate figure
                    figure
                    plot(EXT_PSD(k,m).Freq/signs_engunt(EXT_PSD(k,m).Freq),10*log10(
                        EXT_PSD(k,m).Pxx),getnextlinetype(CFG_RPTTYP));
                    grid on

                    %attach labels
                    xlabel(['Frequency (',signs_engunt(CFG_Fs,'prefix'),'Hz)'])
                    ylabel('Power Spectrum Magnitude (dB)')

                    %choose correct title
                    if CFG_REALRX
                        %titles for real baseband data
                        switch CFG_MODL
                        case 'm-fsk',
                            title(['PSD of Real Rx Baseband (Trial #',num2str(k*m),';'
                                ,num2str(CFG_LVL),'-FSK',...
                                    ';\Deltaf=',num2str(CFG_Fdev),'Hz)']);
                        case 'm-psk',
                            title(['PSD of Real Rx Baseband (Trial #',num2str(k*m),';'
                                ,num2str(CFG_LVL),'-PSK',...
```

```
                        ';\phi=',num2str(CFG_Po),'rad)']);
                case 'stanag4285',
                    title(['PSD of Real Rx Baseband (Trial #',num2str(k*m),';
                        STANAG4285 (8-PSK)',...
                            ';baud=',num2str(CFG_R),')']);
                end
            else
                %titles for synthetic baseband data
                switch CFG_MODL
                case 'm-fsk',
                    title(['PSD of Synthetic Rx Baseband (Trial #',num2str(k*m
                        ),';',num2str(CFG_LVL),'-FSK',...
                            ';\Deltaf=',num2str(CFG_Fdev),'Hz;HD=',num2str(
                                SIG_HAM(k)),...
                            ';SNR=',num2str(CFG_SNR(m)),'dB)']);
                case 'm-psk',
                    title(['PSD of Synthetic Rx Baseband (Trial #',num2str(k*m
                        ),';',num2str(CFG_LVL),'-PSK',...
                            ';\phi=',num2str(CFG_Po),'rad;HD=',num2str(SIG_HAM
                                (k)),...
                            ';SNR=',num2str(CFG_SNR(m)),'dB)']);
                case 'stanag4285',
                    title(['PSD of Synthetic Rx Baseband (Trial #',num2str(k*m
                        ),';STANAG4285 (8-PSK)',...
                            ';baud=',num2str(CFG_R),';HD=',num2str(SIG_HAM(k))
                                ,...
                            ';SNR=',num2str(CFG_SNR(m)),'dB)']);
                end
            end
        end
    end
else
    %otherwise use the PSD function
    for r=1:length(row)
        for c=1:length(col)
            %get row anc column index
            k=row(r); %do not delete this line:  required for the eval
                function
            m=col(c); %do not delete this line:  required for the eval
                function

            %generate figure
            figure

            %compute segment size for psd
            WINDOWsize=CFG_NFFT;
            NOVERLAP=round(WINDOWsize*CFG_OVR/100);
            DFLAG='none';

            %plot psd
```

```
                psd(eval(SIGNAL{rptidx}),CFG_NFFT,CFG_Fs/signs_engunt(CFG_Fs),
                    CFG_NFFT,NOVERLAP,DFLAG);

                %write axis labels
                xlabel(['Frequency (',signs_engunt(CFG_Fs,'prefix'),'Hz)'])
                %ylabel('PSD (dB)')

                %choose correct title
                switch CFG_MODL
                case 'm-fsk',
                    title(['PSD of ',strrep(Sigs{SELECTION(sigidx(rptidx))},'_','\
                        _'),' Trial #',num2str(k*m),' (',num2str(CFG_LVL),'-FSK'
                        ,...
                            ';\Deltaf=',num2str(CFG_Fdev),'Hz;HD=',num2str(SIG_HAM
                                (k)),...
                            ';SNR=',num2str(CFG_SNR(m)),'dB)']);
                case 'm-psk',
                    title(['PSD of ',strrep(Sigs{SELECTION(sigidx(rptidx))},'_','\
                        _'),' Trial #',num2str(k*m),' (',num2str(CFG_LVL),'-PSK'
                        ,...
                            ';\phi=',num2str(CFG_Po),'rad;HD=',num2str(SIG_HAM(k))
                                ,...
                            ';SNR=',num2str(CFG_SNR(m)),'dB)']);
                case 'stanag4285',
                    title(['PSD of ',strrep(Sigs{SELECTION(sigidx(rptidx))},'_','\
                        _'),' Trial #',num2str(k*m),' (STANAG4285 (8-PSK)',...
                            ';baud=',num2str(CFG_R),';HD=',num2str(SIG_HAM(k)),...
                            ';SNR=',num2str(CFG_SNR(m)),'dB)']);
                end
            end
        end
    end
end
catch
    %log error condition
    signs_log(CFG_logfile,dbstack);
    SUCC=1;
end
return

%function to return a line type for plots with multiple curves
function y=getnextlinetype(TYP)
persistent count
LineColour='bgrcmykbgrcmykbgrcmykbgrcmykbgrcmykbgrcmyk';
LineSample='ox+*sdvx+*sdvo+*sdvox*sdvox+sdvox+*dvox+*svox+*sd';
if isempty(count)
    count=1;
end
```

```
%choose plot type
switch TYP
case 'scatter',
    y=[LineColour(count),LineSample(count)];
case 'scatter-solid',
    y=[LineColour(count),LineSample(count),'-'];
case 'scatter-dotted',
    y=[LineColour(count),LineSample(count),':'];
case 'scatter-dashdot',
    y=[LineColour(count),LineSample(count),'-.'];
case 'scatter-dashed',
    y=[LineColour(count),LineSample(count),'--'];
case 'solid',
    y=[LineColour(count),'-'];
case 'dotted',
    y=[LineColour(count),':'];
case 'dashdot',
    y=[LineColour(count),'-.'];
case 'dashed',
    y=[LineColour(count),'--'];
end
count=mod(count-1,length(LineColour))+2;
return
```

H.7 Ⓢigns Miscellaneous Functions

The miscellaneous functions are used by the analysis modules, data generation tools, feature extraction tools, and reporting tools. Some functions are an integral part of the Ⓢ**igns** toolbox but, others are not. All, though, are useful in analyzing signal data and are therefore included in the toolbox.

SIGNS_AISBETTHASH

```
%Sign(s) Tool: SIGNS_AISBETTHASH
%
%[Z,ERR]=SIGNS_AISBETTHASH(X,Y)
%
%The purpose of this function is to return the product of the time-based
%means of X and Y less the time-based covariance of X and Y.
%
%Mandatory Input Parameters:
%   X -         arbitrary numeric vector of length N
%   Y -         arbitrary numeric vector of length N
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   None
%
%Mandatory Output Parameters:
%   Z -         product of the time-based means of X and Y less the
%               time-based covariance or running average of #(X,Y) as specified
%               by WIN
%
%Optional Output Parameters:
%   ERR -       two element cell array. The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   This function is based on theory published by the Australian Department of
%Defence in Technical Report ERL-0367-TR in 1986 by Janet Aisbett. In this paper
%the hash function is defined as
%
% #(X,Y)=2*E{X}*E{Y}-E{XY}
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version  Editor           Changes Made
%-----------  -------  ---------------  --------------------
```

```
function [Z,ERR]=signs_aisbetthash(X,Y)
try
    if length(X)==length(Y)
        N=length(X);

        %compute the hash function
        Z=(2/N^2)*sum(X)*sum(Y)-1/N*sum(X.*Y);

        ERR=[];
    else
        ERR={dbstack,'Input vector dimensions must agree'};
        Z=0;
    end
catch
    %capture error condition
    ERR={dbstack,lasterr};
    Z=0;
end
return
```

SIGNS_ANALOGOUT

```
%Sign(s) Tool: SIGNS_ANALOGOUT
%
%[Y,ERR]=SIGNS_ANALOGOUT(DATA,FS,FC,BW,SB,WIN,PLT,WAVFILE)
%
%The purpose of this function is to downconvert a signal from a data vector to baseband
%and to optionally write a 16-bit wave file of the signal.
%
%Mandatory Input Parameters:
%   DATA -      real or complex time-series data
%   FS -        sampling rate of DATA in Hz
%   FC -        desired sub-carrier frequency (Hz)
%   BW -        desired bandwidth (Hz) about FC such that BW=[FC-BW/2, FC+BW/2]
%   SB -        side band ('usb', 'lsb', or 'dsb'; the default is 'dsb')
%   SR -        output sample rate for wave file
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   WIN -       window used for filtering.  The window vector can be any
%               valid Matlab window.  See HELP WINDOW for valid windows.  For
%               example:  a 256-pt hanning window would be specified by WIN='@hann,256'.
%               The default is a 256-pt hanning window.
%   PLT -       set to 1 to obtain a PSD of the passband
%   WAVFILE -   filename of wavefile of passband. The filename must include the .wav
%               extension.
%
%Mandatory Output Parameters:
%   Y -         baseband output data
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   None
%
%Copyright (c) 2005 James Giesbrecht
%Revision History:
%Date        Version   Editor            Changes Made
%--------    -------   ------------      --------------------
%27/04/06    1.1       JEG               added downsampling feature

function [Y,ERR]=signs_analogout(DATA,FS,FC,BW,SB,SR,varargin)
try
    %default conditions
    ERR=[];
    WIN=hann(256); %default window for filtering
    NBITS=16; %number of bits per sample for the wave file
    PH=0; %phase for oscillator
```

```
PLT=0; %do not plot passband
WAVFILE=[]; %don't write wave file
BW=abs(BW); %ensure BW is positive

%get optional parameters parameter
if nargin>6 & nargin<10
    for j=1:length(varargin)
        if ischar(varargin{j}) & strcmp(varargin{j}(end-3:end),'.wav')
            WAVFILE=varargin{j};
        else
            if ischar(varargin{j})
                WIN=eval(['window(',varargin{j},')']);
            else
                PLT=varargin{j};
            end
        end
    end
end

%get number of FIR taps
N=length(WIN)-1;

%design LPF
Wn=abs(BW)/FS*2; %get normalized frequencies
Blpf=fir1(N,Wn,'low',WIN);
Alpf=1;

%design BPFs
switch SB
case {'usb'},
    Wn=[FC,FC+BW]/FS*2; %get normalized frequencies
    Bbpf1=fir1(N,Wn,'bandpass',WIN);
    Abpf1=1;
case {'lsb'},
    Wn=[FC-BW,FC]/FS*2; %get normalized frequencies
    Bbpf1=fir1(N,Wn,'bandpass',WIN);
    Abpf1=1;
otherwise
    Wn=[FC-BW/2,FC+BW/2]/FS*2; %get normalized frequencies
    Bbpf1=fir1(N,Wn,'bandpass',WIN);
    Abpf1=1;
    Wn=[3*FC-BW/2,3*FC+BW/2]/FS*2; %get normalized frequencies
    Bbpf2=fir1(N,Wn,'bandpass',WIN);
    Abpf2=1;
end

%get IQ data
I=real(DATA);
Q=imag(DATA);
```

```
%apply BPF if USB or LSB
I=filter(Bbpf1,Abpf1,I);
Q=filter(Bbpf1,Abpf1,Q);

%mix IQ data
switch SB
case {'usb','lsb'},
    Wn=FC/FS*2*pi;
    I=I.*cos(Wn*[0:length(I)-1]+PH);
    Q=Q.*sin(Wn*[0:length(Q)-1]+PH);
otherwise
    %mix up DSB signal if it can't be mixed down directly
    if FC<3*BW/2
        Wn=2*FC/FS*2*pi;
        I=I.*cos(Wn*[0:length(I)-1]+PH);
        Q=Q.*sin(Wn*[0:length(Q)-1]+PH);

        %apply second BPF
        I=filter(Bbpf2,Abpf2,I);
        Q=filter(Bbpf2,Abpf2,Q);

        %mix down DSB signal
        Wn=(3*FC-BW/2)/FS*2*pi;
        I=I.*cos(Wn*[0:length(I)-1]+PH);
        Q=Q.*sin(Wn*[0:length(Q)-1]+PH);
    else
        %mix down DSB directly
        Wn=(FC-BW/2)/FS*2*pi;
        I=I.*cos(Wn*[0:length(I)-1]+PH);
        Q=Q.*sin(Wn*[0:length(Q)-1]+PH);
    end
end

%apply LPF
I=filter(Blpf,Alpf,I);
Q=filter(Blpf,Alpf,Q);

%apply hilbert transform (i.e. -90deg phase shift)
Qhilb=imag(hilbert(Q));

%combine I&Q
switch SB
case {'usb'},
    Y=I+Qhilb;
case {'lsb'},
    Y=I-Qhilb;
otherwise
    Y=I-Q;
end
```

```
    %apply downsampling
    Y=downsample(Y,round(FS/SR));

    %truncate data to a 16-bit fraction
    Y=32767.5*(Y/max(abs(Y)))-.5;
    Y=fix(Y*2)/65536;

    %write wavefile
    if ~isempty(WAVFILE)
        wavwrite(Y,SR,NBITS,WAVFILE);
    end

    %plot passband
    if PLT
        NFFT=2^nextpow2(FS)/32; %seems reasonable; NFFT=1024 for 16384<FS<=32768
        figure,psd(Y,NFFT,FS);
        xlabel('Frequency (Hz)')
        title('PSD of Desired Passband')
    end
catch
    %capture error condition
    ERR={dbstack,lasterr};
    Y=[];
end
return
```

SIGNS_APWIN

```
%Sign(s) Tool: SIGNS_APWIN
%
%[Y,ERR]=SIGNS_APWIN(X,WIN)
%
%The purpose of this function is to apply a sliding window to a data sequence.
%
%Mandatory Input Parameters:
%   X -         input data sequence vector or array (real or complex).  If X is
%               an NxM array, the window is applied to the longest dimension of X.
%               If X is an NxN array, the windowing is applied to the rows of X.
%   WIN -       averaging window for coherence output
%               (e.g. a 7 sample boxcar window is specified by WIN = '@rectwin,7')
%               (see Matlab's window command for more window options)
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   None
%
%Mandatory Output Parameters:
%   Y -         averaged sequence
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   None
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%---------   -------   ------------      --------------------
%10/03/05    1.1       J. Giesbrecht     Added matrix support
%13/02/08    1.2       J. Giesbrecht     Added check for WIN longer than X
function [Y,ERR]=signs_apwin(X,WIN)

try
    ERR=[];

    %prepare input array
    if size(X,1)>size(X,2)
        A=X';
    else
        A=X;
    end
    sz=size(A);
```

```
    %prepare sliding window
    win=eval(['window(',WIN,')']);
    win=win/length(win);
    if mod(length(win),2)==0
        st=length(win)/2;
        off=-1;
    else
        st=(length(win)-1)/2;
        off=0;
    end

    %check that window is not longer than longest dimension of X
    if length(win)>max(size(X))
        error('Not enough data for window.')
    end

    %apply sliding window to longest dimension of X
    for j=1:sz(1)
        count=1;
        for k=st+1:sz(2)-st
            Y(j,count)=A(j,k-st:k+st+off)*win;
            count=count+1;
        end
    end

    %prepare output array
    if size(X,1)>size(X,2)
        Y=Y';
    end
    ERR=[];
catch
    %capture error condition
    ERR={dbstack,lasterr};
    Y=0;
end
return
```

SIGNS_BIKAPPAPDF

```
%Sign(s) Tool: SIGNS_BIKAPPAPDF
%
%[Y,ERR]=SIGNS_BIKAPPAPDF(X,SIGMA,K,OPT,YOPT)
%
%The purpose of this function is to return the modified bi-kappa pdf with non-extensivity
%index, K, and variance SIGMA at the values in X, as defined in the PhD thesis, "Aspects
%of HF Communications: HF Noise and Signal Features", by James Giesbrecht.  The PDF that is
%returned is normalized over the range of X.
%
%Mandatory Input Parameters:
%   X -         vector of observations at which to compute the pdf
%   SIGMA -     standard deviation of distribution.  SIGMA can be a vector but only if
%               K is constant.  SIGMA must be greater than zero.
%   K -         degree of nonextensivity or non-locality of system (K>0.5)
%               K can be a vector but only if SIGMA is a constant.
%
%Optional Input Parameters:
%   OPT -       if OPT='plot', the distribution(s) will be plotted
%               but if OPT='plotfit', the bi-kappa distribution will be fitted to the
    distribution
%               specified by the observations in X and the probabilities in YOPT.  The fitted
    distribution
%               will also be plotted.  if OPT='fit' the bi-kappa distribution will only be
    fitted.
%   YOPT -      probabilities of observed distribution to which the bi-kappa distribution
    should
%               be fitted.  YOPT must not contain the log of the probabilties but must be the
%               absolute probabilities measured at the observations in X.
%
%Mandatory Output Parameters:
%   Y -         bi-kappa pdf.  If K or SIGMA is a vector Y will contain an MxN array of
%               distributions where M is the number of variations in either K or
%               SIGMA and where N is the number of observations less TAU.  Each row of the MxN
%               array will correspond to the consecutive values of K or SIGMA.
%               Y is normalized over the range of X so that sum(Y)=1.
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   References:
%      [1]  Leubner M. P. and Voros Z., "A nonextensive entropy path to probability
%           distributions in solar wind turbulence", 2005, Nonlinear Processes in
%           Geophysics, vol 12, pp:171-180
%
%   Leubner and Voros define the bi-kappa distribution a superposition of a halo component
```

```
%   and a core component.  The core component is the same as the halo component for negative
%   values of k.
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor          Changes Made
%----------  --------  --------------  --------------------
%16/03/07    1.1       JEG             Added 'plotfit' and 'fit' options
%19/09/07    1.2       JEG             Normalized pdf so integral of pdf is 1
%12/12/07    1.3       JEG             Added explanation of normalizing factor
function [Y,ERR]=signs_bikappapdf(X,SIGMA,K,varargin)
try
    %check input arguments
    if nargin<3
        error('Not enough input arguments')
    end
    PLT=0;
    FIT=0;
    if min(SIGMA)<=0
        error('Standard deviation invalid');
    end
    if min(K)<=0.5
        error('Kappa must be greater than 0.5');
    end
    if nargin>3
        if strcmp(varargin{1},'plot')
            PLT=1;
        elseif strcmp(varargin{1},'plotfit') | strcmp(varargin{1},'fit')
            if strcmp(varargin{1},'plotfit')
                PLT=1;
            end
            FIT=1;
            if size(varargin{2})==size(X) & isnumeric(varargin{2})
                YOPT=varargin{2};
            else
                error('Argument YOPT is invalid');
            end
        else
            error([varargin{1},' is invalid'])
        end
    end
    if prod(size(SIGMA))>1 & prod(size(K))>1
        error('Both SIGMA and K cannot be vectors simultaneously')
    end
    if find(SIGMA<=0)
        error('SIGMA must be greater than zero')
    end

    %prepare output array
    Yo = zeros(max(length(SIGMA),length(K)),length(X));
```

```
Yi = zeros(max(length(SIGMA),length(K)),length(X));

%compute distribution
if (length(SIGMA)>=1 & length(K)==1) | (length(SIGMA)==1 & length(K)>=1)
    J=max(length(SIGMA),length(K));
    Lstring='legend('; %prepare legend string
    if length(SIGMA)>1 & length(K)==1
        for j=1:J
            Y(j,:)=bikappa(X,SIGMA(j),K);
            if FIT
                Y(j,:)=Y(j,:)*max(YOPT)/max(Y(j,:));
            end
            Lstring=[Lstring,'''{\sigma}=',num2str(SIGMA(j)),''','];
        end
    elseif length(SIGMA)==1 & length(K)>1
        for j=1:J
            Y(j,:)=bikappa(X,SIGMA,K(j));
            if FIT
                Y(j,:)=Y(j,:)*max(YOPT)/max(Y(j,:));
            end
            Lstring=[Lstring,'''{\kappa}=',num2str(K(j)),''','];
        end
    else
        Y=bikappa(X,SIGMA,K);
        if FIT
            Y=Y*max(YOPT)/max(Y);
        end
    end

    %plot curves if necessary
    if PLT
        %finalize legend string (even if it is not required -- just so I don't have to add
            an IF statement)
        Lstring=[Lstring(1:end-1),',0)']; %add positional information

        %plot distribution
        for j=1:J
            semilogy(X,Y(j,:),getnextlinetype('solid'));
            hold on
        end
        title(['Bi-Kappa␣Distribution␣(',num2str(min(K)),'␣\leq␣\kappa␣\leq␣',num2str(max(
            K)),';',...
                num2str(min(SIGMA)),'␣\leq␣\sigma␣\leq␣',num2str(max(SIGMA)),')']);
        xlabel('Observation');
        ylabel('Probability');
        grid on
        %add legend if needed
        if max(length(SIGMA),length(K))>1
            eval(Lstring);
        end
```

```
            grid on
            hold off
        end
    end
    ERR=[];
catch
    %capture error condition
    ERR={dbstack,lasterr};
    Y=[];
end
return

%the generalized bi-kappa distribution
function y=bikappa(x,sig,k)
if k==0
    %In the limit...
    %
    %  lim y = 1
    %  k->0
    %
    %however this state is not allowed, as the normalization constant is
    %undefined.  We can artificially normalize
    y=ones(1,length(x));
else
    if k<0 %branch never actually taken as K must be greater than 0.5
        %compute "core" component
        y=(1-x.^2/k/(sig^2)).^k;
    else
        %compute "halo" component
        y=(1+x.^2/k/(sig^2)).^-k;
    end
end
%nomalize result so that sum of probabilities is 1 over the range of X values;
%over an infinite range of X values, the normalization factor is
%
%                1                    gamma(k)
%   C = ------------------------ X ------------------------
%       sig * (sqrt(k))^-1       gamma(0.5) gamma(k-0.5)
% OR
%                1                    1
%   C = ------------------------ X ----------------
%       sig * (sqrt(k))^-1       beta(0.5,k-0.5)
%
%but because the area under the probability curve is sampled at each
%value of X, we need to add in the X interval, that is, dX.  Hence
%the normalization constant for a digital implementation of the distribution is
%
%                          dX
%   C = ------------------------------------------
%       sig * (sqrt(k))^-1 * beta(0.5,k-0.5)
```

```
%
%   Cn=1./sig./sqrt(k)./beta(0.5,k-0.5)
%
%But over a finite range of X values, the normalization factor is
%
%              1
%   C = ---------------
%         F(b) -F(a)
%
%where F(.) is the cumulative distribution function evaluated at X=b and X=a.
%This difference is also equivalent to the sum(y) over the range.
y=y/sum(y);
return

%function to return a line type for plots with multiple curves
function y=getnextlinetype(TYP)
persistent count
LineColour='bgrcmykbgrcmykbgrcmykbgrcmykbgrcmykbgrcmyk';
LineSample='ox+*sdvx+*sdvo+*sdvox*sdvox+sdvox+*dvox+*svox+*sd';
if isempty(count)
    count=1;
end
%choose plot type
switch TYP
case 'scatter',
    y=[LineColour(count),LineSample(count)];
case 'scatter-solid',
    y=[LineColour(count),LineSample(count),'-'];
case 'scatter-dotted',
    y=[LineColour(count),LineSample(count),':'];
case 'scatter-dashdot',
    y=[LineColour(count),LineSample(count),'-.'];
case 'scatter-dashed',
    y=[LineColour(count),LineSample(count),'--'];
case 'solid',
    y=[LineColour(count),'-'];
case 'dotted',
    y=[LineColour(count),':'];
case 'dashdot',
    y=[LineColour(count),'-.'];
case 'dashed',
    y=[LineColour(count),'--'];
end
count=mod(count-1,length(LineColour))+2;
if count>length(LineColour)
    count=1;
end
return
```

SIGNS_CMHD

```
%Sign(s) Tool: SIGNS_CMHD
%
%[W,ERR]=SIGNS_CMHD(X,Y,NFFT,NRM,PLT)
%
%The purpose of this function is to compute the time-shift invariant
%and frequency-shift invariant cross-Margenau-Hill distribution.  The CMHD
%is defined by
%                      -
%                     /
% CMHD(t,f) = 1/2     | {x(t+m)y*(t)+x(t)y*(t-m)}exp(-j2pfm) dm
%                   -/
%                    m
%
%where t is time, f is frequency, p is pi, and m is the shift. * is the conjugate operator.
%If X=Y then CMHD(t,f) becomes the auto-MHD.  In this case the integral of
%CMHD(t,f) over frequency yields instantaneous power of X (i.e. |x(t)|^2) while the integral
%of CMHD(t,f) over time yields spectral energy density of X (i.e. |X(f)|^2).  If X<>Y then
%CMHD(t,f) becomes the cross-MHD (hence CMHD).  In this case the integral of CMHD(t,f)
%over time yields the cross-spectral density of XY (i.e. |X(f)Y*(f)|) and the integral of
%CMHD(t,f) over frequency yields the instantaneous cross-power of XY (i.e. |x(t)y*(t)|).
%
%In the discrete domain, CMHD(t,f) can be written as
%
%                     N/2
% CMHD(m,k) = 1/2    sum     {x(m+n)y*(m)+x(m)y*(m-n)}exp(-j2*pi*n/N*k)
%                   n=-N/2
%
%Mandatory Input Parameters:
%   X      - a real or complex data vector on which to compute the CMHD.  If
%            the length of X is shorter than the length of Y then Y is truncated to the length
%            of X.
%   Y      - a real or complex data vector on which to compute the CMHD distribution.  If
%            the length of Y is shorter than the length of X then X is truncated to the length
%            of Y.
%
%Optional Input Parameters:
%   NFFT   - number of frequency bins for the distribution (default is 128)
%   NRM    - set to 'normal' to normalize X and Y such that it's energy is unity, by default X
%            and Y are not normalized
%   PLT    - set to 'plot' to plot the distribution, by default SIGNS_CMHD does not plot
%
%Mandatory Output Parameters:
%   W - contains the CMHD of X and Y.  W is complex and its size is NxP
%       where N is the length of X and P is the length of Y.  Rows of W correspond to the time
%       dimension whereas columns of W correspond to normalized frequency.
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
```

```
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   None
%
%Copyright (c) 2005 James Giesbrecht

%Company:
%
%Revision History:
%Date        Version   Editor            Changes Made
%-----------  --------  ----------------  ----------------------
%
function [W,ERR]=signs_cmhd(X,Y,varargin)

ERR=[];
try
    %set default values
    PLT=0;
    NRM=0;
    N=min(length(X),length(Y));
    NFFT=128;
    if nargin>2
        for j=1:length(varargin)
            if ischar(varargin{j})
                if strcmp(varargin{j},'plot') %determine plotting
                    PLT=1;
                end
                if strcmp(varargin{j},'normal') %determine if normalization is required
                    NRM=1;
                end
            end
            if isnumeric(varargin{j})
                NFFT=fix(varargin{j}); %get size of fft
            end
        end
    end
    W=zeros(N,NFFT); %prepare distribution array
    X=reshape(X,1,N); %ensure X is a row vector
    Y=reshape(Y,1,N); %ensure Y is a row vector

    %if necessary normalize X and Y to unity energy
    if NRM
        X=X/sqrt(sum(X.^2));
        Y=Y/sqrt(sum(Y.^2));
    end

    %pad input vectors
    Xp=[zeros(1,N),X,zeros(1,N+1)];
```

```
    Yp=[zeros(1,N),Y,zeros(1,N+1)];
    tc=N+1; %find the first time index of X and Y in Xp and Yp

    %compute CMHD
    m=[-(tc-1):(tc-1)];
    for n=tc:tc+N-1
        q1=Xp(n+m).*conj(Yp(n));
        q2=Xp(n).*conj(Yp(n-m));
        W(n-tc+1,1:NFFT)=0.5*fftshift(fft(q1+q2,NFFT));
    end

    %plot distribution if necessary
    if PLT
        figure
        mesh(1:size(W,1),-size(W,2)/2:size(W,2)/2-1,abs(W'));
        xlabel('Time Index')
        ylabel('Frequency Index')
        title('Magnitude of Auto/Cross MHD')

        figure
        mesh(1:size(W,1),-size(W,2)/2:size(W,2)/2-1,angle(W'));
        xlabel('Time Index')
        ylabel('Frequency Index')
        title('Phase of Auto/Cross MHD (rad)')
    end

catch
    %capture error condition
    ERR={dbstack,lasterr};
    W=[];
end
return
```

SIGNS_DAUBWAVELET

```
%Sign(s) Tool: SIGNS_DAUBWAVELET
%
%[C,ERR]=SIGNS_DAUBWAVELET(X,SCALE,SHIFT,BSC,PLT)
%
%The purpose of this function is to return Daubechies' basic wavelet.
%Daubechies' wavelet is defined by
%
% W(r)=-H0*u(2r-1)+H1*u(2r)-H2*u(2r+1)+H3*u(2r+2)
%
%where
%          1+sqrt(3)            3+sqrt(3)            3-sqrt(3)            1-sqrt(3)
%    H0 = ----------- ,  H1 = ----------- ,  H2 = ----------- ,  H3 = -----------
%             4                    4                    4                    4
%
%and where u(r) is defined by the recurrence relation
%
%    u(r) = H0*u(2r)+H1*u(2r-1)+H2*u(2r-2)+H3*u(2r-3)
%
%with initial conditions
%
%    u(r)=0 for 0>=r>=3
%
%and
%
%    u(0) = 0 , u(1) = 2*H0 , u(2) = 2*H3 , u(3) = 0.
%
%Mandatory Input Parameters:
%    X -          Numeric vector for which the elements are evenly spaced by 2^n
%                 where n is any integer on the integer number line.  In other words
%                 X must be a vector of dyadic numbers (an integral multiple of an
%                 integral power of 2)
%    SCALE -      Scale factor that expands(-1<=SCALE<=1) or contracts(-1>SCALE>1) wavelet.
%                 A negative scale factor performs a horizontal flip of the wavelet.  For no
%                 scaling set SCALE=1.
%    SHIFT -      Shift factor that moves wavelet left (SHIFT>0) or right(SHIFT<0).  For no
%                 shift set SHIFT=0.
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%    BSC -        set to 'basic' to return Daubechies' basic building block rather than the
     wavelet.
%    PLT -        set to 1 to display a graphical version of the wavelet transform
%                 the default is 0
%
%Mandatory Output Parameters:
%    C -          the Daubechies wavelet or basic building block
%
%Optional Output Parameters:
%    ERR -        two element cell array.  The first element is a structure array
```

```
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   Reference: Nievergelt, Yves; "Wavelets Made Easy", Birkhauser, 1999.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%----------  --------  ----------------  --------------------

function [C,ERR]=signs_daubwavelet(X,SCALE,SHIFT,varargin)
try
    %ensure X is dyadic
    if signs_isdyadic(X)
        %set default parameters
        PLT=0;
        BSC=0;

        %extract optional input parameters
        if nargin>3
            for j=1:nargin-3
                if isnumeric(varargin{j})
                    PLT=varargin{j}(1);
                end
                if ischar(varargin{j})
                    if strcmp('basic',varargin{j})
                        BSC=1;
                    end
                end
            end
        end

        %generate the wavelet
        H=[(1+sqrt(3))/4,(3+sqrt(3))/4,(3-sqrt(3))/4,(1-sqrt(3))/4];%compute coefficients
        R=SCALE*X+SHIFT;
        if BSC
            for j=1:length(R)
                C(j)=psi(R(j),H);
            end
        else
            for j=1:length(R)
                C(j)=H*[-psi(2*R(j)-1,H);psi(2*R(j),H);-psi(2*R(j)+1,H);psi(2*R(j)+2,H)];
            end
        end

        %Plot the wavelet or building block if necessary
        if PLT
            figure
```

```
            plot(X,C,'b-')
            grid on
            xlabel('Dyadic Number, R')
            ylabel('Magnitude')
            if BSC
                title('Daubechies'' Basic Building Block')
            else
                title('Daubechies'' Wavelet')
            end
        end
        ERR=[];
    else
        C=0;
        ERR={dbstack,'X is not dyadic'};
    end
catch
    %capture error condition
    ERR={dbstack,lasterr};
    C=0;
end
return

%Daubechies' basic building block
function y=psi(r,H)
if r<=0 | r>=3
    y=0;
elseif r==1
    y=2*H(1);
elseif r==2
    y=2*H(4);
else
    y=H*[psi(2*r,H);psi(2*r-1,H);psi(2*r-2,H);psi(2*r-3,H)];
end
return
```

SIGNS_DAUBWT

```
%Sign(s) Tool: SIGNS_DAUBWT
%
%[D,ERR]=SIGNS_DAUBWT(X,PLT)
%
%The purpose of this function is to return basic Daubechies' Wavelet Transform
%of the vector X.  The basic Daubechies wavelet is defined by
%
% W(r) = -H[0]*psi(2r-1) + H[1]*psi(2r) - H[2]*psi(2r+1) + H[3]*psi(2r+2)
%
%where r is a dyadic number and psi is defined as
%
% psi(r) = H[0]*psi(2r) + H[1]*psi(2r-1) + H[2]*psi(2r-2) + H[3]*psi(2r-3)
%
%and where psi(0) = 0, psi(1) = 1/2(1+sqrt(3)), psi(2) = 1/2(1-sqrt(3)),
%and psi(3) = 0.  The coefficients Hi are
%
% H[0] = 1/4(1+sqrt(3))
% H[1] = 1/4(3+sqrt(3))
% H[2] = 1/4(3-sqrt(3))
% H[3] = 1/4(1-sqrt(3))
%
%The wavelet transform is then
%
%              2^N-1                    2^N-1
% DT{X}(r) =    sum   a[k] psi(r/2-k) +  sum    c[k] W(r/2-1-k).
%               k=0                      k=0
%
%In general, the coefficients a[k] and c[k] are computed by
%
%              L-1
% a[k] =  1/2 sum  H[i] b[i+2k]
%              i=0
%
%and
%
%              L-1
% c[k] =  1/2 sum  ((-1)^i) H[L-1-i] b[i+2k],
%              i=0
%
%where L is the number of wavelet coefficients (in this case 4), and b[k] is defined as
%
%           k+2
% b[k] =    sum   psi(r-k) X[m].
%         m=k+1
%
%Mandatory Input Parameters:
%   X -          arbitrary numeric vector of length 2^N.  If the length of
%                X is not a power of 2, it will be padded with zeros to
```

```
%               reach the next nearest power of 2 length.
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   PLT -       set to 1 to display a graphical version of the wavelet transform
%               the default is 0
%
%Mandatory Output Parameters:
%   D -         the Daubechies' Wavelet Transform of length 2^N.
%
%               Fast Daubechies' Wavelet Transform:
%                   explain results
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   Reference: Nievergelt, Yves; "Wavelets Made Easy", Birkhauser, 1999.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date         Version   Editor            Changes Made
%_________    _______   _____________     ____________________

function [D,ERR]=signs_daubwt(X,varargin)
try
    %determine the the plot flag
    PLT=0;
    if nargin>1
        for j=1:max(length(varargin),1)
            if isnumeric(varargin{j})
                PLT=varargin{j};
            end
        end
    end

    %get the input vector and reshape to row vector
    sz=size(X);
    S=reshape(X,1,length(X));

    %ensure the input vector has a length that is a power of 2
    if mod(log2(length(S)),1)~=0
        N=nextpow2(length(S)); %get next power of two
        S=[S,zeros(1,2^N-length(S))]; %pad the input vector
    else
        N=fix(log2(length(S)));
    end
```

```
    %add smoothed extensions
    N=length(S);
    X=[-N:2*N-1];
    m=[S(2)-S(1),S(N)-S(N-1)];
    b=[S(1),S(N)*(2-N)+S(N-1)*(N-1)];
    Sp=[0:3,4,2,1,-1,0,1];%[m(1)*X(1:N)+b(1),S,m(2)*X(2*N+1:3*N)+b(2)];

    %compute basic building block and basic wavelet
    WIN=4;
    H=[(1+sqrt(3))/4,(3+sqrt(3))/4,(3-sqrt(3))/4,(1-sqrt(3))/4];
    psi=signs_daubwavelet([0:WIN-1],1,0,'basic');
    W=signs_daubwavelet([0:WIN-1],1,-1);

  %compute shifted building blocks
    M=0;
    r=[0:2^-M:N-1];
  SHIFT=[2:-1:-(N-2)];
  PSI_shift=zeros(length(SHIFT),N);
  for k=1:length(SHIFT)
        PSI_shift(k,:)=signs_daubwavelet(r,0.5,SHIFT(k),'basic');
  end

    %compute coefficients b[k]
    Q=2*N+2;
    B=zeros(1,Q);
    for j=1:Q-WIN+1
        j:j+WIN-1
        B(j)=Sp(j:j+WIN-1)*psi';
    end

    %derive transform matrix
    DPSIinv=zeros(N,Q);
    for j=1:N
        r=2*j-1;
        DPSIinv(r,2*j-1:2*j+2)=H;
        DPSIinv(r+1,2*j-1:2*j+2)=fliplr(H).*[1,-1,1,-1];
    end

    %compute the transform
    D=(0.5*DPSIinv*B')';

    %plot the transform if necessary
    if PLT
        figure
        bar(D);
        title('Fast Daubechies'' Wavelet Transform')
        xlabel('Wavelet Frequency Index')
        ylabel('Wavelet Coefficients')
    end
```

```
    %shape the output vector to a row or column vector
    if sz(1)>sz(2)
        D=D';
    end
    ERR=[];
catch
    %capture error condition
    ERR={dbstack,lasterr};
    D=0;
end
return
```

SIGNS_DECILE

```
%Sign(s) Tool: SIGNS_DECILE
%
%[Y,ERR]=SIGNS_DECILE(X,DEC,DIM)
%
%The purpose of this function is compute the nth decile of a vector X.
%For vectors, SIGNS_DECILE(X,DEC) is the nth decile (specified by DEC)
%of the elements in X.  For matrices, SIGNS_DECILE(X,DEC) is a row vector
%containing the nth decile value of each column.  For N-D arrays,
%SIGNS_DECILE(X,DEC) is the nth decile value of the elements along the
%first non-singleton dimension
%of X.
%
%   SIGNS_DECILE(X,DEC,DIM) takes the nth decile along the dimension DIM of X.
%
%   Example: If X = [0 1 2
%                    3 4 5]
%
%   then signs_decile(X,1,1) is [0.3 1.3 2.3]
%   and signs_decile(X,1,2) is [0.2
%                               3.2]
%
%   See also MEAN.
%
%Mandatory Input Parameters:
%   X -          vector on which to compute the decile
%   DEC -        decile, e.g. DEC=1 for 10% percentile, DEC=5 for 50% percentile (i.e. median),
%                DEC=1.7 for 17% percentile
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   DIM -        two element vector indicating which SIG_RxBS to plot.  The first
%
%Mandatory Output Parameters:
%   Y -          nth decile of X
%
%Optional Output Parameters:
%   ERR -        two element cell array.  The first element is a structure array
%                containing the debug trace information, the second element is the
%                error.
%
%Special Notes:
%   Function based on Matlab median.m
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%---------   -------   ------------      --------------------
%17/12/04    1.0       JEG               Converted median.m
```

```
%                                   to signs_decile.m
%(median.m is Copyright 1984-2001 The MathWorks, Inc. $Revision: 5.14 $  $Date: 2001/04/15
    12:01:27 $)
function [y,ERR] = signs_decile(X,DEC,varargin)

try
    if nargin==2,
        DIM = min(find(size(X)~=1));
        if isempty(DIM), DIM = 1; end
    else
        DIM=varargin{1};
    end

    if isempty(X), y = []; return, end

    siz = [size(X) ones(1,DIM-ndims(X))];
    n = size(X,DIM);

    % Permute and reshape so that DIM becomes the row dimension of a 2-D array
    perm = [DIM:max(length(size(X)),DIM) 1:DIM-1];
    X = reshape(permute(X,perm),n,prod(siz)/n);

    % Sort along first dimension
    X = sort(X,1);

    %get indices
    loidx=fix((n-1)*DEC/10)+1;
    del=(n-1)*DEC/10+1-loidx;
    hiidx=loidx+1;

    %compute percentile
    y=del/(hiidx-loidx)*(X(hiidx,:)-X(loidx,:))+X(loidx,:);

    % Check for NaNs
    y(isnan(X(1,:)) | isnan(X(n,:))) = NaN;

    % Permute and reshape back
    siz(DIM) = 1;
    y = ipermute(reshape(y,siz(perm)),perm);
    ERR=[];
catch
    %capture error condition
    ERR={dbstack,lasterr};
    y=[];
end
return
```

SIGNS_DYNRANGE

```
%Sign(s) Tool: SIGNS_DYNRANGE
%
%[Y,ERR]=SIGNS_DYNRANGE(X)
%
%The purpose of this function is to return an estimate of the dynamic range of a vector or
%2-D array X.  It is computed as the range in dB between the smallest resolvable
%element and the largest resolvable element assuming that elements are from
%an N-bit digitizer such that
%
%          <largest magnitude element>
% D. R. = ---------------------------
%          <smallest non-zero element>
%
%Mandatory Input Parameters:
%   X -          abritrary numeric vector or 2-D array
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   None
%
%Mandatory Output Parameters:
%   Y -          estimate of dynamic range of input vector or 2-D array
%
%Optional Output Parameters:
%   ERR -        two element cell array.  The first element is a structure array
%                containing the debug trace information, the second element is the
%                error.
%
%Special Notes:
%   None
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date         Version   Editor            Changes Made
%-----------  -------  -------------      --------------------

function [Y,ERR]=signs_dynrange(X)
try
    %prepare input data
    X=abs(X);

    %get minimum non-zero element
    idx=find(X>0);
    minX=min(min(X(idx)));

    %get maximum magnitude element
    maxX=max(max(X));
```

```
    %compute dynamic range
    Y=20*log10(maxX/minX);
    ERR=[];
catch
    %capture error condition
    ERR={dbstack,lasterr};
    Y=0;
end
return
```

SIGNS_EDITPATH

```
%Sign(s) Tool: EDIT PATH
%
%SUCC=SIGNS_EDITPATH(inputProfile)
%
%The purpose of this function is to modify the file paths in a profile.  This is particularly
    useful
%if the directory structure has changed after profiles for Sign(s) have been created.
%
%  Input Parameters:
%    inputProfile - string variable containing the name of the .mat file that contains
%                   the simulation profile.  The input profile is a .mat file that contains
%                   details of the simulation.  This file can be created with the profile tool
    .
%                   If inputProfile is empty, a profile selection window will appear to allow
%                   the user to select a profile.
%  Output Parameters:
%    SUCC - 0 if the function completed successfully, otherwise 1.
%  Special Notes:
%    See comments in code.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date         Version   Editor            Changes Made
%-----------  --------  ----------        --------------------

function SUCC=signs_editpath(inputProfile)
try
    %get profile
    if exist('inputProfile')
        load(inputProfile);
    else
        %if the profile wasn't passed as an argument, the use the gui
        [filename,path2file]=uigetfile('profile.mat','SIGN(s): Open Profile');
        if isequal(filename,0) | isequal(path2file,0) %check for errors
            disp('SIGN(s): File Open Error or Cancel Selected');
            SUCC=1;
            return
        else
            %prepare profile string
            filename=strjust(strjust(filename,'left'),'right'); %remove possible white space
            path2file=strjust(strjust(path2file,'left'),'right'); %remove possible white space
            inputProfile=[path2file,filename];

            %get key variables
            load(inputProfile);
        end
    end
```

```
%update file paths
pathVar={'CFG_logpath','CFG_profilespath','CFG_resultspath','CFG_temppath','
    CFG_realFileREF','CFG_realFileRX'};
temp=[];
for idx=1:size(pathVar,2)
    temp{idx}=[pathVar{idx},'=',eval(pathVar{idx})];
end
[SELECTION,OK]=listdlg('ListString',temp,'SelectionMode','multiple','ListSize',[1000,200],
    'Name','Current␣Paths',...
    'PromptString','Select␣Path␣Type␣to␣Change');
if OK
    %display help window
    disp(textwrap(temp,132));%msgbox(textwrap(temp,132),'Current Paths');

    %edit the paths
    for idx=1:length(SELECTION)

        %get new path and filename
        [filename,path2file]=uigetfile('*.*',['SIGN(s)␣:␣Select␣any␣file␣in␣the␣new␣
            directory␣to␣change␣',pathVar{SELECTION(idx)}]);
        if isequal(filename,0) | isequal(path2file,0) %check for errors
            disp('SIGN(s)␣:␣File␣Open␣Error␣or␣Cancel␣Selected');
            SUCC=1;
            return
        else
            %extract filename and path strings
            filename=strjust(strjust(filename,'left'),'right'); %remove possible white
                space
            path2file=strjust(strjust(path2file,'left'),'right'); %remove possible white
                space
        end

        %update filenames and paths
        switch pathVar{SELECTION(idx)}
        case 'CFG_logpath',
            CFG_logpath=path2file;
            j=max(findstr('\',CFG_logfile));
            filename=CFG_logfile(j+1:end);
            CFG_logfile=[path2file,filename]; %update log file path
        case 'CFG_profilespath',
            CFG_profilespath=path2file;
        case 'CFG_resultspath',
            CFG_resultspath=path2file; %update results path

            j=max(findstr('\',CFG_datagen_outFile));
            filename=CFG_datagen_outFile(j+1:end);
            CFG_datagen_outFile=[path2file,filename]; %update datagen output file path

            j=max(findstr('\',CFG_extract_outFile));
```

```
            filename=CFG_extract_outFile(j+1:end);
            CFG_extract_outFile=[path2file,filename]; %update extract output file path

            j=max(findstr('\',CFG_rfsim_outFile));
            filename=CFG_rfsim_outFile(j+1:end);
            CFG_rfsim_outFile=[path2file,filename]; %update rfsim output file path

            j=max(findstr('\',CFG_rxsim_outFile));
            filename=CFG_rxsim_outFile(j+1:end);
            CFG_rxsim_outFile=[path2file,filename]; %update rxsim output file path

            j=max(findstr('\',CFG_txsim_outFile));
            filename=CFG_txsim_outFile(j+1:end);
            CFG_txsim_outFile=[path2file,filename]; %update txsim output file path

            j=max(findstr('\',CFG_xswitch_outFile));
            filename=CFG_xswitch_outFile(j+1:end);
            CFG_xswitch_outFile=[path2file,filename]; %update xswitch output file path
        case 'CFG_temppath',
            CFG_temppath=path2file;
        case 'CFG_realFileREF',
            j=max(findstr('\',CFG_realFileREF));
            filename=CFG_realFileREF(j+1:end);
            CFG_realFileREF=[path2file,filename]; %update datagen output file path
        case 'CFG_realFileRX',
            j=max(findstr('\',CFG_realFileRX));
            filename=CFG_realFileRX(j+1:end);
            CFG_realFileRX=[path2file,filename]; %update datagen output file path
        otherwise
        end
    end

    %save updated profile
    OK=questdlg({...
            ['CFG_logpath = ',CFG_logpath],...
            ['CFG_logfile = ',CFG_logfile],...
            ['CFG_profilespath = ',CFG_profilespath],...
            ['CFG_resultspath = ',CFG_resultspath],...
            ['CFG_datagen_outFile = ',CFG_datagen_outFile],...
            ['CFG_extract_outFile = ',CFG_extract_outFile],...
            ['CFG_rfsim_outFile = ',CFG_rfsim_outFile],...
            ['CFG_rxsim_outFile = ',CFG_rxsim_outFile],...
            ['CFG_txsim_outFile = ',CFG_txsim_outFile],...
            ['CFG_xswitch_outFile = ',CFG_xswitch_outFile],...
            ['CFG_realFileREF = ',CFG_realFileREF],...
            ['CFG_realFileRX = ',CFG_realFileRX]},...
        'Confirm Changes?');
    if OK=='Yes'

        %clean up unwanted variables before save
```

```
            clear ans j idx filename path2file SELECTION OK pathVar temp

            %save changed profile
            save(inputProfile);
        else
            disp('Sign(s):␣␣Paths␣Left␣Unchanged');
        end

        %return successful operation
        SUCC=0;
    else
        SUCC=1;
    end
catch
    %log error condition
    signs_log(CFG_logfile,dbstack);
    SUCC=1;
end
```

SIGNS_ENGUNT

```
%Sign(s) Tool: SIGNS_ENGUNT
%
%[Y,ERR]=SIGNS_ENGUNT(X,PRF)
%
%The purpose of this function is to determine the most appropriate scaling
%(in engineering units) for an arbitrary vector.  The scaling is chosen
%based on the value 1 standard deviation away from the mean.
%
%Mandatory Input Parameters:
%   X -         input data sequence vector (real or complex)
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   PRF-        if PRF is 'prefix' then SIGNS_ENGUNT returns one of
%                  f - for femto
%                  p - for pico
%                  n - for nano
%                  u - for micro (signs_engunt returns \mu (LaTeX format))
%                  m - for milli
%                  k - for kilo
%                  M - for mega
%                  G - for giga
%                  T - for tera
%                  E - for exa
%                  P - for peta
%
%Mandatory Output Parameters:
%   Y -         averaged sequence
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   None
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%---------   -------   ------------      --------------------
function [Y,ERR]=signs_engunt(X,varargin)

try
    ERR=[];

    %compute scale factor
    Xs=abs(mean(X)+std(X));
```

```
    Xpwr=floor(log10(Xs));
    Xpwr=Xpwr-mod(Xpwr,3);
    %check to make sure there is at least one significant digit left of the decimal
    if Xpwr<-15
        Xpwr=-15;
    end
    if Xpwr>18
        Xpwr=18;
    end
    Xscale=10^Xpwr;

    %generate output
    if nargin==2
        if strcmp(varargin{1},'prefix')
            if Xpwr>-3 & Xpwr<3
                Y='';
            else
                PRF={'f','p','n','\mu','m','k','M','G','T','E','P'};
                if Xpwr<=-3
                    Y=PRF{Xpwr/3+6};
                end
                if Xpwr>=3
                    Y=PRF{Xpwr/3+5};
                end
            end
        else
            %return scaling factor because PRF is wrong
            Y=Xscale;
        end
    else
        %return scaling factor
        Y=Xscale;
    end
catch
    %capture error condition
    ERR={dbstack,lasterr};
    Y=0;
end
return
```

SIGNS_FSK

```
%Sign(s) Tool: SIGNS_FSK
%
%[Y,ERR]=SIGNS_FSK(X,R,Fc,Fs,Fd,M,TYP)
%
%SIGNS_FSK creates a continuous-phase frequency-shift-keyed modulated carrier of the specified
%peak frequency deviation and carrier frequency.
%
%Mandatory Input Parameters:
%   X - digital data stream in the form of a vector of zeros and ones
%   R - bit rate
%   Fc - carrier frequency in Hz
%   Fs - sampling frequency in Hz
%   Fd - peak frequency deviation in Hz (max distance in Hz away from centre frequency)
%
%Optional Input Parameters:
%   M - number of levels of modulation.  M=2 for 2-FSK, M=4 for 4-FSK, etc.
%       The default is M=2.
%   TYP - is an optional string argument that specifies the format
%   of the output data. It commonly contains a datatype specifier
%   like 'int' or 'float' followed by an integer giving the size in
%   bits.  Any of the following strings, either the MATLAB version,
%   or their C or Fortran equivalent, may be used.  The default type
%   is 'single'.
%        MATLAB    C or Fortran      Description
%        'uchar'   'unsigned char'   unsigned character,  8 bits.
%        'schar'   'signed char'     signed character,  8 bits.
%        'int8'    'integer*1'       integer, 8 bits.
%        'int16'   'integer*2'       integer, 16 bits.
%        'int32'   'integer*4'       integer, 32 bits.
%        'uint8'   'integer*1'       unsigned integer, 8 bits.
%        'uint16'  'integer*2'       unsigned integer, 16 bits.
%        'uint32'  'integer*4'       unsigned integer, 32 bits.
%        'single'  'real*4'          floating point, 32 bits.
%        'float32' 'real*4'          floating point, 32 bits.
%   See Help fread for more information on standard file types.
%   TYP may also be of the form intX or uintX where X is of the
%   following set {1,2,3,...,32} such that the output data
%   consists of whole numbers in the range of -2^X to 2^X-1.
%
%Mandatory Output Parameters:
%   Y - vector of the FSK signal. If an error occurs Y will be empty.
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%Special Notes:
%   * X is padded with zeros if the length of X is not a multiple of M
```

```
%    * Note that TYP does not cause SIGNS_FSK to return a vector of TYP, but rather,
%    a double precision vector containing elements restricted by TYP.  For example,
%    if TYP='int8', the output vector will consist of double precision whole numbers
%    between -128.0000000<=Y<=+127.0000000
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
%Date         Version   Editor              Changes Made
%-----------  --------  ----------------    --------------------
%
function [Y,ERR]=signs_fsk(X,R,Fc,Fs,Fd,varargin)
try
    %set defaults
    TYP='single';
    M=2;
    ERR=[];

    %check number and correct use of input arguments
    switch nargin
    case 5,
    case 6,
        if isnumeric(varargin{1})
            M=varargin{1};
        else
            TYP=varargin{1};
        end
    case 7,
        if isnumeric(varargin{1}) & ischar(varargin{2})
            M=varargin{1};
            TYP=varargin{2};
        else
            if isnumeric(varargin{2}) & ischar(varargin{1})
                M=varargin{2};
                TYP=varargin{1};
            end
        end
    end

    %transpose X to a row vector
    sz=size(X);
    if sz(1)>sz(2)
        X=X';
    end

    %pad X with zeros if its length is not a multiple of log2(M)
    if mod(length(X),log2(M))~=0
        X=[X,zeros(1,length(X)*log2(M)-length(X))];
    end
```

```
%map bits to symbols
Nbits=floor(log2(M)); %get number of bits per symbol
if Nbits>1
    %count=1;
    %for i=1:Nbits:length(X)
    %    bits=X(i:i+Nbits-1);
    %    data(count)=bits*(2.^[0:Nbits-1])';
    %    count=count+1;
    %end
    data=2.^[0:Nbits-1]*reshape(X,Nbits,floor(length(X)/Nbits));
else
    data=X;
end

%convert symbols to NRZ and upsample to sampling rate
data=signs_sgzoom(signs_nrz(data,M),Fs/R);

%determine time base
Ts=1/Fs;
T=length(data)*Ts;
t=[0:Ts:T-Ts];

%compute frequency sensitivity
h=2*Fd*Ts; %modulation index
Kf=h*pi; %frequency sensitivity

%compute phase
phase=Kf*cumsum(data);

%compute FM waveform
I=cos(phase);
Q=sin(phase);
Yi=cos(2*pi*Fc*t).*I-sin(2*pi*Fc*t).*Q;
Yi=Yi/max(abs(Yi)); %ensure signal lies between -1 and +1

%format output
switch TYP
case {'uchar','unsigned␣char','uint8','integer*1'}, %unsigned 8-bit integers
    Y=floor((2^8-1)*(Yi+1)/2);
case {'schar','signed␣char','int8','integer*1'},    %signed 8-bit integers
    Y=floor((2^8-1)*Yi/2-.5);
case {'int16','integer*2'},                         %integer, 16 bits.
    Y=floor((2^16-1)*Yi/2-.5);
case {'int32','integer*4'},                         %integer, 32 bits.
    Y=floor((2^32-1)*Yi/2-.5);
case {'uint16','integer*2'},                        %unsigned integer, 16 bits.
    Y=floor((2^16-1)*(Yi+1)/2);
case {'uint32','integer*4'},                        %unsigned integer, 32 bits.
    Y=floor((2^32-1)*(Yi+1)/2);
case {'single','real*4','float32'},                 %floating point, 32 bits.
```

```
        Y=Yi;
    otherwise
        Y=Yi; %set the default condition

        %check for non-standard int type
        if length(TYP)>=4
            if strcmp(TYP(1:3),'int') %check for signed integer
                N=str2num(TYP(4:end)); %get the number of bits
                if N>=1 & N<=32
                    Y=floor((2^N-1)*Yi/2-.5); %scale the output
                end
            elseif length(TYP)>=5 & strcmp(TYP(1:4),'uint') %check for unsigned integer
                N=str2num(TYP(5:end)); %get the number of bits
                if N>=1 & N<=32
                    Y=floor((2^N-1)*(Yi+1)/2); %scale the output
                end
            end
        end
    end
catch
    %capture error condition
    ERR={dbstack,lasterr};
    Y=[];
end
return;
```

SIGNS_GAUSSPDF

```
%Sign(s) Tool: SIGNS_GAUSSPDF
%
%[Y,ERR]=SIGNS_GAUSSPDF(X,MU,SIGMA)
%
%The purpose of this function is to returns the normal pdf with mean, MU,
%and standard deviation, SIGMA, at the values in X.
%
%Mandatory Input Parameters:
%   X -         values at which to compute the pdf
%   MU -        mean of pdf
%   SIGMA -     standard deviation of pdf
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   None
%
%Mandatory Output Parameters:
%   Y -         normal pdf
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   Function based on Matlab normpdf.m
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
%Date         Version   Editor            Changes Made
%----------   -------   ------------      --------------------
%20/5/2005    1.0       JEG               Converted normpdf.m
%                                         to signs_gausspdf.m
%normpdf.m is
%   Copyright 1993-2000 The MathWorks, Inc.
%   $Revision: 2.8 $  $Date: 2000/05/26 18:53:24 $

%   References:
%      [1]  M. Abramowitz and I. A. Stegun, "Handbook of Mathematical
%      Functions", Government Printing Office, 1964, 26.1.26.
function [Y,ERR]=gausspdf(X,MU,SIGMA)
try
    if nargin < 3,
        SIGMA = 1;
    end

    if nargin < 2;
        MU = 0;
```

```
    end

    if nargin < 1,
        error('Requires at least one input argument.');
    end

    Y = zeros(size(X));

    k = find(SIGMA > 0);
    if any(k)
        xn = (X - MU) ./ SIGMA;
        Y = exp(-0.5 * xn .^2) ./ (sqrt(2*pi) .* SIGMA);
    end

    % Return NaN if SIGMA is negative or zero.
    k1 = find(SIGMA <= 0);
    if any(k1)
        tmp    = NaN;
        Y = tmp(ones(size(k1)));
    end
    ERR=[];
catch
    %capture error condition
    ERR={dbstack,lasterr};
    y=[];
end
return
```

SIGNS_HAARWT

```
%Sign(s) Tool: SIGNS_HAARWT
%
%[C,ERR]=SIGNS_HAARWT(X,METH,PLT)
%
%The purpose of this function is to return the Haar Wavelet Transform
%of the vector X.  The Haar wavelet is defined by
%
%             { 1   a <= t < v
% H[a,b)(t) = {-1   v <= t < b   where v = (a+b)/2
%             { 0   otherwise
%
%samples of a function defined by
%
% f[k] = sum ( f(kT) x {u(t-kT)-u(t-(k+1))} )
%
%can be represented by the sum and difference of
%
%        f(kT)+f((k+1)T)                 f(kT)-f((k+1)T)
% f[k] = ---------------- U[a,b)(t) + ---------------- H[a,b)(t)
%               2                               2
%
%                  { 1   a <= t < b
%where U[a,b)(t) = {
%                  { 0   otherwise
%
%consequently, the Haar Wavelet Transform works on consecutive pairs
%of samples.
%
%Mandatory Input Parameters:
%   X -          arbitrary numeric vector of length 2^N.  If the length of
%                X is not a power of 2, it will be padded with zeros to
%                reach the next nearest power of 2 length.
%   PLT -        set to 1 to display a graphical version of the wavelet transform
%                the default is 0
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   METH -       'ordered' for the Ordered Fast Haar Wavelet Transform
%                'inplace' for the In-place Fast Haar Wavelet Transform (default)
%
%Mandatory Output Parameters:
%   C -          the Haar Wavelet Transform of length 2^N.
%
%                Ordered Fast Haar Wavelet Transform:
%                    The first element in the vector will be the mean of all
%                    elements of X.  The second element will be the lowest frequency
%                    wavelet coefficient, the next two elements will be the coefficients
%                    for the next highest frequency wavelets, the next four elements will
%                    be the coefficients for the next highest frequency wavelets, and so
```

```
%                   on.  For example, if X=[3,1,9,7,7,9,5,7] then the Ordered Fast Haar
%                   Wavelet Transform C=[6,-1,-3,1,1,1,-1,-1].  Given C, X can be written as
%                   X = 6U[0,1) - 1H[0,1) - 3H[0,1/2) + 1H[1/2,1) + 1H[0,1/4)
%                   + 1H[1/4,1/2)-1H[1/2,3/4)-1H[3/4,1).
%
%               In-place Fast Haar Wavelet Transform:
%                   The first element in the vector will be the mean of all
%                   elements of X.  The second element will be the first of the highest
%                   frequency wavelets, the third element will be the first of the 2nd
%                   highest frequency wavelets and so on as in the example below.
%                   For example, if X=[3,1,9,7,7,9,5,7] then the In-place Fast Haar
%                   Wavelet Transform C=[6,1,-3,1,-1,-1,1,-1].  Given C, X can be written as
%                   X = 6U[0,1) + 1H[0,1/4) - 3H[0,1/2) + 1H[1/4,1/2) - 1H[0,1)
%                   - 1H[1/2,3/4) + 1H[1/2,1) - 1H[3/4,1).  Notice that though the in-place
%                   and ordered transforms contain the same information, the ordering of the
%                   wavelet coefficients differ.
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   Reference: Nievergelt, Yves; "Wavelets Made Easy", Birkhauser, 1999.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%---------   -------   ------------      --------------------

function [C,ERR]=signs_haarwt(X,varargin)
try
    %determine the method
    METH='inplace';
    PLT=0;
    if nargin>1
        for j=1:max(length(varargin),2)
            if ~isnumeric(varargin{j})
                METH=varargin{j};
            else
                PLT=varargin{j};
            end
        end
    end

    %get the input vector and reshape to row vector
    sz=size(X);
    S=reshape(X,1,length(X));
```

```
%ensure the input vector has a length that is a power of 2
if mod(log2(length(S)),1)~=0
    N=nextpow2(length(S)); %get next power of two
    S=[S,zeros(1,2^N-length(S))]; %pad the input vector
else
    N=fix(log2(length(S)));
end

%compute the transform with the correct method
switch METH,

%Ordered Fast Haar Wavelet Transform
case 'ordered',
    A=S;
    C=zeros(1,length(S)); %prepare array for wavelet coefficients
    idx=2^N; %initialize index for storage of wavelet coefficients
    for j=1:N
        %number of wavelet coefficients for this sweep
        Nc=2^(N-j);

        %compute and store wavelet coefficients
        C(idx-Nc+1:idx)=-diff(reshape(A,2,length(A)/2))/2;

        %update index
        idx=idx-Nc;

        %create next set of pairs
        A=sum(reshape(A,2,length(A)/2))/2;
    end
    C(1)=A; %store the last value (mean of all samples)

%In-place Fast Haar Wavelet Transform
case 'inplace',
    C=S; %prepare array for wavelet coefficients
    for j=1:N
        %compute step
        stp=2^(j-1);

        %create pairs
        A=reshape(C(1:stp:end),2,2^(N-j));

        %compute and store wavelet coefficients
        C(stp+1:2*stp:end)=-diff(A)/2;

        %compute and store next set of averages
        C(1:2*stp:end)=sum(A)/2;
    end
end

%plot the transform if necessary
```

```
    if PLT
        figure
        bar(C);
        title([METH,' Haar Wavelet Transform'])
        xlabel('Wavelet Frequency Index')
        ylabel('Wavelet Coefficients')
    end

    %shape the output vector to a row or column vector
    if sz(1)>sz(2)
        C=reshape(C,length(C),1);
    end
    ERR=[];
catch
    %capture error condition
    ERR={dbstack,lasterr};
    C=0;
end
return
```

SIGNS_HAARWT2

```
%Sign(s) Tool: SIGNS_HAARWT2
%
%[C,ERR]=SIGNS_HAARWT2(X,METH,PLT)
%
%The purpose of this function is to return the two-dimensional
%Haar Wavelet Transform of the array X.  The Haar wavelet is defined by
%
%              { 1   a <= t < v
% H[a,b)(t) = {-1   v <= t < b   where v = (a+b)/2
%              { 0   otherwise
%
%samples of a function defined by
%
% f[k] = sum ( f(kT) x {u(t-kT)-u(t-(k+1))} )
%
%can be represented by the sum and difference of
%
%         f(kT)+f((k+1)T)                  f(kT)-f((k+1)T)
% f[k] = ---------------- U[a,b)(t) +  ---------------- H[a,b)(t)
%                2                                2
%
%                    { 1   a <= t < b
%where U[a,b)(t) = {
%                    { 0   otherwise
%
%consequently, the Haar Wavelet Transform works on consecutive pairs
%of samples.
%
%Mandatory Input Parameters:
%   X -          arbitrary numeric square array of size 2^N x 2^N.  If the size of
%                X is not a power of 2, it will be padded with zeros to
%                reach the next nearest power of 2 length.
%   PLT -        set to 1 to display a graphical version of the wavelet transform
%                the default is 0
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   METH -       'ordered' for the Ordered Fast Haar Wavelet Transform
%                'inplace' for the In-place Fast Haar Wavelet Transform (default)
%
%Mandatory Output Parameters:
%   C -          the 2D Haar Wavelet Transform of size 2^N x 2^N.
%
%                Ordered Fast Haar Wavelet Transform:
%                    The first element in the vector will be the mean of all
%                    elements of X.  All subsequent elements represent horizontal,
%                    vertical, and diagonal "jumps" in the data.  For example,
%                         X =  9 7
%                              5 3
```

```
%                   then the ordered 2D transform works by applying the ordinary
%                   ordered 1D transform to the rows and then to the columns yielding
%                        Rows of X :  C'= (9+7)/2 (9-7)/2 = 8 1
%                                         (5+3)/2 (5-3)/2   4 1
%
%                     Columns of C':  C"= (8+4)/2 (1+1)/2 = 6 1
%                                         (8-4)/2 (1-1)/2   2 0
%
%                   and using the wavelet functions and given C=C", X can be written as
%                   X = 6Ux[0,1)*Uy[0,1) + 1Ux[0,1)*Hy[0,1) + 2Hx[0,1)*Uy[0,1) + 0Hx[0,1)*Hy
    [0,1).
%
%               In-place Fast Haar Wavelet Transform:
%                   The first element in the vector will be the mean of all
%                   elements of X.  All subsequent elements represent horizontal,
%                   vertical, and diagonal "jumps" in the data.  For example,
%                        X =  9 7 9 7
%                             5 3 5 3
%                             9 7 9 7
%                             5 3 5 3
%                   then the in-place 2D transform works by repeatedly applying the ordinary
%                   inplace 1D transform to the rows and then to the columns yielding
%
%                   SWEEP#1
%                        Rows of X :  C'= (9+7)/2 (9-7)/2 (9+7)/2 (9-7)/2 = 8 1 8 1
%                                         (5+3)/2 (5-3)/2 (5+3)/2 (5-3)/2   4 1 4 1
%                                         (9+7)/2 (9-7)/2 (9+7)/2 (9-7)/2   8 1 8 1
%                                         (5+3)/2 (5-3)/2 (5+3)/2 (5-3)/2   4 1 4 1
%                     Columns of C':  C"= 6 1 6 1
%                                         2 0 2 0
%                                         6 1 6 1
%                                         2 0 2 0
%
%                   SWEEP#2
%                        Rows of C" :  C"'= (6+6)/2 1 (6-6)/2 1 = 6 1 0 1
%                                             2     0     2     0   2 0 2 0
%                                           (6+6)/2 1 (6-6)/2 1   6 1 0 1
%                                             2     0     2     0   2 0 2 0
%                     Columns of C':  C""= 6 1 0 1
%                                         2 0 2 0
%                                         0 1 0 1
%                                         2 0 2 0
%
%                   and using the wavelet functions and given C=C"", X can be written as
%                   X = 6Ux[0,1)*Uy[0,1) + 0Ux[0,1)*Hy[0,1) + 1Ux[0,1)*Hy[0,1/2) + 1Ux[0,1)*Hy
    [1/2,1) +
%                       2Hx[0,1/2)*Uy[0,1) + 0*Hx[0,1/2)*Hy[0,1/2) + ...
%
%                   The in-place and ordered transforms contain the same information, only the
     ordering of the
```

```
%                   wavelet coefficients differ.
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   Reference: Nievergelt, Yves; "Wavelets Made Easy", Birkhauser, 1999.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date         Version   Editor            Changes Made
%----------   -------   ------------      --------------------

function [C,ERR]=signs_haarwt2(X,varargin)
try
    %determine the method
    METH='inplace';
    PLT=0;
    if nargin>1
        for j=1:length(varargin)
            if ~isnumeric(varargin{j})
                METH=varargin{j};
            else
                PLT=varargin{j};
            end
        end
    end

    %get the input array
    sz=size(X);
    S=X;

    %ensure the array size is a power of 2
    if (mod(log2(sz(1)),1)~=0) | (mod(log2(sz(2)),1)~=0)
        Ny=nextpow2(sz(2)); %get next power of two for number of columns
        Nx=nextpow2(sz(1)); %get next power of two for number of rows
        N=max(Nx,Ny); %get padded array order
        S=zeros(2^N); %initialize padded array
        S(1:sz(1),1:sz(2))=X; %insert original array
    else
        N=fix(log2(max(sz))); %get array order
    end

    %compute the transform with the correct method
    switch METH,

    %Ordered Fast Haar Wavelet Transform
```

```
case 'ordered',
    A=S;
    C=zeros(2^N); %prepare array for wavelet coefficients

    %do the transform
    for j=0:N-1
        %number of wavelet coefficients for this sweep
        Nc=2^(N-j);

        %initialize temporary storage
        temp=zeros(Nc);

        %compute and store row wavelet coefficients
        for k=1:Nc
            temp(k,1:2:end)=sum(reshape(A(k,:),2,Nc/2))/2;
            temp(k,2:2:end)=-diff(reshape(A(k,:),2,Nc/2))/2;
        end

        %compute and store column wavelet coefficients
        A=temp;
        for k=1:Nc
            temp(1:2:end,k)=sum(reshape(A(:,k),2,Nc/2)',2)/2;
            temp(2:2:end,k)=-diff(reshape(A(:,k),2,Nc/2)',1,2)/2;
        end

        %extract 2-D wavelet coefficients and rearrange
        PHI00=temp(1:2:end,1:2:end);
        PHI01=temp(1:2:end,2:2:end);
        PHI10=temp(2:2:end,1:2:end);
        PHI11=temp(2:2:end,2:2:end);
        A=PHI00;

        %reorder coefficient array
        C(1:Nc,1:Nc)=[PHI00,PHI01;PHI10,PHI11];
    end

%In-place Fast Haar Wavelet Transform
case 'inplace',
    C=S; %prepare inplace array for wavelet coefficients

    %do the transform
    for j=0:N-1
        %number of wavelet coefficients for this sweep
        Nc=2^(N-j);

        %initialize temporary storage
        temp=zeros(Nc);

        %initialize indexing for rows and columns
        m=2^j; %get slope of mapping function (i.e. y=mx+b where x=(k-1) and b=1)
```

```
            %compute and store row wavelet coefficients
            for k=1:Nc
                y=m*(k-1)+1; %compute row index
                PHI=reshape(C(y,1:m:end),2,Nc/2); %get next set of elements
                temp(k,1:2:end)=sum(PHI)/2;
                temp(k,2:2:end)=-diff(PHI)/2;
            end

            %update computation array
            C(1:m:end,1:m:end)=temp;

            %compute and store column wavelet coefficients
            for k=1:Nc
                y=m*(k-1)+1; %compute column index
                PHI=reshape(C(1:m:end,y),2,Nc/2)'; %get next set of elements
                temp(1:2:end,k)=sum(PHI,2)/2;
                temp(2:2:end,k)=-diff(PHI,1,2)/2;
            end

            %update wavelet array
            C(1:m:end,1:m:end)=temp;
        end
    end

    %plot the transform if necessary
    if PLT
        figure
        surf(C)
        shading interp
        colorbar
        title([METH,' Haar Wavelet Transform'])
        xlabel('Wavelet Row Index')
        ylabel('Wavelet Column Index')
    end

    ERR=[];
catch
    %capture error condition
    ERR={dbstack,lasterr};
    C=0;
end
return
```

SIGNS_HAMDIST

```
%Sign(s) Tool: SIGNS_HAMDIST
%
%[S,ERR]=SIGNS_HAMDIST(CS,RS)
%
%SIGNS_HAMDIST returns the hamming distance between two binary sequences.
%
%Mandatory Input Parameters:
%   CS - compare data sequence (a vector of 1s and 0s).  Any non-zero
%        data element is considered a binary 1.
%   RS - reference data sequences (a vector of 1s and 0s).  Any non-zero
%        data element is considered a binary 1.
%
%Optional Input Parameters:
%   None
%
%Mandatory Output Parameters:
%   HD - scalar representing the number of differing data elements
%        between CS and RS. If the length of RS > CS or the length
%        of CS > RS then the hamming distance is increased by the
%        difference between the lengths of RS and CS because it is
%        assumed that the shorter sequence is different from the
%        longer by number of unmatched data elements.
%
%Optional Output Parameters:
%   None
%
%Special Notes:
%
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
%Date         Version   Editor            Changes Made
%---------    -------   ------------      --------------------
%19/01/06     1.1       JEG               Incorporated into Sign(s)
function [S,ERR]=signs_hamdist(CS,RS)

ERR=[];
try
  S=sum(xor(CS,RS));
catch
    %capture error condition
    ERR={dbstack,lasterr};
    S=[];
end
return
```

SIGNS_HFNOISE

```
%Sign(s) Tool: SIGNS_HFNOISE
%
%[S,ERR]=SIGNS_HFNOISE(MODEL,CAT,F,BW)
%
%The purpose of this function is to compute the level of the HF environmental
%noise based on a particular noise model.
%
%Mandatory Input Parameters:
%   MODEL - the noise model to use: either 'ccir' or 'ausstd'.
%   CAT -   the noise model category.  For 'ccir' the categories are:
%
%               1 or 'business' for a industrial environment
%               2 or 'residential' for a residential environment
%               3 or 'rural' for a rural environment
%               4 or 'remote' for a remote environment
%
%           The categories for the Australian standard (i.e. 'ausstd') are:
%
%               1 - not implemented
%               2 - not implemented
%               3 - not implemented
%               4 - not implemented
%               5 or 'remote' for a remote Australian site.
%
%   F -     is the frequency (MHz) at which the noise level is to be computed.
%           F can be a vector of frequencies.
%
%Optional Input Parameters:
%   BW -    the bandwidth (Hz) over which the noise is to be computed.  The
%           default bandwidth is 1 Hz.
%
%Mandatory Output Parameters:
%   S -     contains HF noise level (dB) at the specific frequency, F, and bandwidth, BW,
%           if BW is specified.  S is empty if a noise level cannot be determined.
%           If BW is not specified, S defaults to the noise level in a 1 Hz bandwidth.
%
%Optional Output Parameters:
%   ERR -   two element cell array.  The first element is a structure array
%           containing the debug trace information, the second element is the
%           error.
%
%Special Notes:
%
%Copyright (c) 2005 James Giesbrecht

%Company:
%
%Revision History:
```

```
%Date        Version   Editor          Changes Made
%----------  --------  --------------  --------------------
%04/12/06    1.01      JEG             added "lower" to switch statement on the MODEL variable
%
function [S,ERR]=signs_hfnoise(MODEL,CAT,F,varargin)

ERR=[];
try
    if nargin>=4 & ~ischar(varargin{1})
        BW=varargin{1}; %get user specified bandwidth
    else
        BW=1; %default bandwidth is 1 Hz.
    end
    switch lower(MODEL)
    case 'ccir',
        switch lower(CAT)
        case {1,'business'},
            G=27.7;
            N1=-127.2;
            S=N1+G*log10(F)+10*log10(BW);
        case {2,'residential'},
            G=27.7;
            N1=-131.5;
            S=N1+G*log10(F)+10*log10(BW);
        case {3,'rural'},
            G=27.7;
            N1=-136.8;
            S=N1+G*log10(F)+10*log10(BW);
        case {4,'remote'},
            G=27.7;
            N1=-150.4;
            S=N1+G*log10(F)+10*log10(BW);
        otherwise
            S=[];
        end
    case 'ausstd',
        switch lower(CAT)
        case {1,'business'},
        case {2,'residential'},
        case {3,'rural'},
        case {4,'remote'},
        case {5,'remote'},
            G=1;
            N1=-168.5;
            S=N1+20*G*log10(F)+10*log10(BW);
        otherwise
            S=[];
        end
    otherwise
        S=[];
```

```
    end
catch
    %capture error condition
    ERR={dbstack,lasterr};
    S=[];
end
return
```

SIGNS_HYPER2F1

```
%Sign(s) Tool: SIGNS_HYPER2F1
%
%[Y,ERR]=SIGNS_HYPER2F1(A,B,C,Z,N)
%
%The purpose of this function is to return the hypergeometric function given
%parameters A, B, C, and the variable Z.  The hypergeometric function is defined as
%
%                              N  [ (A)n (B)n     Z^n
% 2F1(A,B,C,Z) = lim     SUM   [------------ x ----
%                 N->inf    n=0 [    (C)n         n!
%
%where (.)n is the Pochhammer symbol for the rising factorial.
%
%Mandatory Input Parameters:
%   A -      parameter A for the hypergeometric function
%   B -      parameter B for the hypergeometric function
%   C        parameter C for the hypergeometric function
%   Z -      a numeric vector containing the values of the variable Z
%
%Optional Input Parameters:
%   N -      the number interations for each coefficient (the default is 10)
%
%Mandatory Output Parameters:
%   Y -      the hypergeometric function
%
%Optional Output Parameters:
%   ERR -    two element cell array.  The first element is a structure array
%            containing the debug trace information, the second element is the
%            error.
%
%Special Notes:
%   None
%
%Copyright (c) 2007 James Giesbrecht
function [Y,ERR]=signs_hyper2f1(a,b,c,z,varargin)
try
    %set the number of coefficients to calculate for each element of z
    N=10; %default
    if nargin==5
        if isnumeric(varargin{1})
            N=varargin{1};
        end
    end

  %compute the hypergeometric coefficients
    for n=1:N
        alpha(n)=prod(a:a+n-1)*prod(b:b+n-1)/prod(c:c+n-1)/factorial(n);
    end
```

```
    Y=zeros(size(z,1),size(z,2));
    for j=1:length(z)
        Y(j)=1+sum(alpha.*z(j).^[1:N]);
    end
    ERR=[];
catch
    %capture error condition
    ERR={dbstack,lasterr};
    Y=[];
end
return
```

SIGNS_IHAARWT

```
%Sign(s) Tool: SIGNS_IHAARWT
%
%[X,ERR]=SIGNS_IHAARWT(C,METH,PLT)
%
%The purpose of this function is to return the inverse Haar Wavelet Transform
%of the vector C.  The Haar wavelet is defined by
%
%              { 1   a <= t < v
% H[a,b)(t) = {-1   v <= t < b   where v = (a+b)/2
%              { 0   otherwise
%
%samples of a function defined by
%
% f[k] = sum ( f(kT) x {u(t-kT)-u(t-(k+1))} )
%
%can be represented by the sum and difference of
%
%         f(kT)+f((k+1)T)                  f(kT)-f((k+1)T)
% f[k] = ---------------- U[a,b)(t) +  ---------------- H[a,b)(t)
%                2                                2
%
%                    { 1   a <= t < b
%where U[a,b)(t) = {
%                    { 0   otherwise
%
%consequently, the inverse Haar Wavelet Transform works on consecutive pairs
%of samples.
%
%Mandatory Input Parameters:
%   C -          arbitrary numeric vector of length 2^N representing the Haar
%                Wavelet Transform of X.  If the length of
%                C is not a power of 2, it will be padded with zeros to
%                reach the next nearest power of 2 length.
%
%                Ordered Fast Haar Wavelet Transform:
%                    The first element in the vector will be the mean of all
%                    elements of X.  The second element will be the lowest frequency
%                    wavelet coefficient, the next two elements will be the coefficients
%                    for the next highest frequency wavelets, the next four elements will
%                    be the coefficients for the next highest frequency wavelets, and so
%                    on.  For example, if X=[3,1,9,7,7,9,5,7] then the Ordered Fast Haar
%                    Wavelet Transform C=[6,-1,-3,1,1,1,-1,-1].  Given C, X can be written as
%                    X = 6U[0,1) - 1H[0,1) - 3H[0,1/2) + 1H[1/2,1) + 1H[0,1/4)
%                    + 1H[1/4,1/2)-1H[1/2,3/4)-1H[3/4,1).
%
%                In-place Fast Haar Wavelet Transform:
%                    The first element in the vector will be the mean of all
%                    elements of X.  The second element will be the first of the highest
```

```
%                   frequency wavelets, the third element will be the first of the 2nd
%                   highest frequency wavelets and so on as in the example below.
%                   For example, if X=[3,1,9,7,7,9,5,7] then the In-place Fast Haar
%                   Wavelet Transform C=[6,1,-3,1,-1,-1,1,-1]. Given C, X can be written as
%                   X = 6U[0,1) + 1H[0,1/4) - 3H[0,1/2) + 1H[1/4,1/2) - 1H[0,1)
%                   - 1H[1/2,3/4) + 1H[1/2,1) - 1H[3/4,1). Notice that though the in-place
%                   and ordered transforms contain the same information, the ordering of the
%                   wavelet coefficients differ.
%
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   METH -      'ordered' for the Ordered Fast Haar Wavelet Transform
%               'inplace' for the In-place Fast Haar Wavelet Transform (default)
%   PLT -       set to 1 to display a graphical version of the inverse wavelet transform
%               the default is 0
%
%Mandatory Output Parameters:
%   X -         the original vector (of length 2^N) prior to the forward Haar Wavelet
    Transform.
%
%Optional Output Parameters:
%   ERR -       two element cell array. The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   Reference: Nievergelt, Yves; "Wavelets Made Easy", Birkhauser, 1999.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor          Changes Made
%-----------  -------  -------------   --------------------

function [X,ERR]=signs_ihaarwt(C,varargin)
try
    %determine the method
    METH='inplace';
    PLT=0;
    if nargin>1
        for j=1:max(length(varargin),2)
            if ~isnumeric(varargin{j})
                METH=varargin{j};
            else
                PLT=varargin{j};
            end
        end
    end

    %get the input vector and reshape to row vector
```

```
sz=size(C);
S=reshape(C,1,length(C));

%ensure the input vector has a length that is a power of 2
if mod(log2(length(S)),1)~=0
    N=nextpow2(length(S)); %get next power of two
    S=[S,zeros(1,2^N-length(S))]; %pad the input vector
else
    N=fix(log2(length(S)));
end

%compute the transform with the correct method
switch METH,

%Inverse Ordered Fast Haar Wavelet Transform
case 'ordered',
    X=zeros(1,length(S)); %prepare array original samples
    for j=N:-1:1
        %number of wavelet coefficients for this sweep
        Nc=2^(N-j);

        %get wavelet coefficients
        C=S(Nc+1:2*Nc);

        %get averages
        A=S(1:Nc);

        %compute and store original elements
        X(1:2:2*Nc)=A+C;
        X(2:2:1+2*Nc)=A-C;

        %update S
        S=[X(1:2*Nc),S(2*Nc+1:end)];

    end
    X=S; %store the last value (mean of all samples)

%Inverse In-place Fast Haar Wavelet Transform
case 'inplace',
    X=S;
    for j=N:-1:1
        %compute step
        stp=2^(j-1);

        %create pairs
        A=reshape(X(1:stp:end),2,2^(N-j));

        %compute and store samples
        X(1:2*stp:end)=sum(A);
        X(stp+1:2*stp:end)=-diff(A);
```

```
        end
    end

    %plot the inverse transform if necessary
    if PLT
        figure
        plot(X,'bo-');
        title(['Inverse␣',METH,'␣Haar␣Wavelet␣Transform'])
        xlabel('Time␣Index')
        ylabel('Sample␣Magnitude')
    end

    %shape the output vector to a row or column vector
    if sz(1)>sz(2)
        X=reshape(X,length(X),1);
    end
    ERR=[];
catch
    %capture error condition
    ERR={dbstack,lasterr};
    X=0;
end
return
```

SIGNS_IHAARWT2

```
%Sign(s) Tool: SIGNS_IHAARWT
%
%[X,ERR]=SIGNS_IHAARWT(C,METH,PLT)
%
%The purpose of this function is to return the inverse Haar Wavelet Transform
%of the vector C.  The Haar wavelet is defined by
%
%              { 1   a <= t < v
% H[a,b)(t) = {-1   v <= t < b   where v = (a+b)/2
%              { 0   otherwise
%
%samples of a function defined by
%
% f[k] = sum ( f(kT) x {u(t-kT)-u(t-(k+1))} )
%
%can be represented by the sum and difference of
%
%         f(kT)+f((k+1)T)                  f(kT)-f((k+1)T)
% f[k] = ---------------- U[a,b)(t) +  ---------------- H[a,b)(t)
%                2                                2
%
%                    { 1   a <= t < b
%where U[a,b)(t) = {
%                    { 0   otherwise
%
%consequently, the inverse Haar Wavelet Transform works on consecutive pairs
%of samples.
%
%Mandatory Input Parameters:
%   C -          arbitrary numeric square array of size 2^N x 2^N.  If the size of
%                C is not a power of 2, it will be padded with zeros to
%                reach the next nearest power of 2 size.
%
%                Ordered Fast Haar Wavelet Transform:
%                    The first element in the vector will be the mean of all
%                    elements of X.  All subsequent elements represent horizontal,
%                    vertical, and diagonal "jumps" in the data.  For example,
%                         X =  9 7
%                              5 3
%                    then the ordered 2D transform works by applying the ordinary
%                    ordered 1D transform to the rows and then to the columns yielding
%                         Rows of X :  C'= (9+7)/2 (9-7)/2 = 8 1
%                                          (5+3)/2 (5-3)/2   4 1
%
%                       Columns of C':  C"= (8+4)/2 (1+1)/2 = 6 1
%                                          (8-4)/2 (1-1)/2   2 0
%
%                    and using the wavelet functions and given C=C", X can be written as
```

```
%                   X = 6Ux[0,1)*Uy[0,1) + 1Ux[0,1)*Hy[0,1) + 2Hx[0,1)*Uy[0,1) + 0Hx[0,1)*Hy
    [0,1).
%
%               In-place Fast Haar Wavelet Transform:
%                   The first element in the vector will be the mean of all
%                   elements of X.  All subsequent elements represent horizontal,
%                   vertical, and diagonal "jumps" in the data.  For example,
%                       X =  9 7 9 7
%                            5 3 5 3
%                            9 7 9 7
%                            5 3 5 3
%                   then the in-place 2D transform works by repeatedly applying the ordinary
%                   inplace 1D transform to the rows and then to the columns yielding
%
%                   SWEEP#1
%                       Rows of X :  C'= (9+7)/2 (9-7)/2 (9+7)/2 (9-7)/2 = 8 1 8 1
%                                        (5+3)/2 (5-3)/2 (5+3)/2 (5-3)/2   4 1 4 1
%                                        (9+7)/2 (9-7)/2 (9+7)/2 (9-7)/2   8 1 8 1
%                                        (5+3)/2 (5-3)/2 (5+3)/2 (5-3)/2   4 1 4 1
%                     Columns of C':  C"= 6 1 6 1
%                                         2 0 2 0
%                                         6 1 6 1
%                                         2 0 2 0
%
%                   SWEEP#2
%                       Rows of C" :  C"'= (6+6)/2 1 (6-6)/2 1 = 6 1 0 1
%                                            2     0    2     0   2 0 2 0
%                                        (6+6)/2 1 (6-6)/2 1   6 1 0 1
%                                            2     0    2     0   2 0 2 0
%                     Columns of C':  C""= 6 1 0 1
%                                          2 0 2 0
%                                          0 1 0 1
%                                          2 0 2 0
%
%                   and using the wavelet functions and given C=C"", X can be written as
%                   X = 6Ux[0,1)*Uy[0,1) + 0Ux[0,1)*Hy[0,1) + 1Ux[0,1)*Hy[0,1/2) + 1Ux[0,1)*Hy
    [1/2,1) +
%                       2Hx[0,1/2)*Uy[0,1) + 0*Hx[0,1/2)*Hy[0,1/2) + ...
%
%                   The in-place and ordered transforms contain the same information, only the
     ordering of the
%                   wavelet coefficients differ.
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   METH -      'ordered' for the Ordered Fast Haar Wavelet Transform
%               'inplace' for the In-place Fast Haar Wavelet Transform (default)
%   PLT -       set to 1 to display a graphical version of the inverse wavelet transform
%               the default is 0
%
%Mandatory Output Parameters:
```

```
%   X -         the original vector (of 2^N x 2^N) prior to the forward Haar Wavelet Transform
    .
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   Reference: Nievergelt, Yves; "Wavelets Made Easy", Birkhauser, 1999.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%----------  --------  ----------------  --------------------

function [X,ERR]=signs_ihaarwt(C,varargin)
try
    %determine the method
    METH='inplace';
    PLT=0;
    if nargin>1
        for j=1:length(varargin)
            if ~isnumeric(varargin{j})
                METH=varargin{j};
            else
                PLT=varargin{j};
            end
        end
    end

    %get the input array
    sz=size(C);
    S=C;

    %ensure the array size is a power of 2
    if (mod(log2(sz(1)),1)~=0) | (mod(log2(sz(2)),1)~=0)
        Ny=nextpow2(sz(2)); %get next power of two for number of columns
        Nx=nextpow2(sz(1)); %get next power of two for number of rows
        N=max(Nx,Ny); %get padded array order
        S=zeros(2^N); %initialize padded array
        S(1:sz(1),1:sz(2))=C; %insert original array
    else
        N=fix(log2(max(sz))); %get array order
    end

    %compute the transform with the correct method
    switch METH,
```

```
%Inverse Ordered Fast Haar Wavelet Transform
case 'ordered',
    A=S;

    %prepare arrays for basic inverse 2D wavelet transform
    PHI00=[1,1;1,1];
    PHI01=[1,-1;1,-1];
    PHI10=[1,1;-1,-1];
    PHI11=[1,-1;-1,1];

    %do the inverse ordered wavelet transform
    for j=1:N
        %number of wavelet coefficients for this sweep
        Nc=2^j;

        %rearrange A
        temp=A(1:Nc,1:Nc);
        A(1:2:Nc,1:2:Nc)=temp(1:Nc/2,1:Nc/2);
        A(1:2:Nc,2:2:Nc)=temp(1:Nc/2,Nc/2+1:Nc);
        A(2:2:Nc,1:2:Nc)=temp(Nc/2+1:Nc,1:Nc/2);
        A(2:2:Nc,2:2:Nc)=temp(Nc/2+1:Nc,Nc/2+1:Nc);

        %compute and store row/column 2x2 nverse wavelet coefficients
        for rows=1:2:Nc
            for cols=1:2:Nc
                %get next 2x2 array
                temp=A(rows:rows+1,cols:cols+1);

                %compute inverse for 2x2 arrays along row and column dimensions
                A(rows:rows+1,cols:cols+1)=temp(1,1)*PHI00+temp(1,2)*PHI01+temp(2,1)*PHI10
                    +temp(2,2)*PHI11;
            end
        end
    end
    X=A;

%Inverse In-place Fast Haar Wavelet Transform
case 'inplace',
    X=S;

    %prepare arrays for basic inverse 2D wavelet transform
    PHI00=[1,1;1,1];
    PHI01=[1,-1;1,-1];
    PHI10=[1,1;-1,-1];
    PHI11=[1,-1;-1,1];

    %do the inverse ordered wavelet transform
    for j=1:N
        %number of wavelet coefficients for this sweep
        Nc=2^j;
```

```
            stp=2^(N-j);

            %extract inplace wavelet coefficients
            A=X(1:stp:end,1:stp:end);

            %compute and store row/column 2x2 inverse wavelet coefficients
            for rows=1:2:Nc
                for cols=1:2:Nc
                    %get next 2x2 array
                    temp=A(rows:rows+1,cols:cols+1);

                    %compute inverse for 2x2 arrays along row and column dimensions
                    A(rows:rows+1,cols:cols+1)=temp(1,1)*PHI00+temp(1,2)*PHI01+temp(2,1)*PHI10
                        +temp(2,2)*PHI11;
                end
            end

            %update data array
            X(1:stp:end,1:stp:end)=A;
        end
    end

    %plot the inverse transform if necessary
    if PLT
        figure
        surf(X)
        shading interp
        colorbar
        title(['Inverse ',METH,' Haar Wavelet Transform'])
        xlabel('Sample Index')
        ylabel('Sample Magnitude')
    end

    ERR=[];
catch
    %capture error condition
    ERR={dbstack,lasterr};
    X=0;
end
return
```

SIGNS_ISDYADIC

```
%Sign(s) Tool: SIGNS_ISDYADIC
%
%[D,ERR]=SIGNS_ISDYADIC(X)
%
%The purpose of this function is to test the dyadic nature of an array.
%A number is dyadic if it is an integral multiple of an integral power of 2.
%For example, 3/2 is a dyadic number because it can be written as 3x2^-1.
%Consequently a dyadic number, X, must be factorable into
%
% X = m * 2^-n = p/q * 2^s
%
%where m, n, and s are integers. p and q are ideally integers with q being a
%power of 2.  If p or q are not integers or if p and q are integers but
%q is not a power of 2 then the number X is not dyadic.
%
%Mandatory Input Parameters:
%   X -         Real numeric array for which 2^-32 <= |X| <= 2^32.
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   None
%
%Mandatory Output Parameters:
%   D -         true if all elements in the array X are dyadic, false otherwise
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   Reference: Nievergelt, Yves; "Wavelets Made Easy", Birkhauser, 1999.
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%-----------  -------  ----------------  ---------------------

function [D,ERR]=signs_isdyadic(X)
try
    %limit maximum value
    MAXINT=2^32;

    %find all non-zero X values (since by definition zero is dyadic)
    X=X(find(X~=0));

    if ~isempty(X)
        %find all X values that are not integers (since by definition an
```

```
%integer can be written as K*2^0 where K is the integer)
X=X(find(abs(X)~=fix(abs(X))));

if ~isempty(X)
    %find all X values that are not a power of 2 (since by definition
    %all powers of 2 are dyadic (e.g. 2^-1, 2^5, 2^-28, 2^16))
    X=X(find(log2(abs(X))~=fix(log2(abs(X)))));

    if ~isempty(X)
        %get the fractional part p/q and exponent s
        [p_by_q,s]=log2(abs(X)); %p/q and s

        %initialize p and q and vector of signs
        q=zeros(1,length(p_by_q));
        p=zeros(1,length(p_by_q));

        %find non-integer ratios (an integer implies that q=2^0)
        idx_rat=find(p_by_q-fix(p_by_q)~=0);

        %compute q
        q=2.^ceil(max(abs(log2(p_by_q(idx_rat)))));
        k=p_by_q.*q;
        while ~isequal(k(idx_rat),fix(k(idx_rat)))
            q=q*2.^nextpow2(k);
            k=p_by_q(idx_rat)*q;
            if q>MAXINT
                q=MAXINT;
                break;
            end
        end

        %compute p
        p=p_by_q.*q;

        %compute m and n
        if p==fix(p) & q==fix(q)
  m=p/min(gcd(p,q));
  n=-(s-log2(q));

  %test for membership in the set of all integers
  if m==fix(m) & n==fix(n)%if p==fix(p) & q==fix(q) & log2(q)==fix(log2(q))
                D=1;
  else
                D=0;
  end
else
  D=0;
end
    else
        D=1;
```

```
            end
        else
            D=1;
        end
    else
        D=1;
    end
catch
    %capture error condition
    ERR={dbstack,lasterr};
    D=0;
end
return
```

SIGNS_LOG

```
%Sign(s) Tool: SIGNS_LOG
%
%SUCC=SIGNS_LOG(LOGFILE,ENTRY1,ENTRY2,...)
%
%The purpose of this function is to read or write a log file with specific
%entries.  SIGNS_LOG(LOGFILE) displays the contents of the logfile.
%SIGNS_LOGFILE(LOGFILE,ENTRY1,...) inserts one or more entries into the logfile.
%
%Mandatory Input Parameters:
%   LOGFILE -   filename and path of the log file (.txt).  If LOGFILE='stdout'
%               then the entry is written to the screen.
%
%Optional Input Parameters:
%   ENTRYx -    string for entry into logfile. If ENTRYx is the two-element
%               structure returned by the Matlab DBSTACK function then a SIGN(s)
%               error message with the DBSTACK information is produced.
%
%Mandatory Output Parameters:
%   SUCC -      0 if no errors occur, 1 otherwise
%
%Optional Output Parameters:
%   None
%
%Special Notes:
%   None
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
%Date         Version   Editor            Changes Made
%----------   -------   --------------    --------------------

function SUCC=signs_log(LOGFILE,varargin)

SUCC=0;

try
    %read the logfile and display it
    if nargin==1
        eval(['type ',LOGFILE])

    %insert entries into the logfile
    else
        dtetime=datestr(now,0); %get the date and time
        if strcmp(LOGFILE,'stdout')
            fid=1;
        else
            fid=fopen(LOGFILE,'a');
```

```
        end
        for j=1:length(varargin)
            %read entry
            msg=varargin{j};

            %check for dbstack structure
            if isstruct(msg)
                fields=fieldnames(msg); %get the field names
                if length(fields)==2 & strcmp(fields{1},'line') & strcmp(fields{2},'name')
                    msg=['***SIGN(s): Error -- ',msg.name,' at line ',num2str(msg.line),' --> '
                        ',lasterr];
                end
            end

            %write message to screen and file
            disp(msg);
            if ischar(msg)
                fprintf(fid,'%s %s\n',dtetime,msg);
            else
                fprintf(fid,'%s %s\n',dtetime,num2str(msg));
            end
        end
        if fid~=1
            fclose(fid);
        end
        disp(['***SIGN(s): Messages have been logged in ',LOGFILE]);
    end
catch
    %display the error condition
    stk=dbstack;
    disp(['***SIGN(s): Error -- ',stk.name,' at line ',num2str(stk.line),' --> ',lasterr])
    disp('***SIGN(s): Messages have not been successfully logged')
    SUCC=1;
end
return
```

SIGNS_LZW

```
%Sign(s) Tool: SIGNS_LZW
%
%[S,ERR]=SIGNS_LZW(X,COMP,MEMDSK,SYM)
%
%The purpose of this function is to compress/uncompress a data
%sequence using the Lempel-Ziv-Welch algorithm.
%
%Mandatory Input Parameters:
%   X -         data vector to compress or uncompress.  X must be of type double, 8-bit
    integer, or 16-bit
%               integer for compression.  It must be of uint8 or uint16 for decompression.
%   COMP -      'c' to compress, 'u' to uncompress
%               If COMP='u' it can be followed with a suffix that indicates the datatype of
    the expanded
%               sequence.  The syntax is COMP='uF' where F is defined below.
%
%               F is a flag indicating how to decompress X.  X must be a compressed uint8 or
    uint16 sequence.
%               The following describes the meaning of F during decompression.  F is ignored
    for
%         for compression and is assumed 0 if not specified for decompression.
%
%          F  Interpretation
%             ---  ----------------
%          0  X will be expanded to a sequence of double-precision floats.  This is the
    default.
%
%          1  X will be expanded to a sequence of unsigned 8-bit integers.
%
%               2  X will be expanded to a sequence of signed 8-bit integers.
%
%               3  X will be expanded to a sequence of unsigned 16-bit integers.
%
%          4  X will be expanded to a sequence of signed 16-bit integers.
%
%   MEMDSK -    1 utilizes the LZW algorithm with disk memory (valid only if X is of type
    double)
%               0 utilizes the LZW algorithm with physical memory
%
%Optional Input Parameters:
%   SYM -       number of bits per symbol (either 8 or 16).  This is not the size of the
    codeword for the
%               the LZW algorithm, rather it indicates the size of a "character" for the
    character-based
%               LZW compression algorithm.  The default assumes 8 bits.  If SYM is 8 then 12-
    bit LZW is used,
%               if SYM=16 then 20-bit LZW is used.
%
```

```
%Mandatory Output Parameters:
%   Y - contains the compressed data sequence if COMP is 'c' and the uncompressed data
    sequence
%       if COMP is 'u'.  If COMP is 'c', Y will be an array of unsigned 8-bit or 16-bit
    integers.  The length
%       of this array may be longer than the input array X.  For example, if X is a double
    array, Y will
%       represent each byte of the compressed double array so that if compared to a
    corresponding double array,
%       Y would be of length L/8.  For example, if X=[1 2 3], the compressed sequence will be
%       Y=[0,1,0,16,16,240,3,241,1,16,80,64,16,81,0,0,128,64,255,240,0].  The length of this
    compressed array
%       is 21 bytes or 2.625 double precision numbers (21/8). If COMP is 'u', Y will be an
    array of double
%       equal to the original uncompressed data sequence.
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   LZW uses a code size of 12 bits and accesses the mex file (written in C), disklzw.dll or
    memlzw.dll depending
%   on the size of the input array.
%
%Copyright (c) 2003 James Giesbrecht

%Company:
%
%Revision History:
%Date        Version   Editor            Changes Made
%--------    -------   ------------      --------------------
%24/02/06    1.1       JEG               Added support for 16-bit symbols.
%
function [S,ERR]=signs_lzw(X,COMP,MEMDSK,varargin)

ERR=[];
lasterr=[];
try
    if MEMDSK==1
        S=signs_dll_disklzw(X,COMP);
    else
        SYM=8;
        if nargin>3 & isnumeric(varargin{1})
            SYM=varargin{1};
        end
        F=0;
        if length(COMP)>1
            F=str2num(COMP(2));
```

```
        end
        switch SYM
        case 8,
            S=signs_dll_memlzw8(X,COMP(1),uint8(F)); %use this for the normal compression
                table (5021 entries) 12-bit LZW
            %S=signs_dll_memlzw8_8191(X,COMP(1),uint8(F)); %use this for the large compression
                 table (8191 entries) 12-bit LZW
            %S=signs_dll_memlzw8_13(X,COMP(1),uint8(F)); %use this for the 13-bit LZW
                compression table (9029 entries)
        case 16,
            S=signs_dll_memlzw16(X,COMP(1),uint8(F));
        end
    end
catch
    %capture error condition
    ERR={dbstack,lasterr};
    S=[];
end
return
```

SIGNS_MIXER

```
%Sign(s) Tool: SIGNS_MIXER
%
%[Y,ERR]=SIGNS_MIXER(X,Wn,PH)
%
%This function mimics an up/down mixer found in analogue signal mixing.
%Essentially, the function performs y(t)=x(t) * cos(wt+phi).
%
%Mandatory Input Parameters:
%   X - an input vector representing x(t) (real or complex). The real
%       part of X is the in-phase component (I) and the imaginary part
%       of X is interpreted as the quadrature component (Q).
%   Wn - normalized carrier frequency where 1 is Fs/2 and 0 is dc
%       and where Fs is the output sampling frequency.
%   PH - phase of the mixer in radians. PH can be a vector
%       provided it is the same length as X.
%
%Optional Input Parameters:
%   None
%
%Mandatory Output Parameters:
%   Y - the resultant mixed output.
%
%Optional Output Parameters:
%   ERR -       two element cell array. The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%Special Notes:
%   Fs/2 must be greater than the highest frequency component of the mixed-up/down signal.
%   When mixing a signal X, the sampling rate of X must be the same as the desired
%   output sampling rate. This may mean that X has to be upsampled or downsampled to FS
%   before calling this function.
%
%Copyright (c) 2004 James Giesbrecht
%Revision History:
%Date        Version   Editor            Changes Made
%---------   -------   ------------      --------------------
function [Y,ERR]=signs_mixer(X,Wn,PH)
try
    dataR=reshape(real(X),1,prod(size(X)));
    dataI=reshape(imag(X),1,prod(size(X)));
    Y=dataR.*cos(2*pi*Wn/2*[0:length(X)-1]+PH)-dataI.*sin(2*pi*Wn/2*[0:length(X)-1]+PH);
    Y=reshape(Y,size(X,1),size(X,2));
    ERR=[];
catch
    ERR={dbstack,lasterr}; %capture error condition
    Y=0;
end
return
```

SIGNS_NRZ

```
%Sign(s) Tool: SIGNS_NRZ
%
%[Y,ERR]=SIGNS_NRZ(X,M)
%
%SIGNS_NRZ converts any positive vector X into a non-return-to-zero waveform.
%
%Mandatory Input Parameters:
%   X - a vector of symbols.  The symbols must correspond to the number of
%       levels.  For example, X must consist of 1's and 0's for bipolar NRZ.
%       If X consists of symbols of the set {0,1,2,3} and M = 4, then a
%       4-level NRZ will be returned.
%
%Optional Input Parameters:
%   M - for multilevel NRZ specify M=2,4,6,8,...
%       The default is M=2.  If M is odd, the result will be a RZ
%       (return to zero) waveform.
%
%Mandatory Output Parameters:
%   Y - the returned NRZ waveform.  If M=2 or if M
%       is unspecified, Y will contain levels of +/-1 where
%       elements of X>0 corresponding to +1 and elements of X=0
%       corresponding to -1.
%       For M>2 and M even, Y will contain M levels between +/-1.
%       If M is odd, Y will contain M levels between +/-1 with one
%       of the levels equal to zero.
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Copyright (c) 2004 James Giesbrecht
%
%Company:
%
%Revision History:
%Date         Version   Editor              Changes Made
%-----------  --------  ----------------    --------------------
function [Y,ERR]=signs_nrz(X,varargin)
try
    ERR=[];
    M=2;
    if nargin>=2
        if isnumeric(varargin{1})
            M=abs(varargin{1});
        end
    end
```

```
    %Check M
    if M==2
        X(find(X>0))=1;
        X(find(X<=0))=-1;
        Y=X;
    else
        %shift the input vector according to M
        Y=2/(M-1)*X-1;
    end
catch
    %capture error condition
    ERR={dbstack,lasterr};
    Y=0;
end
return
```

SIGNS_PSK

```
%Sign(s) Tool: SIGNS_PSK
%
%[Y,ERR]=SIGNS_PSK(X,R,Fc,Fs,Po,M,TYP)
%
%SIGNS_PSK creates a continuous-phase phase-shift-keyed modulated carrier of the specified
%peak phase deviation and carrier frequency.
%
%Mandatory Input Parameters:
%   X - digital data stream in the form of a vector of zeros and ones
%   R - bit rate
%   Fc - carrier frequency in Hz
%   Fs - sampling frequency in Hz
%   Po - phase offset from positive abcissa of constellation plane (radians)
%
%Optional Input Parameters:
%   M - number of levels of modulation.  M=2 for 2-PSK, M=4 for 4-PSK, etc.
%       The default is M=2.
%   TYP - is an optional string argument that specifies the format
%   of the output data. It commonly contains a datatype specifier
%   like 'int' or 'float' followed by an integer giving the size in
%   bits.  Any of the following strings, either the MATLAB version,
%   or their C or Fortran equivalent, may be used.  The default type
%   is 'single'.
%        MATLAB    C or Fortran      Description
%        'uchar'   'unsigned char'   unsigned character,  8 bits.
%        'schar'   'signed char'     signed character,  8 bits.
%        'int8'    'integer*1'       integer, 8 bits.
%        'int16'   'integer*2'       integer, 16 bits.
%        'int32'   'integer*4'       integer, 32 bits.
%        'uint8'   'integer*1'       unsigned integer, 8 bits.
%        'uint16'  'integer*2'       unsigned integer, 16 bits.
%        'uint32'  'integer*4'       unsigned integer, 32 bits.
%        'single'  'real*4'          floating point, 32 bits.
%        'float32' 'real*4'          floating point, 32 bits.
%   See Help fread for more information on file types.
%   TYP may also be of the form intX or uintX where X is of the
%   following set {1,2,3,...,32} such that the output data
%   consists of whole numbers in the range of -2^X to 2^X-1.
%
%Mandatory Output Parameters:
%   Y - vector of the PSK signal. If an error occurs Y will be empty.
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%Special Notes:
%   * X is padded with zeros if the length of X is not a multiple of M
```

```
%    * Note that TYP does not cause SIGNS_FSK to return a vector of TYP, but rather,
%    a double precision vector containing elements restricted by TYP.  For example,
%    if TYP='int8', the output vector will consist of double precision whole numbers
%    between -128.0000000<=Y<=+127.0000000
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
%Date         Version   Editor            Changes Made
%----------   -------   ----------        --------------------
%17/03/05     1.01      JEG               Corrected mapping of symbols to phases such that phase
    =2*pi/M*<symbol>
function [Y,ERR]=signs_psk(X,R,Fc,Fs,Po,varargin)
try
    %set defaults
    TYP='single';
    M=2;
    ERR=[];

    %check number and correct use of input arguments
    switch nargin
    case 5,
    case 6,
        if isnumeric(varargin{1})
            M=varargin{1};
        else
            TYP=varargin{1};
        end
    case 7,
        if isnumeric(varargin{1}) & ischar(varargin{2})
            M=varargin{1};
            TYP=varargin{2};
        else
            if isnumeric(varargin{2}) & ischar(varargin{1})
                M=varargin{2};
                TYP=varargin{1};
            end
        end
    end

    %transpose X to a row vector
    sz=size(X);
    if sz(1)>sz(2)
        X=X';
    end

    %pad X with zeros if its length is not a multiple of log2(M)
    if mod(length(X),log2(M))~=0
        X=[X,zeros(1,length(X)*log2(M)-length(X))];
    end
```

```
%map bits to symbols
Nbits=floor(log2(M)); %get number of bits per symbol
if Nbits>1
    %count=1;
    %for i=1:Nbits:length(X)
    %    bits=X(i:i+Nbits-1);
    %    data(count)=bits*(2.^[0:Nbits-1])';
    %    count=count+1;
    %end
    data=2.^[0:Nbits-1]*reshape(X,Nbits,floor(length(X)/Nbits));
else
    data=X;
end

%upsample to sampling rate (do not convert to NRZ)
data=signs_sgzoom(data,Fs/R);

%generate time base
Ts=1/Fs;
T=length(data)*Ts;
t=[0:Ts:T-Ts];

%compute phase sensitivity
Kp=2*pi/M;

%compute phase
phase=Kp*data+Po;

%compute PM waveform
I=cos(phase);
Q=sin(phase);
Yi=cos(2*pi*Fc*t).*I-sin(2*pi*Fc*t).*Q;
Yi=Yi/max(abs(Yi)); %ensure signal lies between -1 and +1

%format output
switch TYP
case {'uchar','unsigned char','uint8','integer*1'}, %unsigned 8-bit integers
    Y=floor((2^8-1)*(Yi+1)/2);
case {'schar','signed char','int8','integer*1'},    %signed 8-bit integers
    Y=floor((2^8-1)*Yi/2-.5);
case {'int16','integer*2'},                         %integer, 16 bits.
    Y=floor((2^16-1)*Yi/2-.5);
case {'int32','integer*4'},                         %integer, 32 bits.
    Y=floor((2^32-1)*Yi/2-.5);
case {'uint16','integer*2'},                        %unsigned integer, 16 bits.
    Y=floor((2^16-1)*(Yi+1)/2);
case {'uint32','integer*4'},                        %unsigned integer, 32 bits.
    Y=floor((2^32-1)*(Yi+1)/2);
case {'single','real*4','float32'},                 %floating point, 32 bits.
```

```
        Y=Yi;
    otherwise
        Y=Yi; %set the default condition

        %check for non-standard int type
        if length(TYP)>=4
            if strcmp(TYP(1:3),'int') %check for signed integer
                N=str2num(TYP(4:end)); %get the number of bits
                if N>=1 & N<=32
                    Y=floor((2^N-1)*Yi/2-.5); %scale the output
                end
            elseif length(TYP)>=5 & strcmp(TYP(1:4),'uint') %check for unsigned integer
                N=str2num(TYP(5:end)); %get the number of bits
                if N>=1 & N<=32
                    Y=floor((2^N-1)*(Yi+1)/2); %scale the output
                end
            end
        end
    end
catch
    %capture error condition
    ERR={dbstack,lasterr};
    Y=0;
end
return;
```

SIGNS_READ

```
%Sign(s) Tool: SIGNS_READ
%
%[Y,ERR]=SIGNS_READ(FILE,PRECISION,MASK,COMP,NUM)
%
%The purpose of this function is to read values from a file.
%
%Mandatory Input Parameters:
%   FILE -      filename and path of binary data file
%   PRECISION - type of data to read according to the following.
%                   MATLAB    C or Fortran     Description
%                   'uchar'   'unsigned char'  unsigned character, 8 bits.
%                   'schar'   'signed char'    signed character, 8 bits.
%                   'int8'    'integer*1'      integer, 8 bits.
%                   'int16'   'integer*2'      integer, 16 bits.
%                   'int32'   'integer*4'      integer, 32 bits.
%                   'int64'   'integer*8'      integer, 64 bits.
%                   'uint8'   'integer*1'      unsigned integer, 8 bits.
%                   'uint16'  'integer*2'      unsigned integer, 16 bits.
%                   'uint32'  'integer*4'      unsigned integer, 32 bits.
%                   'uint64'  'integer*8'      unsigned integer, 64 bits.
%                   'single'  'real*4'         floating point, 32 bits.
%                   'float32' 'real*4'         floating point, 32 bits.
%                   'double'  'real*8'         floating point, 64 bits.
%                   'float64' 'real*8'         floating point, 64 bits.
%               see FREAD for more options.
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   MASK -      this parameter is used to modify the data, after it has been
%               read from the file, by specifying what happens to each bit
%               of each sample.  For example, for 32-bit data the following
%               MASK strings will strip off the upper 2 bits and lower 12 bits
%               of each sample before returning:
%                   MASK='b00111111111111111111000000000000' or
%                   MASK='h3FFFF000'
%               Consecutive zero bits in the LSB positions correspond to
%               right shifts of the sample while zero bits to the left of the
%               first one bit correspond to bit masking.  Example:
%                   MASK='b00111111111111001111000000000000'
%                         MMMMMMMMMMMMMMMMMMSSSSSSSSSSSS
%               will cause each sample to be shifted right 12 bits (count the S's) and
%               the upper two bits and the 17th and 18th bits to be masked.  The default
%               is to do nothing to the samples (i.e. MASK='h1FFFFFFFFFFFFF').
%   COMP -      one of the following
%                   'I' to return in-phase data (assumed to be all odd numbered samples)
%                   'Q' to return quad-phase data (assumed to be all even numbered samples)
%                   'complex' to return I+jQ
%                   'real' to interpret all data as real data (this is the default)
%   NUM -       number of elements of PRECISION to read from the file (can be 1 to inf).
```

```
%               The default is to read all elements.
%
%Mandatory Output Parameters:
%   Y -         data from file converted to double precision.
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   I and Q will be swapped if the function detects that I is following Q.  The
%   check is determined simply by looking for the relative position of the maximum
%   in I and Q.
%
%Copyright (c) 2005 James Giesbrecht
%Revision History:
%Date        Version   Editor            Changes Made
%---------   -------   ------------      --------------------
%27/04/06    1.1       JEG               Added feature to read less than full file

function [Y,ERR]=signs_read(FILE,PRECISION,varargin)
try
    %default conditions
    ERR=[];
    COMP='real';
    MASK=bitmax; %set to maximum bit value
    N=ceil(log2(MASK));
    RSHFT=0;
    NUM=inf;

    %get optional parameters parameter
    if nargin>2 & nargin<6
        for i=1:length(varargin)
            param=lower(varargin{i});
            if ischar(param)
                %check for the COMP parameter
                switch param
                case {'i','q','complex','real'},
                    COMP=param;
                otherwise
                    %check for MASK parameter
                    switch param(1)
                    case {'b','h'},
                        switch param(1)
                        case 'b',
                            MASK=param(2:end);
                        case 'h',
                            MASK=hex2dec(param(2:end));
                            MASK=dec2bin(MASK);
```

```
                    end
                    %get number of right shifts (all consecutive zero bits in the LSB
                        positions of the mask i.e. S's (see comments above))
                    RSHFT=length(MASK(max(find(MASK=='1')):end))-1;

                    %get the mask for the bitand function below (all bits to the left of
                        the S's (see comments above)
                    MASK=MASK(1:length(MASK)-RSHFT);
                    N=length(MASK); %get number of bits in mask
                    MASK=bin2dec(MASK); %convert to decimal
                end
            end
        elseif isnumeric(param)
            NUM=param;
        end
    end
end

%read file
fid=fopen(FILE,'r');
[data,count]=fread(fid,NUM,PRECISION);
fclose(fid);

%apply right shift
data=floor(data./2^RSHFT); %shift the data right

%mask positive values
posidx=find(data>0); %find all data greater than zero
data(posidx)=bitand(data(posidx),MASK);

%need to apply two's complement to mask negative values
negidx=find(data<0); %find all data less than zero
temp=bitxor(-data(negidx),2^N-1)+1; %invert samples and add 1
data(negidx)=-(bitxor(bitand(temp,MASK),2^N-1)+1); %get new negative value

%get I-Q data
I=data(1:2:end)'; %get in-phase (assumed)
Q=data(2:2:end)'; %get quad-phase (assumed)

%check the phases
maxI=min(find(I==max(I)));
maxQ=min(find(Q==max(Q)));
if abs(maxI-maxQ)==1 & maxI>maxQ %swap I and Q since we guessed the wrong order
    temp=I;
    I=Q;
    Q=temp;
end

%prepare output
I=reshape(I,1,length(I));
```

```
    Q=reshape(Q,1,length(Q));
    data=reshape(data,1,length(data));
    switch COMP
    case 'i',
        Y=I;
    case 'q',
        Y=Q;
    case 'complex',
        Y=complex(I,Q);
    case 'real',
        Y=data;
    end
catch
    %capture error condition
    ERR={dbstack,lasterr};
    Y=[];
    I=[];
    Q=[];
end
return
```

SIGNS_RICEPDF

```
%Sign(s) Tool: SIGNS_RICEPDF
%
%[Y,ERR]=SIGNS_RICEPDF(X,S2,SIGMA)
%
%The purpose of this function is to returns the rice pdf with noncentrality
%parameter, S2, and standard deviation, SIGMA, at the values in X.
%
%Mandatory Input Parameters:
%   X -         values at which to compute the pdf
%   S2 -        noncentrality parameter
%   SIGMA -     standard deviation of pdf
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   None
%
%Mandatory Output Parameters:
%   Y -         rician pdf
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%----------  --------  ----------        --------------------
%
%   References:
%      [1]  J. G. Proakis, "Digital Communications" 2nd Ed.,
%           1989, McGraw-Hill Book Co.; pp29-31.
%      [2]  E. Kreysig, "Advanced Engineering Mathematics" 6th Ed.,
%           1988, John Wiley & Sons Inc.; pp208.
function [Y,ERR]=signs_ricepdf(X,S2,SIGMA)
try
    %check input parameters
    if nargin < 3,
        SIGMA = 1;
    end

    if nargin < 2;
        S2 = 0;
    end

    if nargin < 1,
```

```
        error('Requires at least one input argument.');
    end

    if find(X<0)
        error('X must be non-negative')
    end

    %prepare output vector
    Y = zeros(size(X));

    %compute modified bessel function of the first kind
    Io=besseli(0,X*sqrt(S2)/SIGMA);

    %compute pdf
    Y=X/SIGMA^2.*exp(-(X.^2+S2)/2/SIGMA^2).*Jo;

    % Return NaN if SIGMA is negative or zero.
    k1 = find(SIGMA <= 0);
    if any(k1)
        tmp    = NaN;
        Y = tmp(ones(size(k1)));
    end
    ERR=[];
catch
    %capture error condition
    ERR={dbstack,lasterr};
    y=[];
end
return
```

SIGNS_RMS

```
%Sign(s) Tool: SIGNS_RMS
%
%[Y,ERR]=SIGNS_RMS(X)
%
%The purpose of this function is to return the root-mean-squre of a vector X.
%It is computed as the square-root of the sum of each element of X^2 divided
%by the length of X.
%
%Mandatory Input Parameters:
%   X -         abritrary numeric vector
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   None
%
%Mandatory Output Parameters:
%   Y -         rms value of input vector
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   None
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%---------   -------   ------------      --------------------

function [Y,ERR]=signs_rms(X)
try
    Y=sqrt(1/length(X)*sum(X.^2));
    ERR=[];
catch
    %capture error condition
    ERR={dbstack,lasterr};
    Y=0;
end
return
```

SIGNS_RNDBIKAPPA

```
%Sign(s) Tool: SIGNS_RNDBIKAPPA
%
%[Y,ERR]=SIGNS_RNDBIKAPPA(M,N,S,K)
%
%SIGNS_RNDBKAP returns an array of random numbers in the
%interval (-1,1) from a bi-kappa distribution with zero mean.
%
%Mandatory Input Parameters:
%   M - number of rows in the resultant array.
%   N - number of columns in the resultant array.
%   S - standard deviation of the distribution.
%   K - extensivity factor (K>0.5)
%
%Optional Input Parameters:
%   None
%
%Mandatory Output Parameters:
%   Y - array of random numbers from a bi-kappa distribution.
%
%Optional Output Parameters:
%   None
%
%Special Notes:
% The bi-kappa distributed random numbers are generated from
%an inverse sampling algorithm of a uniform distribution.
%
%Copyright (c) 2007 James Giesbrecht

%Revision History:
%Date         Version   Editor              Changes Made
%---------    -------   ------------        --------------------
function [Y,ERR]=signs_rndbikappa(M,N,S,K)

ERR=[];
try
    %check S and K
    if min(S)<=0
        error('Standard deviation invalid');
    end
    if min(K)<=0.5
        error('Kappa must be greater than 0.5');
    end

    %range of output values
    H=1; %high value
    L=-1; %low value

  %generate bi-kappa distribution
```

```
bins=1000;
RL=-1;
RH=1;
step=(RH-RL)/(bins-1);
d=signs_bikappapdf([RL:step:RH],S,K);

%generate normalized CDF of bi-kappa distribution
D=cumsum(d);

%generate the random numbers in interval of (0,1)
%these are the cumulative probability values used to perform the
%inverse sampling
numbers=rand(1,M*N); %generate the uniformly distributed random numbers

%prepare constants
y=numbers; %numbers from uniform distribution
Y=zeros(1,M*N); %array to fill
K=M*N; %number times to interate
maxX=length(D); %maximum value of x if we cannot find a value of D greater than y(c)
minX=1; %minimum value of x if we cannot find a value of D less than y(c)
m=(H-L)/(bins-1); %slope for converting the determined x value of the CDF to the interval
    (L,H)
b=L-m; %intercept for converting x value of CDF to interval (L,H)

%perform inverse sampling
for c=1:K
    %determine positions of nearest neighbours on CDF
    x_lo=max(find(D<y(c))); %find closest neighbour in D on the low side
    x_hi=min(find(D>y(c))); %find closest neighbour in D on the high side

    %determine cumalitive probability for nearest neighbours
    y_lo=D(x_lo);
    y_hi=D(x_hi);
    if isempty(x_lo)
        x=minX;
    elseif isempty(x_hi)
        x=maxX;
    else
        %linearily interpolate to find the fractional index corresponding to x
        x=((x_hi-x_lo)/(y_hi-y_lo))*(y(c)-y_lo)+x_lo;
    end

    %convert x to the interval (L,H) using y=mx+b
    %m=(H-L)/(bins-1);
    %b=L-m;
    Y(c)=m*x+b;
end

%reshape result into appropriate array dimensions
Y=reshape(Y,M,N);
```

```
catch
    %capture error condition
    ERR={dbstack,lasterr};
    Y=[];
end
return
```

SIGNS_ROUND

```
%Sign(s) Tool: SIGNS_ROUND
%
%[Y,ERR]=SIGNS_ROUND(X,M)
%
%SIGNS_ROUND rounds the numeric elements in X to the number of significant digits M.
%
%Mandatory Input Parameters:
%   X - a numeric array.  X can be a complex numeric array but the imaginary component
%       may not be rounded correctly.  To solve this, pass the real component and
%       imaginary component to the function separately.
%
%Optional Input Parameters:
%   M - number of significant digits.  If M is less than 1 or not specified then all
%       elements of X are rounded to the nearest integer using the standard Matlab
%       ROUND function.  The default for M is 0.
%
%Mandatory Output Parameters:
%   Y - a numeric array containing rounded elements.
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Copyright (c) 2007 James Giesbrecht
%
%Company:
%
%Revision History:
%Date         Version   Editor             Changes Made
%-----------  --------  -----------------  ---------------------
function [Y,ERR]=signs_round(X,varargin)
try
    ERR=[];
    M=0;
    if nargin>=2
        if isnumeric(varargin{1})
            M=abs(varargin{1});
        end
    end

    %round each element of X
    if M<1
        Y=round(X);
    else
        Y=X;
        idx=find(X~=0);
        if ~isempty(idx)
```

```
            %round small numnbers
            smallnums=find(abs(X)<1 & X~=0); %find fractions
            Esmall=fix(log10(abs(X(smallnums))));
            Y(smallnums)=Y(smallnums)./10.^Esmall; %move decimal point so that number looks
                like 0.XXXX
            Y(smallnums)=fix(Y(smallnums)*10^M+.5*sign(Y(smallnums)))/10^M.*10.^Esmall;

            %round big numbers
            bignums=find(abs(X)>=1); %find whole numbers >= 1
            Ebig=ceil(log10(abs(X(bignums))));
            Y(bignums)=Y(bignums)./10.^Ebig; %move decimal point so that number looks like 0.
                XXXX
            Y(bignums)=fix(Y(bignums)*10^M+.5*sign(Y(bignums)))/10^M.*10.^Ebig;

        end
    end
catch
    %capture error condition
    ERR={dbstack,lasterr};
    Y=0;
end
return
```

SIGNS_SGZOOM

```
%Sign(s) Tool: SIGNS_SGZOOM
%
%[Y,ERR]=SIGNS_SGZOOM(DATA,N)
%
%This function zooms in on a DATA sequence by N times.  If DATA=[1 2 3]
%and N=3, then Y=[1 1 1 2 2 2 3 3 3].
%
%Mandatory Input Parameters:
%   DATA -      input vector of samples to zoom in on
%   N    -      upsampling factor (i.e. "zoom" factor)
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   None
%
%Mandatory Output Parameters:
%   Y -         upsampled version of DATA
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   None
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
%Date         Version   Editor            Changes Made
%-----------  --------  ----------------  --------------------
function [Y,ERR]=signs_sgzoom(DATA,N)
try
    Y=DATA'*ones(1,round(N));
    Y=reshape(Y',1,prod(size(Y)));
    ERR=[];
catch
    %capture error condition
    ERR={dbstack,lasterr};
    Y=0;
end
return;
```

SIGNS_SNR

```
%Sign(s) Tool: SIGNS_SNR
%
%[Y,ERR]=SIGNS_SNR(X,METH)
%
%The purpose of this function is to return the SNR of the signal X.
%
%Mandatory Input Parameters:
%   X -         input data sequence vector containing a baseband signal plus noise
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   METH -      method by which the SNR is computed.  METH can be any of
%                   Method          Description
%                   ---------       -----------------
%                   aisbett                 sqrt(#(A^2,A^2))    (default method)
%                                   SNR = ------------------------
%                                         E(A^2)-sqrt(#(A^2,A^2))
%
%   BLK -       Segment length for processing.  Setting BLK to an integer less
%               then the length of X forces SIGNS_SNR to process the signal
%               as a series of blocks of data of length BLK.  If X is not an
%               integral number of BLK blocks long, the residue of samples
%               are ignored.  BLK must be an integer.
%
%Mandatory Output Parameters:
%   SNR -       Estimated signal-to-noise ratio (SNR) in units of W/W.  A negative
%               result indicates that the SNR estimate is received a negative power
%               estimate from the hash function.
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error
%   PS -        estimate of true signal power.  A negative result indicates that the
%               hash function was unable to correctly estimate the power.
%   PSN -       power of signal plus noise.  A negative result indicates that the
%               hash function was unable to correctly estimate the power.
%
%Special Notes:
%   None
%
%Copyright (c) 2006 James Giesbrecht

%Revision History:
%Date       Version  Editor          Changes Made
%---------  -------  -------------   --------------------
function [SNR,ERR,PS,PSN]=signs_snr(X,varargin)
try
    ERR=[];
```

```
    BLK=length(X); %default block size
    METH='aisbett';

    %check for optional block size
    if nargin>1
        for j=1:nargin-1
            if isnumeric(varargin{j})
                BLK=varargin{j}(1);
            end
            if ischar(varargin{j})
                METH=varargin{j};
            end
        end
    end
    %compute the SNR --  %this method generally only works for constant envelope signals
    switch METH
    case 'aisbett'
        %extract signal envelope
        A=sqrt(X.^2+(imag(hilbert(X))).^2);

        %constant of proportionality
        C=1.0;

        %estimate true signal power
        hash=signs_aisbetthash(A.^2,A.^2); %this yields peak power squared
        PS=sqrt(hash); %to get peak power, take the square root

        %compute average power of signal plus noise
        Esn=mean(A.^2); %this yields (amplitude^2 + 2*<average noise power>)
        PSN=(Esn-PS); %this yields 2*<average noise power>

        %compute SNR; this yields peak signal power over 2*<average noise> or in other words
        %average signal power over average noise power.
        SNR=PS/PSN;

        %convert to average powers
        PS=PS/2;
        PSN=PSN/2; %remove the factor of 2 in PSN
    otherwise
    end
catch
    ERR={dbstack,lasterr}; %capture error condition
    SNR=0;
end
return
```

SIGNS_ST4285

```
%Sign(s) Tool: SIGNS_ST4285
%
%[Y,ERR,PHASE]=SIGNS_ST4285(X,R,Fs,TYP,NOPLOT)
%
%SIGNS_ST4285 creates a STANAG4285 baseband signal filtered with
%a blackman-harris bandpass FIR.  This filter has corner
%frequencies of 300Hz and 3300Hz.
%
%Mandatory Input Parameters:
%   X - a vector of ones and zeros representing data bits where
%       X(1) is the first bit to be transmitted.
%   R - information throughput (bps) of data only.  R
%       can take on values of 1200, 2400, or 3600 bps.  The default
%       is 1200 bps and SIGNS_ST4285 reverts to 1200 bps if R is
%       not one of the valid values.
%   Fs- sampling rate in samples/sec.
%
%Optional Input Parameters:
%   TYP - is an optional string argument that specifies the format
%   of the output data. It commonly contains a datatype specifier
%   like 'int16' or 'float32' followed by an integer giving the size in
%   bits.  Any of the following strings, either the MATLAB version,
%   or their C or Fortran equivalent, may be used.  The default type
%   is 'single'.
%        MATLAB    C or Fortran      Description
%        'uchar'   'unsigned char'   unsigned character,  8 bits.
%        'schar'   'signed char'     signed character,  8 bits.
%        'int8'    'integer*1'       integer, 8 bits.
%        'int16'   'integer*2'       integer, 16 bits.
%        'int32'   'integer*4'       integer, 32 bits.
%        'uint8'   'integer*1'       unsigned integer, 8 bits.
%        'uint16'  'integer*2'       unsigned integer, 16 bits.
%        'uint32'  'integer*4'       unsigned integer, 32 bits.
%        'single'  'real*4'          floating point, 32 bits.
%        'float32' 'real*4'          floating point, 32 bits.
%   See Help fread for more information on file types.
%   TYP may also be of the form intX or uintX where X is of the
%   following set {1,2,3,...,32} such that the output data
%   consists of whole numbers in the range of -2^X to 2^X-1.
%
%   NOPLOT - 1 to suppress plots of the stanag signal, 0 to plot the data.
%          The default is 1.
%
%Mandatory Output Parameters:
%   Y - vector of the STANAG4285 signal. If an error occurs Y will be empty.
%
%Optional Output Parameters:
%   PHASE - vector of the phase of the STANAG4285 signal (in radians) or the
```

```
%           empty vector if an error occurs.
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%Special Notes:
%   * See NATO STANAG 4285 documentation.
%   * Note that TYP does not cause SIGNS_ST4285 to return a vector of TYP, but rather,
%   a double precision vector containing elements restricted by TYP.  For example,
%   if TYP='int8', the output vector will consist of double precision whole numbers
%   between -128.0000000<=Y<=+127.0000000
%
%Copyright (c) 2003 James Giesbrecht
%Revision History:
%Date        Version   Editor            Changes Made
%---------   -------   ------------      --------------------
%
function [Y,ERR,varargout]=signs_st4285(X,R,Fs,varargin)
try
    %set defaults
    TYP='single';
    NOPLOT=1;
    ERR=[];
    M_ary=2;

    %check number and correct use of input arguments
    switch nargin
    case 3,
    case {4,5},
        for i=1:length(varargin)
            if ischar(varargin{i})
                TYP=varargin{i};
            else
                if varargin{i}==0
                    NOPLOT=0;
                end
            end;
        end;
    end

    %determine PSK level and bit time
    switch R
    case 1200,
        M_ary=2;                            %set level of PSK to BPSK
        ptitle='1200bps 2-PSK Data';
    case 2400,
        M_ary=4;                            %set level of PSK to QPSK
        ptitle='2400bps 4-PSK Data';
    case 3600,
        M_ary=8;                            %set level of PSK to 8-PSK
        ptitle='3600bps 8-PSK Data';
```

```
end

%map data bits to symbol numbers and break into data sub-frames
Num_Frames=ceil(length(X)/log2(M_ary)/128);
%Num_Frames=ceil(length(dataSYM)/128);    %determine number of frames
if mod(length(X)/log2(M_ary),128)~=0
    %pad dataSYM with symbol zero to fit data blocks
    pad=128*Num_Frames*log2(M_ary)-length(X);
    X=[X,zeros(1,pad)];
end
dataSYM=get_symbols(X,M_ary);
Num_SubFrames=ceil(length(dataSYM)/32); %determine number of sub-frames
dataSYM=reshape(dataSYM,32,Num_SubFrames)';

%compose synchronization sub-frame
syncSYM=get_symbols(sync_bits,2);

%compose the reference sub-frame as per page A-3 and page A-7 fig A-3 of the standard
refSYM=get_symbols(zeros(1,16*log2(M_ary)),M_ary);

%get scrambling symbol numbers
scramSYM=get_symbols(scram_bits,8);

%compose frames
subCNT=1;        %initialize sub-frame counter
for idx=1:Num_Frames
    dataFRAME=[];
    for jdx=1:3
        dataFRAME=[dataFRAME,dataSYM(subCNT,:),refSYM];
        subCNT=subCNT+1;
    end;
    dataFRAME=[dataFRAME,dataSYM(subCNT,:)];

    %add sync symbols and scramble data and reference symbols
    FRAMES(idx,:)=[syncSYM,mod(scramSYM+dataFRAME,8)];
end

%flatten FRAMES matrix into a vector of consecutive frames
FRAMES=reshape(FRAMES',1,prod(size(FRAMES)));

%get symbol phases
FRAMES=map_symbols(FRAMES);

%compute time base
Tsym=1/2400; %base data rate in STANAG documentation
Ts=1/Fs;

%upsample symbols to fit timebase
slope=diff(FRAMES); %get slope between samples
ZF=round(Tsym/Ts);%get zoom factor
```

```
PHASE=cumsum([FRAMES(1),sigzoom(slope,ZF)/ZF]);

%adjust timebase
t=[0:length(PHASE)-1]*Ts;

%generate IQ informatoin
I=real(PHASE);%cos(PHASE);
Q=imag(PHASE);%sin(PHASE);

%design LPF transmission filter
%the transmission filter is a raised cosine filter with a square-root magnitude, roll-off
    of 0.2, Kaiser window with parameter 1
%and 40th order (41 taps) (at Fs=9600)
Nord=round(40*Fs/9600); %required to get appropriate order for rolloff
FIRtran=firrcos(Nord,1200,0.2,Fs,'rolloff','sqrt',(Nord+mod(Nord,2))/2,kaiser(Nord+1,1));
%Filter coefficients if Fs=9600
%Num=[-0.00335222036259,-0.00166012114162, 0.00252573279234, 0.00627425505627,
    0.00589943606950,-0.00000000000000,-0.00847652555243,...
%   -0.01321630823988,-0.00869110957833, 0.00496287632074, 0.02005422806027,
    0.02482553603050, 0.01123466928961,-0.01753061297288,...
%   -0.04526057732903,-0.04930512890708,-0.01301954895877, 0.06275259684864,
    0.15642733070437, 0.23374184271459, 0.26366197723676,...
%    0.23374184271459, 0.15642733070437,
    0.06275259684864,-0.01301954895877,-0.04930512890708,-0.04526057732903,-0.01753061297288,...

%    0.01123466928961, 0.02482553603050, 0.02005422806027,
    0.00496287632074,-0.00869110957833,-0.01321630823988,-0.00847652555243,...
%   -0.00000000000000, 0.00589943606950, 0.00627425505627,
    0.00252573279234,-0.00166012114162,-0.00335222036259];

%apply LPF transmission filters
If=filter(FIRtran,1,I);
Qf=filter(FIRtran,1,Q);
If=If(ceil(length(FIRtran)/2):end-floor(length(FIRtran)/2)); %remove end effects
Qf=Qf(ceil(length(FIRtran)/2):end-floor(length(FIRtran)/2)); %remove end effects

%modulate the IF (1800Hz) sub-carrier
SI=cos(2*pi*1800*t(1:length(If))).*If;
SQ=-sin(2*pi*1800*t(1:length(Qf))).*Qf;
S=SI+SQ;

%prepare sideband BPF
FIR_N=round(128*Fs/9600); %arbitrary method for selecting window length
H=blackmanharris(FIR_N);  %blackman-harris window
Fc1=300;
Fc2=3300;
Wc1=Fc1*2/Fs;
Wc2=Fc2*2/Fs;
FIR=fir1(FIR_N-1,[Wc1 Wc2],H);
Yi=filter(FIR,1,S);
```

```
Yi=Yi(ceil(length(FIR)/2):end-floor(length(FIR)/2));
Yi=Yi/max(abs(Yi)); %normalize signal to remain within -1 to +1

%plot results
if ~NOPLOT
    %plot LPF FIR filter design
    figure
    [h,f,s]=freqz(FIRtran,1,Nord,Fs);
    s.plot='mag';s.xunits='hz';s.yunits='db';freqzplot(h,f,s);
    title('LPF Baseband Transmission FIR Magnitude Response');

    %plot BPF FIR filter design
    figure
    [h,f,s]=freqz(FIR,1,FIR_N,Fs);
    s.plot='mag';s.xunits='hz';s.yunits='db';freqzplot(h,f,s);
    title('BPF Sideband FIR Magnitude Response');

    %plot time-domain waveform
    figure
    plot(t(1:length(Yi)),Yi);
    ylabel('Amplitude')
    xlabel('Time (seconds)')
    title(['STANAG 4285 Waveform -- ',ptitle])
    grid on

    %plot phase symbols
    figure
    plot(t,angle(PHASE))
    ylabel('Phase (rad)')
    xlabel('Time (seconds)')
    title(['STANAG 4285 Phase -- ',ptitle])
    grid on

    %plot psd
    figure
    psd(Yi,512,Fs,512);
    xlabel('Frequency (Hz)')
    title(['PSD of STANAG 4285 Waveform -- ',ptitle])
end;

%format output
switch TYP
case {'uchar','unsigned char','uint8','integer*1'}, %unsigned 8-bit integers
    Y=floor((2^8-1)*(Yi+1)/2);
case {'schar','signed char','int8','integer*1'},    %signed 8-bit integers
    Y=floor((2^8-1)*Yi/2-.5);
case {'int16','integer*2'},                         %integer, 16 bits.
    Y=floor((2^16-1)*Yi/2-.5);
case {'int32','integer*4'},                         %integer, 32 bits.
    Y=floor((2^32-1)*Yi/2-.5);
```

```
    case {'uint16','integer*2'},                    %unsigned integer, 16 bits.
        Y=floor((2^16-1)*(Yi+1)/2);
    case {'uint32','integer*4'},                    %unsigned integer, 32 bits.
        Y=floor((2^32-1)*(Yi+1)/2);
    case {'single','real*4','float32'},             %floating point, 32 bits.
        Y=Yi;
    otherwise
        Y=Yi; %set the default condition

        %check for non-standard int type
        if length(TYP)>=4
            if strcmp(TYP(1:3),'int') %check for signed integer
                N=str2num(TYP(4:end)); %get the number of bits
                if N>=1 & N<=32
                    Y=floor((2^N-1)*Yi/2-.5); %scale the output
                end
            elseif length(TYP)>=5 & strcmp(TYP(1:4),'uint') %check for unsigned integer
                N=str2num(TYP(5:end)); %get the number of bits
                if N>=1 & N<=32
                    Y=floor((2^N-1)*(Yi+1)/2); %scale the output
                end
            end
        end
    end
    if nargout==3
        varargout{1}=PHASE;
    end;
catch
    %capture error condition
    ERR={dbstack,lasterr};
    Y=[];
    PHASE=[];
end
return;

%map_symbol maps symbol numbers to phases in radians.
function Z=map_symbols(X)
SYMbase=exp(j*pi/4*[0:7]);
Z=SYMbase(X+1);
return;

%get_symbol maps bits to the appropriate symbol numbers given the data stream, X, and
%the level of PSK (ie. M_ary=2 for BPSK, 4 for QPSK, and 8 for 8-PSK).
function Z=get_symbols(X,M_ary)
switch M_ary
case 2,
    K=[1];                                  %set multiplier for bit-symbol mapping
    Bit_to_SYM=[0,4];                       %choose appropriate symbols
case 4,
    K=[2,1];                                %set multiplier for bit-symbol mapping
```

```
    Bit_to_SYM=[0,2,6,4];                %choose appropriate symbols
case 8,
    K=[4,2,1];                           %set multiplier for bit-symbol mapping
    Bit_to_SYM=[1,0,2,3,6,7,5,4];        %choose appropriate symbols
end

%map data bits to symbol numbers
temp=reshape(X,length(K),length(X)/length(K))'*K';
Z=Bit_to_SYM([temp+1]);
return;

%sync_bits computes the pseudo-random sequence for the frame synchronization bits
%based on the 5-bit initialization sequence, 11010 (see NATO standard).
function Z=sync_bits()
%sync symbols are generated by the following algorithm
%taps=[1,1,0,1,0];
%for idx=1:80
%    Z(idx)=taps(5);
%    temp(1)=xor(taps(5),taps(3));
%    temp(2:5)=taps(1:4);
%    taps=temp;
%end
Z=[0,1,0,1,1,0,0,1,1,1,1,1,0,0,0,1,1,0,1,1,1,0,1,...
        0,1,0,0,0,0,1,0,0,1,0,1,1,0,0,1,1,1,1,1,0,0,0,...
        1,1,0,1,1,1,1,0,1,0,1,0,0,0,0,1,0,0,1,0,1,1,0,0,...
        1,1,1,1,1,1,0,0,0,1,1,0];
return;

%scram_bits computes the pseudo-random scrambling sequence
%based on the 9-bit initialization sequence, 111111111 (see NATO standard).
function Z=scram_bits()
%scramblinb bits generated by the following algorithm
%taps=[1,1,1,1,1,1,1,1,1];
%step=3;
%for idx=1:step:176*3
%    Z(idx:idx+step-1)=taps(7:7+step-1);
%    for jdx=1:3
%        temp(1)=xor(taps(9),taps(5));
%        temp(2:9)=taps(1:8);
%        taps=temp;
%    end
%end
Z=[1,1,1,1,1,1,1,1,1,0,0,0,1,0,0,1,1,1,1,1,1,0,1,1,...
      1,0,0,0,1,0,1,0,1,1,1,1,1,0,1,0,0,0,0,0,1,0,0,1,...
      0,0,0,1,0,1,1,0,1,0,1,1,0,1,0,0,0,1,1,1,0,0,1,...
      1,1,1,0,1,1,1,1,0,1,1,0,0,0,0,1,0,1,0,1,0,0,1,...
      0,1,0,0,0,1,1,1,0,0,1,0,1,1,0,1,0,1,0,1,1,1,0,...
      0,0,0,0,1,0,1,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,...
      0,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,1,1,1,0,...
      0,0,1,0,1,0,1,0,0,1,1,0,0,1,0,1,1,1,1,0,1,0,1,...
```

```
    1,0,1,1,0,0,0,1,0,0,1,0,1,0,0,0,1,1,0,0,0,1,0,...
    0,1,1,0,0,1,1,1,1,1,1,1,1,1,0,1,0,0,1,0,0,1,0,...
    0,1,1,0,1,1,0,1,1,1,0,0,1,0,0,1,0,1,1,1,0,1,1,...
    0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,1,1,1,0,0,0,0,1,...
    1,0,1,0,0,0,0,1,1,0,0,1,1,0,1,1,1,1,1,1,1,0,1,...
    0,1,0,0,1,1,0,0,0,0,1,1,1,0,1,0,0,1,1,0,1,0,0,...
    1,1,1,1,0,0,1,1,1,0,0,1,1,1,1,0,1,0,1,1,0,0,1,...
    0,0,0,0,1,1,1,0,1,1,0,1,1,0,1,0,1,1,0,1,1,0,0,...
    0,0,1,0,0,1,1,1,0,1,1,1,1,1,0,1,0,1,0,1,0,1,0,...
    0,0,0,0,0,1,0,1,1,0,0,0,1,0,1,1,1,0,0,1,1,0,1,...
    0,1,1,1,1,1,0,0,0,0,0,0,1,1,1,1,0,0,0,1,1,0,0,...
    1,0,0,1,1,1,1,1,0,1,1,1,0,1,0,1,0,1,0,1,0,0,1,...
    0,0,0,0,0,1,1,0,1,0,0,0,1,1,0,0,0,1,0,1,1,1,1,...
    1,1,0,1,1,0,1,0,0,1,0,1,0,0,0,1,1,0,1,0,1,1,1,...
    1,0,0,0,1,1,0,1,1,1,1,1,1,0,0,1,0,0,0,1,1,1];
return;
```

SIGNS_TSALLISPDF

```
%Sign(s) Tool: SIGNS_TSALLISPDF
%
%[Y,ERR]=SIGNS_TSALLISPDF(X,Q,BETA,CHOICE,P2,P3)
%
%The purpose of this function is to return the tsallis pdf with non-extensivity
%index, Q, and inverse energy factor, BETA, at the values in X.
%
%Mandatory Input Parameters:
%   X -         vector of observations at which to compute the pdf
%   Q -         real-valued index of non-extensivity (0 < Q < 2). Q can
%               also be a vector if CHOICE is 'plot' and BETA is a single
%               element vector.
%   BETA -      mean energy of observations (can be a vector if CHOICE is
%               'plot' and Q is a single element vector)
%
%Optional Input Parameters:
%   CHOICE -    'first' [1],'second' [2],'third' [3],'fourth' [4], or
%               'general'ized [4] choice Tsallis distributions.  The default
%               is 'general'.  CHOICE may also be 'plot' which plots a series
%               of curves of the generalized distribution for various BETA
%               and Q.  Replace Q with 2-Q to get plots of the generalized first
%               choice distribution.
%   P2 -        secondary parameter is either
%               (1) probability normalization factor for 1st choice Tsallis
%                   distribution
%               (2) for 3rd choice Tsallis distribution, an estimate of the
%                   normalized power-law mean defined by
%                        sum ( Pi^Q*Xi )
%                   P2 = ---------------          for i=1:W
%                          sum ( Pi^Q )
%                   where Pi is the probability of each Xi.
%   P3 -        tertiary parameter is the convergence error to which 3rd choice
%               Tsallis distribution is computed
%
%Mandatory Output Parameters:
%   Y -         tsallis pdf.  If CHOICE is 'plot' Y will contain an MxN array of
%               distributions where M is the number of variations in either BETA or
%               Q and where N is the number of observations.  Each row of the MxN
%               array will correspond to the consecutive values of BETA or Q.
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   References:
%      [1]  Tsallis C, "Possible generalization of Boltzmann-Gibbs statistics",
```

```
%            1988 J. Stat. Phys. 52 479
%       [2]  Curado E M F and Tsallis C, "Generalized statistical mechanics:
%            connection with thermodynamics", 1991 J. Phys. A: Math. Gen. 24 L69
%       [3]  Tsallis C, Mendes R S and Plastino A R, "The role of constraints
%            within generalized nonextensive statistics", 1998 Physica A 261 534
%       [4]  G L Ferri1, S Martnez and A Plastino, "Equivalence of the four
%            versions of Tsallis's statistics", J. Stat. Mech. (2005) P04009
%            doi:10.1088/1742-5468/2005/04/P04009
%            PII: S1742-5468(05)97614-0
%Copyright (c) 2005 James Giesbrecht

%Revision History:
%Date         Version   Editor            Changes Made
%-----------  --------  ----------------  --------------------
%
function [Y,ERR]=signs_tsallispdf(X,Q,BETA,CHOICE,P2,P3)
try

    %check input parameters
    if nargin<4
        CHOICE='general';
    end

    if Q<=0 | Q>2
        error('Non-extensivity factor must be:  0 < Q < 2')
    end

    if BETA<0
        error('Mean energy must be non-negative');
    end

    %if find(X<0)
    %    error('X must be a positive vector');
    %end

    %prepare output vector
    Y = zeros(size(X));

    %compute distributions
    switch CHOICE
    case 'first', %Original Tsallis (1988) [1]
        %In the limit...
        %  lim Y = inf  therefore Q cannot equal 1.
        %  Q->1
        %Also by [4] 0 < Q < 2 for valid results
        if Q==1 | Q<=0 | Q>=2
            error('Non-extensivity factor must be 0 < Q < 2 and Q <> 1 for first choice
                Tsallis distribution')
        end
        ALPHA=P2;
```

```
    Y=((1-Q)*(ALPHA+BETA*X)/Q).^(1/(Q-1));
case 'second', %Curado-Tsallis (1991) [2]
    %In the limit...
    %            exp(-B*X)
    %  lim Y = ------------
    %  Q->1     sum(exp(-B*X))
    %
    if Q==1
        Numerator=exp(-BETA*X);
    else
        Numerator=(1-(1-Q)*BETA*X).^(1/(1-Q));
    end
    Zq=sum(Numerator);
    Y=Numerator/Zq;
case 'third', %Tsallis-Mendes-Plastino (1998) [3]
    if Q<=0 | Q>=2
        error('Non-extensivity factor must be 0 < Q < 2 for third choice Tsallis
            distribution')
    end
    Uq=P2;
    E=P3;
    Xqlast=9999999;
    Zqlast=9999999;
    Xq=1;
    Zq=1;
    %In the limit...after Xq and Zq have converged,
    %  lim Y =      exp(-B*(X-Uq)/Xq)
    %  Q->1    ------------------------
    %            sum(exp(-B*(X-Uq)/Xq))
    %Also by [4] 0 < Q < 2 for valid results
    if Q==1
        while abs(Xq-Xqlast)>E | abs(Zq-Zqlast)>E
            Xqlast=Xq;
            Zqlast=Zq;
            Zq=sum(exp(-BETA*(X-Uq)/Xq));
            Xq=Zq^(1-Q);
        end
        Numerator=exp(-BETA*(X-Uq)/Xq);
    else
        while abs(Xq-Xqlast)>E | abs(Zq-Zqlast)>E
            Xqlast=Xq;
            Zqlast=Zq;
            Zq=sum((1-(1-Q)*BETA/Xq*(X-Uq)).^(1/(1-Q)));
            Xq=Zq^(1-Q);
        end
        Numerator=(1-(1-Q)*BETA/Xq*(X-Uq)).^(1/(1-Q));
    end
    Y=Numerator/Zq;
case 'fourth', %Tsallis-Mendes-Plastino-Renyi (2005) [4]
    Uq=P2;
```

```
        %In the limit...after Xq and Zq have converged,
        %  lim Y =      exp(-B*(X-Uq))
        %  Q->1    ------------------------
        %              sum(exp(-B*(X-Uq)))
        if Q==1
            Numerator=exp(-BETA*(X-Uq));
        else
            Numerator=(1-(1-Q)*BETA*(X-Uq)).^(1/(1-Q));
        end
        Zq=sum(Numerator);
        Xq=Zq^(1-Q);
        Y=Numerator/Zq;
    case 'general', %generalized method (2005) [4]
        Numerator=exp_q(-BETA*X,Q);
        Z=sum(Numerator);
        Y=Numerator/Z;
    case 'plot', %plot a series of curves
        if (length(BETA)==1 & length(Q)>=1) | (length(BETA)>=1 & length(Q)==1)
            J=max(length(BETA),length(Q));
            Lstring='legend('; %prepare legend string
            if length(BETA)>1
                for j=1:J
                    Numerator=exp_q(-BETA(j)*X,Q);
                    Z=sum(Numerator);
                    PD=Numerator/Z;
                    Y(j,1:length(X))=PD;
                    Lstring=[Lstring,'''{\beta}=',num2str(BETA(j)),''','];
                end
            else
                for j=1:J
                    Numerator=exp_q(-BETA*X,Q(j));
                    Z=sum(Numerator);
                    PD=Numerator/Z;
                    Y(j,1:length(X))=PD;
                    Lstring=[Lstring,'''Q=',num2str(Q(j)),''','];
                end
            end

            %plot curves
            for j=1:J
                semilogy(X,real(Y(j,:)),getnextlinetype('solid'));
                hold on
            end
            Lstring=[Lstring(1:end-1),',0)']; %add positional information
            title(['Real-valued␣Generalized␣Tsallis␣Distributions␣(',num2str(min(Q)),'␣\leq␣Q␣
                \leq␣',num2str(max(Q)),';',...
                num2str(min(BETA)),'␣\leq␣\beta␣\leq␣',num2str(max(BETA)),')']);
            xlabel('Observation');
            ylabel('Probability');
            eval(Lstring);
```

```
            grid on
            hold off
        else
            error('Both␣BETA␣and␣Q␣cannot␣be␣vectors␣for␣the␣plotting␣option')
        end
    end
    ERR=[];
catch
    %capture error condition
    ERR={dbstack,lasterr};
    Y=[];
end
return

%the q-generalized exponential
function y=exp_q(x,q)
if q==1
    %In the limit...
    %           x
    %  lim y = e
    %  q->1
    %
    y=exp(x);
else
    y=(1+(1-q)*x).^(1/(1-q));
end
return

%function to return a line type for plots with multiple curves
function y=getnextlinetype(TYP)
persistent count
LineColour='bgrcmykbgrcmykbgrcmykbgrcmykbgrcmykbgrcmyk';
LineSample='ox+*sdvx+*sdvo+*sdvox*sdvox+sdvox+*dvox+*svox+*sd';
if isempty(count)
    count=1;
end
%choose plot type
switch TYP
case 'scatter',
    y=[LineColour(count),LineSample(count)];
case 'scatter-solid',
    y=[LineColour(count),LineSample(count),'-'];
case 'scatter-dotted',
    y=[LineColour(count),LineSample(count),':'];
case 'scatter-dashdot',
    y=[LineColour(count),LineSample(count),'-.'];
case 'scatter-dashed',
    y=[LineColour(count),LineSample(count),'--'];
case 'solid',
    y=[LineColour(count),'-'];
```

```
case 'dotted',
    y=[LineColour(count),':'];
case 'dashdot',
    y=[LineColour(count),'-.'];
case 'dashed',
    y=[LineColour(count),'--'];
end
count=mod(count-1,length(LineColour))+2;
if count>length(LineColour)
    count=1;
end
return
```

SIGNS_VARHAM

```
%Sign(s) Tool: SIGNS_VARHAM
%
%[Y,ERR]=SIGNS_VARHAM(RS,HD)
%
%SIGNS_VARHAM returns a random vector of hamming distance HD away from RS.
%
%Mandatory Input Parameters:
%   RS - reference data sequences (a vector of 1s and 0s).  Any non-zero
%        data element is considered a binary 1.
%   HD - a scalar representing the desired distance between S and RS.
%        If HD is larger then the length of RS, then HD superseded by the
%        length of RS.
%
%Optional Input Parameters:
%   None
%
%Mandatory Output Parameters:
%   Y  - binary vector of hamming distance HD away from RS.
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
%Date         Version   Editor             Changes Made
%-----------  -------   ---------------    ---------------------
%
function [Y,ERR]=signs_varham(RS,HD)
try
    ERR=[];
    %adjust HD if necessary
    if length(RS)<=HD
        Y=~abs(sign(RS));
    else
        %randomly choose data elements to invert and make sure
        %that no index is repeated
        index=unique(floor(rand(1,HD)*length(RS)+1));
        while length(index)<HD
            index=unique([index,floor(rand(1,HD)*length(RS)+1)]);
        end

        %ensure that the index vector is no longer than HD by doubling the
```

```
        %length of the vector and then choosing a random startpoint and
        %adding to that HD...this ensures that if the random startpoint
        %is near the end of the index vector that the addition of HD
        %will yield valid indices within 1:length(index)
        %e.g. index = [2 3 5 7 8 9], HD=3
        %     index2=[2 3 5 7 8 9 2 3 5 7 8 9]
        %     if idx=5 then index will become
        %     index=[8 9 2]
        if length(index)>HD
            idx=round(rand(1)*length(index)+1);
            index2=[index,index];
            index=index2(idx:idx+HD);
        end

        %adjust data elements
        Y=abs(sign(RS));
        Y(index)=~Y(index);
    end
catch
    %capture error condition
    ERR={dbstack,lasterr};
    Y=0;
end
return
```

SIGNS_VSIGT

```
%Sign(s) Tool: SIGNS_VSIGT
%
%[SUCC,ERR]=SIGNS_VSIGT(PROFILE,REFTRC,RXTRC,NSEG,OFF,PSE)
%
%The purpose of this function is to plot the time-domain baseband signals
%transmitted by TXSIM and downconverted by RXSIM.
%
%Mandatory Input Parameters:
%   PROFILE -   filename and path of the simulation profile .mat file
%   REFTRC -    index for SIG_REF (i.e. which baseband signal?)
%   RXTRC -     two element vector indicating which SIG_RxBS to plot.  The first
%               element is the row index for SIG_RxBS and the second element is
%               column index for SIG_RxBS.
%   NSEG -      number of segments to plot.  SIGNS_VSIGT automatically determines
%               how many samples to plot in each segment.  The higher this number
%               the fewer the samples per plot and the greater the zoom-in effect.
%   OFF -       integer indicating the number of samples to offset the plotting
%               of SIG_RxBS
%   PSE -       number of seconds to pause between plotting of subsequent
%               portions of the time-domain signal.  I PSE=inf, then a keyboard
%               press is required to move from one time segment to another.
%
%Optional Input Parameters (can appear in any order after mandatory inputs):
%   None
%
%Mandatory Output Parameters:
%   SUCC -      0 if no errors occur, 1 otherwise
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   None
%
%Copyright (c) 2004 James Giesbrecht

%Revision History:
%Date        Version   Editor            Changes Made
%---------   -------   ------------      --------------------

function [SUCC,ERR]=signs_vsigt(PROFILE,REFTRC,RXTRC,NSEG,OFF,PSE)
SUCC=0;
ERR=[];
try
    %load variable files
    load(PROFILE)
```

```
    load(CFG_txsim_outFile)
    load(CFG_rxsim_outFile)
    %choose Trace
    SZ=floor((length(SIG_RxBS{RXTRC(1),RXTRC(2)})-OFF)/NSEG); %get step size
    figure
    %plot the time segments
    if PSE==inf
        REFmin=min(SIG_REF{REFTRC})-.1*abs(min(SIG_REF{REFTRC}));
        REFmax=max(SIG_REF{REFTRC})+.1*abs(max(SIG_REF{REFTRC}));
        RXmin=min(SIG_RxBS{RXTRC(1),RXTRC(2)});
        RXmax=max(SIG_RxBS{RXTRC(1),RXTRC(2)});
        for i=1:SZ:length(SIG_RxBS{RXTRC(1),RXTRC(2)})-OFF
            samples=[i:i+SZ-1];
            subplot(2,1,1), plot(samples,SIG_REF{REFTRC}(samples));
            axis([min(samples),max(samples),REFmin,REFmax]);
            xlabel('Sample␣Number')
            title('Reference␣Baseband␣Signal')
            subplot(2,1,2), plot(samples,SIG_RxBS{RXTRC(1),RXTRC(2)}(samples+OFF));
            axis([min(samples),max(samples),RXmin,RXmax]);
            xlabel('Sample␣Number')
            title('Received␣Baseband␣Signal')
            text(min(samples)+1,RXmin+.25,['Sample␣Offset=',num2str(OFF)])
            pause
        end
    else
        REFmin=min(SIG_REF)-.1*min(SIG_REF);
        REFmax=max(SIG_REF)+.1*max(SIG_REF);
        RXmin=min(SIG_RxBS{RXTRC(1),RXTRC(2)})-.1*min(SIG_RxBS{RXTRC(1),RXTRC(2)});
        RXmax=max(SIG_RxBS{RXTRC(1),RXTRC(2)})+.1*min(SIG_RxBS{RXTRC(1),RXTRC(2)});
        for i=1:SZ:length(SIG_RxBS{RXTRC(1),RXTRC(2)})-OFF
            samples=[i:i+SZ-1];
            subplot(2,1,1), plot(samples,SIG_REF{REFTRC}(samples));
            axis([min(samples),max(samples),REFmin,REFmax]);
            xlabel('Sample␣Number')
            title('Reference␣Baseband␣Signal')
            subplot(2,1,2), plot(samples,SIG_RxBS{RXTRC(1),RXTRC(2)}(samples+OFF));
            axis([min(samples),max(samples),RXmin,RXmax]);
            xlabel('Sample␣Number')
            title('Received␣Baseband␣Signal')
            text(min(samples)+1,RXmin+.25,['Sample␣Offset=',num2str(OFF)])
            pause(PSE)
        end
    end
catch
    %capture error condition
    ERR={dbstack,lasterr};
    SUCC=1;
end
return
```

SIGNS_WIGNER

```
%Sign(s) Tool: SIGNS_WIGNER
%
%[W,ERR]=SIGNS_WIGNER(X,Y,NFFT,NRM,PLT)
%
%The purpose of this function is to compute the time-shift invariant
%and frequency-shift invariant Wigner distribution.  The Wigner distribution
%is defined by
%                 -
%                /
% Wxy(t,f) =    | x(t+m/2)y*(t-m/2)exp(-j2(pi)fm) dm
%              -/
%               m
%
%where t is time, f is frequency, p is pi, and m is the shift. * is the conjugate operator.
%If X=Y then Wxy(t,f) becomes the auto-Wigner Distribution.  In this case the integral of
%Wxy(t,f) over frequency yields instantaneous power of X (i.e. |x(t)|^2) while the integral
%of Wxy(t,f) over time yields spectral energy density of X (i.e. |X(f)|^2).  If X<>Y then
%Wxy(t,f) becomes the cross-Wigner Distribution.  In this case the integral of Wxy(t,f)
%over time yields the cross-spectral density of XY (i.e. |X(f)Y*(f)|) and the integral of
%Wxy(t,f) over frequency yields the instantaneous cross-power of XY (i.e. |x(t)y*(t)|).
%
%In the discrete domain, Wxy(t,f) can be written as
%
%                  N/2
% Wxy(m,k) =  2   sum     x(m+n)y*(m-n)exp(-j2*pi*n/N*k)
%                n=-N/2
%
%Mandatory Input Parameters:
%   X      - a real or complex data vector on which to compute the Wigner distribution.  If
%            the length of X is shorter than the length of Y then Y is truncated to the length
%            of X.
%   Y      - a real or complex data vector on which to compute the Wigner distribution.  If
%            the length of Y is shorter than the length of X then X is truncated to the length
%            of Y.
%
%Optional Input Parameters:
%   NFFT   - number of frequency bins for the distribution (default is 128)
%   NRM    - set to 'normal' to normalize X such that it's energy is unity, by default X
%            is not normalized
%   PLT    - set to 'plot' to plot the distribution, by default SIGNS_WIGNER does not plot
%
%Mandatory Output Parameters:
%  W - contains the Wigner distribution of X and Y.  W is complex and its size is NxP
%      where N is the length of X and P is the length of Y.  Rows of W correspond to the time
%      dimension whereas columns of W correspond to normalized frequency.
%
%Optional Output Parameters:
%   ERR -       two element cell array.  The first element is a structure array
```

```
%               containing the debug trace information, the second element is the
%               error.
%
%Special Notes:
%   None
%
%Copyright (c) 2005 James Giesbrecht

%Company:
%
%Revision History:
%Date        Version   Editor          Changes Made
%-----------  --------  --------------  ---------------------
%
function [W,ERR]=signs_wigner(X,Y,varargin)

ERR=[];
try
    %set default values
    PLT=0;
    NRM=0;
    N=min(length(X),length(Y));
    NFFT=128;
    if nargin>2
        for j=1:length(varargin)
            if ischar(varargin{j})
                if strcmp(varargin{j},'plot') %determine plotting
                    PLT=1;
                end
                if strcmp(varargin{j},'normal') %determine if normalization is required
                    NRM=1;
                end
            end
            if isnumeric(varargin{j})
                NFFT=fix(varargin{j}); %get size of fft
            end
        end
    end
    W=zeros(N,NFFT); %prepare distribution array
    X=reshape(X,1,N); %ensure X is a row vector
    Y=reshape(Y,1,N); %ensure Y is a row vector

    %if necessary normalize X and Y to unity energy
    if NRM
        X=X/sqrt(sum(X.^2));
        Y=Y/sqrt(sum(Y.^2));
    end

    %pad input vectors
    if ~mod(N,2)
```

```
        Xp=[zeros(1,N/2),X,zeros(1,N/2+1)];
        Yp=[zeros(1,N/2),Y,zeros(1,N/2+1)];
        tc=N/2+1; %find the first time index of X and Y in Xp and Yp
    else
        Xp=[zeros(1,(N-1)/2),X,zeros(1,(N+1)/2)];
        Yp=[zeros(1,N/2),Y,zeros(1,N/2+1)];
        tc=(N-1)/2+1;  %find the first time index of X and Y in Xp and Yp
    end

    %compute Wigner Distribution (calculation is incorrect 26/08/05)
    for m=tc:tc+N-1
        idx=m-tc+1;
        W(idx,1:NFFT)=2*fftshift(ifft(Xp((m+1)+[-(tc-1):(tc-1)]).*conj(Yp((m+1)-[-(tc-1):(tc
            -1)])),NFFT));
    end

    %plot distribution if necessary
    if PLT
        figure
        mesh(1:size(W,1),-size(W,2)/2:size(W,2)/2-1,abs(W'));
        xlabel('Time␣Index')
        ylabel('Frequency␣Index')
        title('Magnitude␣of␣Wigner␣Distribution')

        figure
        mesh(1:size(W,1),-size(W,2)/2:size(W,2)/2-1,angle(W'));
        xlabel('Time␣Index')
        ylabel('Frequency␣Index')
        title('Phase␣of␣Wigner␣Distribution␣(rad)')
    end

catch
    %capture error condition
    ERR={dbstack,lasterr};
    W=[];
end
return
```

H.8 Ⓢigns C Routines

The following C routines are called by the **signs_lzw.m** function. The routines are significantly modified from open-source software that performs disk-based LZW compression. These routines, perform LZW compression in **MATLAB**® memory and are capable of compressing 8-bit characters, or 16-bit words. The main function, **memlzw.c**, implements a **MATLAB**® Mex interface so that the functions can be called from the **MATLAB**® command prompt. The second routine, **memlzwengine8.c**, performs compression on 8-bit characters. The last function, **memlzwengine16.c**, performs compression on 16-bit characters. Via C compiler directives, these functions can be recompiled with larger or smaller codeword tables.

MEMLZW

```
/*
%MEMLZW(8/16)
%
%Y=MEMLZW8(X,COMP,F) or Y=MEMLZW16(X,COMP,F)
%
%Matlab gateway function for the purpose of compressing/decompressing a data
%sequence using the Lempel-Ziv-Welch algorithm on 16-bit symbols
%
%Mandatory Input Parameters:
%   X   - data vector to process (can be integers or floats)
%   COMP  - character flag: 'c' to compress, 'u' to uncompress
%   F   - unsigned 8-bit integer flag indicating how to decompress X.  X
%       must be a compressed usigned-8-bit integer sequence.  The following
%       describes the meaning of F during decompression.  F is ignored for
%       for compression, but for the sake of the parameter list it must be
%       one of the following (even though it is not used).
%
%      F  Interpretation
%         --  ----------------
%      0  X will be expanded to a sequence of double-precision floats.  This is the default.
%
%      1  X will be expanded to a sequence of unsigned 8-bit integers.
%
%         2  X will be expanded to a sequence of signed 8-bit integers.
%
%         3  X will be expanded to a sequence of unsigned 16-bit integers.
%
%      4  X will be expanded to a sequence of signed 16-bit integers.
%
```

```
%Optional Input Parameters:
%    None
%
%Mandatory Output Parameters:
%    Y - contains the compressed data sequence or the decompressed data sequence
%        according to the interpretation indicated by F.
%
%Optional Output Parameters:
%    None
%
%Special Notes:
% Original code was written by Mark R. Nelson (Copyright (c) 1989) and
% modified to accommodate operation in the MATLAB environment by James Giesbrecht
% (Copyright (c) 2004).
%
%Copyright (c) 2003 James Giesbrecht

%Company:
%
%Revision History:
%Date         Version   Editor             Changes Made
%----------   -------   ------------       --------------------
%
*/

//Pre-processor definitions
//#define DEBUGON //"comment out" this line if debug output is not required.
//#define BIT16 //"comment out" this line if compiling for 8bit symbols.  THIS IS NOW INCLUDED
      IN THE COMMAND LINE FOR THE COMPILER

//Included definition files
#include "mex.h"
#include "stdio.h"
#include "string.h"
#include "stdlib.h"

//Function prototypes
#ifndef BIT16
  #define SYM 1 //one byte per symbol
  typedef unsigned char symbol;
  #ifdef TAB8191
    #include "memlzwengine8_8191.c" //this version has a 8191 element table
  #else
    #ifdef LZW13
      #include "memlzwengine8_13.c" //this version has a 9029 element table and 13-bit codes
    #else
      #include "memlzwengine8.c" //this version has a 5021 element table
    #endif
  #endif
#else
```

```
  #define SYM 2 //two bytes per symbol
  typedef unsigned short int symbol;
  #include "memlzwengine16.c"
#endif

//Error Messages
#define SYNTAX "Syntax:␣␣<output␣variable>␣=␣function(<numeric␣vector>,<'c'␣or␣'u'>,<8-bit␣
    integer␣flag:␣0␣to␣4>)\n"
#define INVDAT "Invalid␣datatype␣for␣input␣array\n"
#define MEMCMP "Out␣of␣memory␣for␣LZW␣compression\n"
#define INVD08 "Input␣array␣must␣be␣uint8␣for␣decompression\n"
#define INVD16 "Input␣array␣must␣be␣uint16␣for␣decompression\n"
#define MEMEXP "Out␣of␣memory␣for␣LZW␣expansion\n"

/*****************************************************************************************
    */
//MATLAB 6p1 interface
/*****************************************************************************************
    */
void mexFunction(int NumOut, mxArray *OutArray[], int NumIn, const mxArray *InArray[])
{
  //function prototypes
  unsigned int lzwengine(const symbol *ptrIN, unsigned int INlength, symbol *ptrOUT, char mode
      );

  //define local variables
  symbol *OUTchptr;      //pointer used to point to symbol data in the output array
  symbol *memptr;        //pointer used to point to allocated symbol array
  symbol *cptr;         //temporary symbol pointer for allocated memory
  char modeptr[2];         //pointer used to address the compress/decompress character flag
  char *flagptr;         //pointer to the interpretation flag
  unsigned int ROWS;       //number of rows in input array
  unsigned int COLS;       //number of columns in input array
  unsigned int Xlen;       //number of elements in input array (product of ROWS and COLS)
  unsigned int length;     //number of elements in the compressed sequence
  unsigned int i;         //loop counter(s)

  //define constants
  const int INMAX=3;     //maximum number of parameters in the input array
  const int OUTMAX=1;    //maximum number of parameters in the output array
  const int FLAGMX=4;    //maximum value of the flag parameter
  const int FLAGMN=0;    //minimum valued of the flag parameter
  const int BUFX=2;    //buffer size multiplier for compression
  const int BUFXD=10;    //buffer size multiplier for decompression

  //check for valid input and output variables
  if( (NumIn!=INMAX) \
    || (NumOut>OUTMAX) \
    || !mxIsNumeric(InArray[0]) \
    || mxIsEmpty(InArray[0]) \
```

```
    || ( (mxGetM(InArray[0])>1) && (mxGetN(InArray[0])>1) ) \
    || mxIsNumeric(InArray[1]) \
    || mxIsEmpty(InArray[1]) \
    || ( (*(char *)mxGetData(InArray[1])!='c') && (*(char *)mxGetData(InArray[1])!='u') ) \
    || !mxIsNumeric(InArray[2]) \
    || !mxIsUint8(InArray[2]) \
    || (mxIsUint8(InArray[2]) && ((int)*(unsigned char *)mxGetData(InArray[2])>FLAGMX)) )
{
  mexErrMsgTxt(SYNTAX);
}

//get input parameters
ROWS = mxGetM(InArray[0]);        //number of rows
COLS = mxGetN(InArray[0]);        //number of columns
Xlen = ROWS*COLS;            //number of elements in the vector
mxGetString(InArray[1],modeptr,2);  //pointer to mode selection (i.e. 'c'ompress or 'u'
    ncompress)
flagptr = mxGetData(InArray[2]);  //pointer to data interpretion flag

//choose compression or expansion
switch (*modeptr)
{
//compress
case 'c':

  //check valid classes
  switch (mxGetClassID(InArray[0]))
  {
  case mxDOUBLE_CLASS:
  case mxINT8_CLASS:
  case mxUINT8_CLASS:
  case mxINT16_CLASS:
  case mxUINT16_CLASS:
    break;
  default:
    mexErrMsgTxt(INVDAT);
  }

  //allocate memory
  if((memptr=mxCalloc(Xlen*BUFX,mxGetElementSize(InArray[0])))==NULL)
    mexErrMsgTxt(MEMCMP);

  //compress the data
  cptr=memptr;
  length=lzwengine(mxGetData(InArray[0]), (Xlen*mxGetElementSize(InArray[0]))>>(SYM-1), cptr
      , 'c');

  //allocate memory for Output Matrix
  if (ROWS==1)
  {
```

```
#ifndef BIT16
        OutArray[0]=mxCreateNumericMatrix(ROWS,length,mxUINT8_CLASS,mxREAL);
#else
        OutArray[0]=mxCreateNumericMatrix(ROWS,length,mxUINT16_CLASS,mxREAL);
#endif
    }
    else
    {
#ifndef BIT16
        OutArray[0]=mxCreateNumericMatrix(length,COLS,mxUINT8_CLASS,mxREAL);
#else
        OutArray[0]=mxCreateNumericMatrix(length,COLS,mxUINT16_CLASS,mxREAL);
#endif
    }

    //copy result into output matrix
    OUTchptr=mxGetData(OutArray[0]);
    cptr=memptr;
    for(i=0; i<length; i++)
      *OUTchptr++ = *cptr++;

    //cleanup
    mxFree(memptr);
    break;

  //decompress
  case 'u':

#ifndef BIT16
    //check that input array is uint8 array
    if(!mxIsUint8(InArray[0]))
      mexErrMsgTxt(INVD08);
#else
    //check that input array is uint16 array
    if(!mxIsUint16(InArray[0]))
      mexErrMsgTxt(INVD16);
#endif

    switch (*flagptr)
    {
    case 0: //decompress to double precision floating point
      //Allocate memory for decompressed array
      if((memptr=mxCalloc(Xlen,sizeof(double)))==NULL)
        mexErrMsgTxt(MEMEXP);

      //expand the data
      cptr=memptr;
      length=lzwengine(mxGetData(InArray[0]), Xlen, cptr, 'u');

      //Allocate memory for double Output Matrix
```

```
    if (ROWS==1)
      OutArray[0]=mxCreateDoubleMatrix(ROWS,(length*SYM)>>3,mxREAL);
    else
      OutArray[0]=mxCreateDoubleMatrix((length*SYM)>>3,COLS,mxREAL);

    //copy result into output matrix
    OUTchptr = mxGetData(OutArray[0]);
    cptr=memptr;
    for(i=0; i<length; i++)
      *OUTchptr++= *cptr++;

    //cleanup
    mxFree(memptr);
    break;

  case 1: //decompress to 8-bit integer
  case 2:
    //Allocate memory for decompressed array
    if((memptr=mxCalloc((Xlen*BUFXD)<<(SYM-1),sizeof(char)))==NULL)
      mexErrMsgTxt(MEMEXP);

    //expand the data
    cptr=memptr;
    length=lzwengine(mxGetData(InArray[0]), Xlen, cptr, 'u');

    //Allocate memory for int8 Output Matrix
    if (ROWS==1)
    {
      //determine what type of output array
      switch (*flagptr)
      {
      case 1: //unsigned 8-bit integers
        OutArray[0]=mxCreateNumericMatrix(ROWS,length<<(SYM-1),mxUINT8_CLASS,mxREAL);
        break;
      case 2: //signed 8-bit integers
        OutArray[0]=mxCreateNumericMatrix(ROWS,length<<(SYM-1),mxINT8_CLASS,mxREAL);
      }
    }
    else
    {
      //determine what type of output array
      switch (*flagptr)
      {
      case 1: //unsigned 8-bit integers
        OutArray[0]=mxCreateNumericMatrix(length<<(SYM-1),COLS,mxUINT8_CLASS,mxREAL);
        break;
      case 2: //signed 8-bit integers
        OutArray[0]=mxCreateNumericMatrix(length<<(SYM-1),COLS,mxINT8_CLASS,mxREAL);
      }
    }
```

```
    //copy result into output matrix
    OUTchptr = mxGetData(OutArray[0]);
    cptr=memptr;
    for(i=0; i<length; i++)
      *OUTchptr++= *cptr++;

    //cleanup
    mxFree(memptr);
    break;
  case 3: //decompress to 16-bit integers
  case 4:
    //Allocate memory for decompressed array
    if((memptr=mxCalloc((Xlen*BUFXD),sizeof(short int)))==NULL)
      mexErrMsgTxt(MEMEXP);

    //expand the data
    cptr=memptr;
    length=lzwengine(mxGetData(InArray[0]), Xlen, cptr, 'u');

    //Allocate memory for int16 Output Matrix
    if (ROWS==1)
    {
      //determine what type of output array
      switch (*flagptr)
      {
      case 3: //unsigned 16-bit integers
        OutArray[0]=mxCreateNumericMatrix(ROWS, length>>(2-SYM),mxUINT16_CLASS,mxREAL);
        break;
      case 4: //signed 16-bit integers
        OutArray[0]=mxCreateNumericMatrix(ROWS, length>>(2-SYM),mxINT16_CLASS,mxREAL);
      }
    }
    else
    {
      //determine what type of output array
      switch (*flagptr)
      {
      case 3: //unsigned 16-bit integers
        OutArray[0]=mxCreateNumericMatrix(length>>(2-SYM),COLS,mxUINT16_CLASS,mxREAL);
        break;
      case 4: //signed 16-bit integers
        OutArray[0]=mxCreateNumericMatrix(length>>(2-SYM),COLS,mxINT16_CLASS,mxREAL);
      }
    }

    //copy result into output matrix
    OUTchptr=mxGetData(OutArray[0]);
    cptr=memptr;
    for(i=0; i<length; i++)
```

```
            *OUTchptr++= *cptr++;

          //cleanup
          mxFree(memptr);
          break;
        default:
          mexErrMsgTxt(SYNTAX);
          break;
        }
        break;
      default:
        mexErrMsgTxt(SYNTAX);
        break;
      }
}
```

MEMLZWENGINE8

```
/*
%LZWENGINE8
%
%Y=LZWENGINE8(PTRIN,INLENGTH,PTROUT,MODE)
%i
%The purpose of this function is to compress/decompress a data
%sequence using the Lempel-Ziv-Welch algorithm.  This function is the main processing
%subroutine for MEMLZW.
%
%Mandatory Input Parameters:
%   PTRIN - void pointer to the data vector to process(can be integers or floats)
%   INLENGTH- the length of the input array
%   PTROUT  - pointer to the output array
%   MODE  - 'c' for compression, or 'u' for uncompress/decompress
%
%Optional Input Parameters:
%   None
%
%Mandatory Output Parameters:
%   Y - Compression: the number of elements in the compressed sequence.
%   Decompression: the number of elements in the decompressed sequence.
%   OR: EOF_FLAG if either compression or decompression fails.
%
%Optional Output Parameters:
%   None
%
%Special Notes:
% Original was written by Mark R. Nelson (Copyright (c) 1989) and
% modified to accommodate operation in the MATLAB environment by James Giesbrecht
% (Copyright (c) 2003, 2004, 2006).

%Company:
%
%Revision History:
%Date        Version   Editor            Changes Made
%---------   -------   -------------     --------------------
%2003/4       1.1      J. Giesbrecht     modification for use in Matlab 6.1
%2006         2.0      J. Giesbrecht     modification to handle multi-byte symbols
*/

/*****************************************************************************/

//complier defines
#define SYMb 8*SYM      //number of bits in each symbol
#define BITS 12                     /* Setting the number of bits to 12, 13*/
#define HASHING_SHIFT BITS-SYMb     /* or 14 affects several constants.     */
#define MAX_VALUE (1 << BITS) - 1 /* Note that MS-DOS machines need to    */
#define MAX_CODE MAX_VALUE - 1      /* compile their code in large model if*/
```

```
                                /* 14 bits are selected.              */
#define EOF_FLAG 9999

//The string table size needs to be a prime number that is somewhat larger
//than 2**BITS.
#if BITS == 14
  #define TABLE_SIZE 18041 //suggested by M. Nelson
#endif
#if BITS == 13
  #define TABLE_SIZE 9029 //suggested by M. Nelson
#endif
#if BITS <= 12
  #define TABLE_SIZE 5021 //suggested by M. Nelson
#endif

//Error Messages
#define LZWENG_SPACE "lzwengine: Fatal error allocating table space\n"
#define LZWENG_SYNTX "lzwengine: Unknown command\n"
#define LZWENG_ERREX "lzwengine: Fatal error during code expansion\n"

//global variables for all multi-byte compression/decompression routines
unsigned int *code_value;             /* This is the code value array        */
unsigned int *prefix_code;       /* This array holds the prefix codes   */

//global variables for 8-bit compression/decompression
symbol *append_character;  /* This array holds the appended chars */
symbol decode_stack[4000]; /* This array holds the decoded string */

/*******************************************************************************/
//Main function for the lzwengine
/*******************************************************************************/
unsigned int lzwengine(const symbol *ptrIN, unsigned int INlength, symbol *ptrOUT, char mode)
{
  //8-bit function prototypes
  unsigned int compressc(const symbol *fidIN, unsigned int INlength, symbol *fidOUT);
  unsigned int expandc(const symbol *fid, unsigned int INlength, symbol *fidOUT);
  void memputc(const symbol ch, symbol *outptr);
  unsigned int memgetc(const symbol *inptr, unsigned int length, bool clr);

  // variables
  unsigned int return_value;

  /*
  **  These three buffers are needed for the compression/expansion phase.
  */
  code_value=mxCalloc(TABLE_SIZE,sizeof(unsigned int));
  prefix_code=mxCalloc(TABLE_SIZE,sizeof(unsigned int));
  append_character=mxCalloc(TABLE_SIZE,sizeof(unsigned int));
  if (code_value==NULL || prefix_code==NULL || append_character==NULL)
    mexErrMsgTxt(LZWENG_SPACE);
```

```
  //compress or decompress data
  switch(mode)
  {
  case 'c': //compress data
    return_value=compressc(ptrIN, INlength, ptrOUT);
    break;
  case 'u': //decompress data
    return_value=expandc(ptrIN, INlength, ptrOUT);
    break;
  default:
    mexErrMsgTxt(LZWENG_SYNTX);
    break;
  };
  mxFree(code_value);
  mxFree(prefix_code);
  mxFree(append_character);
  return(return_value);
}

/*****************************************************************************************/
//Compression functions
/*****************************************************************************************/
unsigned int compressc(const symbol *fidIN, unsigned int INlength, symbol *fidOUT)
{
  //function prototypes
  unsigned int char_findmatch(unsigned int hash_prefix,unsigned int hash_character);
  unsigned int char_outcode(symbol **ptrfidOUT, unsigned int code, bool clr);

  //variables
  unsigned int next_code;
  unsigned int character;
  unsigned int string_code;
  unsigned int index;
  unsigned int count=0, i;

  next_code=256;              /* Next code is the next available string code*/
  for (i=0;i<TABLE_SIZE;i++)  /* Clear out the string table before starting */
    code_value[i]=-1;
  memgetc(fidIN,INlength,1); //initialize memgetc
  string_code=memgetc(fidIN++,INlength,0);    /* Get the first code    */

  /*
  ** This is the main loop where it all happens.  This loop runs util all of
  ** the input has been exhausted.  Note that it stops adding codes to the
  ** table after all of the possible codes have been defined.
  */
  while ((character=memgetc(fidIN++,INlength,0)) != EOF_FLAG)
  {
    index=char_findmatch(string_code,character);// See if the string is in
```

```
    if (code_value[index] != -1)            // the table.  If it is,
      string_code=code_value[index];       // get the code value.  If
    else                                    // the string is not in the
    {                                       // table, try to add it.
      if (next_code <= MAX_CODE)
      {
        code_value[index]=next_code++;
        prefix_code[index]=string_code;
        append_character[index]=character;
      }
      count+=char_outcode(&fidOUT,string_code,0);  // When a string is found
      string_code=character;              // that is not in the table
    }                                     // I output the last string
  }                                       // after adding the new one

  count+=char_outcode(&fidOUT,string_code,0); // Output the last code
  count+=char_outcode(&fidOUT,MAX_VALUE,0);    // Output the end of buffer code
  count+=char_outcode(&fidOUT,0,0);            // This code flushes the output buffer
  count+=char_outcode(&fidOUT,0,1);       // This code clears the static variables in
      char_outcode
  return(count);
}

// This is the hashing routine.  It tries to find a match for the prefix+char
// string in the string table.  If it finds it, the index is returned.  If
// the string is not found, the first available index in the string table is
// returned instead.
unsigned int char_findmatch(unsigned int hash_prefix,unsigned int hash_character)
{
  int index;
  int offset;
  index = (hash_character << HASHING_SHIFT) ^ hash_prefix;
  if (index == 0)
    offset = 1;
  else
    offset = TABLE_SIZE - index;
  while (1)
  {
    if (code_value[index] == -1)
        return(index);
    if (prefix_code[index] == (unsigned) hash_prefix && append_character[index] == (unsigned)
        hash_character)
        return(index);
    index -= offset;
    if (index < 0)
      index += TABLE_SIZE;
  }
  return(0);
}
```

```
//This function get the next correct output code for the compressed sequence
unsigned int char_outcode(symbol **ptrfidOUT, unsigned int code, bool clr)
{
  static int output_bit_count=0;
  static unsigned int output_bit_buffer=0;
  unsigned int charcount=0;
  symbol *fidOUT;

  fidOUT=*ptrfidOUT;  //get the address of next output location
  if(clr)
  {
    output_bit_count=0;
    output_bit_buffer=0;
  }
  else
  {
    output_bit_buffer |= code << (8*sizeof(int)-BITS-output_bit_count);
    output_bit_count += BITS;
    while (output_bit_count >= SYMb)
    {
      memputc((symbol)(output_bit_buffer >> (8*sizeof(int)-SYMb)),fidOUT++);
      charcount++;
      output_bit_buffer <<= SYMb;
      output_bit_count -= SYMb;
    }
  }
  *ptrfidOUT=fidOUT;
  return(charcount);
}

/*****************************************************************************/
//Expansion functions for 8-bit symbols
/*****************************************************************************/
unsigned int expandc(const symbol *fid, unsigned int INlength, symbol *fidOUT)
{
  //function prototypes
  symbol *char_decode(symbol *buffer,unsigned int code);
  unsigned int char_incode(symbol **ptrfidIN, unsigned int INlength, bool clr);

  //variables
  unsigned int next_code;
  unsigned int new_code;
  unsigned int old_code;
  unsigned int character, count=0;
  symbol *string, *fidIN;

  //initialize memgetc
  fidIN=fid;
  memgetc(fidIN,INlength,1);
```

```
next_code=256;              // This is the next available code to define
old_code=char_incode(&fidIN, INlength, 0);  // Read in the first code, initialize the
character=old_code;           // character variable, and send the first
memputc((symbol)old_code,fidOUT++);        // code to the output file
count++;

/*
**  This is the main expansion loop.  It reads in characters from the LZW file
**  until it sees the special code used to inidicate the end of the data.
*/
while ((new_code=char_incode(&fidIN, INlength, 0)) != MAX_VALUE)
{
  /*
  ** This code checks for the special STRING+CHARACTER+STRING+CHARACTER+STRING
  ** case which generates an undefined code.  It handles it by decoding
  ** the last code, and adding a single character to the end of the decode string.
  */
  if (new_code>=next_code)
  {
    *decode_stack=character;
    string=char_decode(decode_stack+1,old_code);
  }
  /*
  ** Otherwise we do a straight decode of the new code.
  */
  else
    string=char_decode(decode_stack,new_code);
  /*
  ** Now we output the decoded string in reverse order.
  */
  character=*string;
  while (string >= decode_stack)
  {
    memputc(*string--,fidOUT++);
    count++;
  }
  /*
  ** Finally, if possible, add a new code to the string table.
  */
  if (next_code <= MAX_CODE)
  {
    prefix_code[next_code]=old_code;
    append_character[next_code]=character;
    next_code++;
  }
  old_code=new_code;
}
//clear the static variables in char_incode
char_incode(&fidIN, INlength, 1);
return(count);
```

```
}

// This routine simply decodes a string from the string table, storing
// it in a buffer.  The buffer can then be output in reverse order by
// the expansion program.
symbol *char_decode(symbol *buffer,unsigned int code)
{
  int i;
  i=0;
  while (code > 255)
  {
    *buffer++ = append_character[code];
    code=prefix_code[code];
    if (i++>=MAX_CODE)
      mexErrMsgTxt(LZWENG_ERREX);
  }
  *buffer=code;
  return(buffer);
}

// The following two routines are used to output variable length
// codes.  They are written strictly for clarity, and are not
// particularly efficient.

unsigned int char_incode(symbol **ptrfidIN, unsigned int INlength, bool clr)
{
  unsigned int return_value;
  static int input_bit_count=0;
  static unsigned int input_bit_buffer=0;
  unsigned int inputcharacter;
  symbol *fidIN;

  fidIN=*ptrfidIN;
  if(clr)
  {
    input_bit_count=0;
    input_bit_buffer=0;
    return_value=0;
  }
  else
  {
    while ((input_bit_count <= 8*sizeof(int)-SYMb))
    {
      inputcharacter=memgetc(fidIN,INlength,0);
      if(inputcharacter!=EOF_FLAG)
      {
        fidIN++;
        input_bit_buffer |= inputcharacter << (8*sizeof(int)-SYMb-input_bit_count);
        input_bit_count += SYMb;
      }
```

```
      else
        break;
    }
    return_value=input_bit_buffer >> (8*sizeof(int)-BITS);
    input_bit_buffer <<= BITS;
    input_bit_count -= BITS;
  }
  *ptrfidIN=fidIN;
  return(return_value);
}

/*****************************************************************************/
//Common functions accessed by compression and decompression functions
/*****************************************************************************/

// memgetc is similar to getc except that it reads an element from memory
unsigned int memgetc(const symbol *inptr, unsigned int length, bool clr)
{
  static unsigned int count=0;
  unsigned int return_value;

  if (clr)
  {
    return_value= EOF_FLAG;
    count=0;
  }
  else
  {
    if(count<length)
    {
      count++;
      return_value=*inptr;
    }
    else
      return_value= EOF_FLAG;
  }
  return(return_value);
}

//memputc is similar to putc except that it writes a character to memory
void memputc(const symbol ch, symbol *outptr)
{
  *outptr=ch;
}
```

MEMLZWENGINE16

```
/*
%LZWENGINE16
%
%Y=LZWENGINE16(PTRIN,INLENGTH,PTROUT,MODE)
%i
%The purpose of this function is to compress/decompress a data
%sequence using the Lempel-Ziv-Welch algorithm.  This function is the main processing
%subroutine for MEMLZW.
%
%Mandatory Input Parameters:
%    PTRIN - void pointer to the data vector to process(can be integers or floats)
%    INLENGTH- the length of the input array
%    PTROUT  - pointer to the output array
% MODE  - 'c' for compression, or 'u' for uncompress/decompress
%
%Optional Input Parameters:
%    None
%
%Mandatory Output Parameters:
%    Y - Compression: the number of elements in the compressed sequence.
%    Decompression: the number of elements in the decompressed sequence.
%    OR: EOF_FLAG if either compression or decompression fails.
%
%Optional Output Parameters:
%    None
%
%Special Notes:
% Original code was written by Mark R. Nelson (Copyright (c) 1989) and
% modified to accommodate operation in the MATLAB environment by James Giesbrecht
% (Copyright (c) 2003, 2004, 2006).

%Company:
%
%Revision History:
%Date         Version   Editor             Changes Made
%---------    -------   ------------       --------------------
 %2003/4       1.1       J. Giesbrecht      modification for use in Matlab 6.1
 %2006         2.0       J. Giesbrecht      modification to handle multi-byte symbols
 */

/*****************************************************************************/

 //complier defines
#define SYMb 8*SYM            //number of bits in each symbol (SYM defined in MEMLZW)
#define BITS 20               //number of code bits
#define HASHING_SHIFT BITS-SYMb    //shift for hash function
#define MAX_VALUE (1 << BITS) - 1  //maximum symbol
#define MAX_CODE MAX_VALUE - 1     //maximum code value
```

```
#define EOF_FLAG 0x99999999      //end of stream marker

//The string table size needs to be a prime number that is somewhat larger
//than 2**BITS.  The primes below were chosen using Matlab:
//A=primes(2^(BITS+.5));min(A(find(A>1000+2^BITS)))
#if BITS == 22
  #define TABLE_SIZE 4195307
#endif
#if BITS == 21
  #define TABLE_SIZE 2098153
#endif
#if BITS <= 20
  #define TABLE_SIZE 1049599
#endif

//Error Messages
#define LZWENG_SPACE "lzwengine: Fatal error allocating table space\n"
#define LZWENG_SYNTX "lzwengine: Unknown command\n"
#define LZWENG_ERREX "lzwengine: Fatal error during code expansion\n"

//global variables for all multi-byte compression/decompression routines
unsigned int *code_value;     /* This is the code value array        */
unsigned int *prefix_code;    /* This array holds the prefix codes   */

//global variables for 16-bit compression/decompression
symbol *append_character; /* This array holds the appended chars */
symbol decode_stack[4000];  /* This array holds the decoded string */

/*****************************************************************************/
//Main function for the lzwengine
/*****************************************************************************/
unsigned int lzwengine(const symbol *ptrIN, unsigned int INlength, symbol *ptrOUT, char mode)
{
  //16-bit function prototypes
  unsigned int compressc(const symbol *fidIN, unsigned int INlength, symbol *fidOUT);
  unsigned int expandc(const symbol *fid, unsigned int INlength, symbol *fidOUT);
  void memputc(const symbol ch, symbol *outptr);
  unsigned int memgetc(const symbol *inptr, unsigned int length, bool clr);
  void lsr_long(unsigned int *longword, unsigned int N);
  void lsl_long(unsigned int *longword, unsigned int N);

  // variables
  unsigned int return_value;

  /*
  **  These three buffers are needed for the compression/expansion phase.
  */
  code_value=mxCalloc(TABLE_SIZE,sizeof(unsigned int));
  prefix_code=mxCalloc(TABLE_SIZE,sizeof(unsigned int));
  append_character=mxCalloc(TABLE_SIZE,sizeof(unsigned int));
```

```
  if (code_value==NULL || prefix_code==NULL || append_character==NULL)
    mexErrMsgTxt(LZWENG_SPACE);

  //compress or decompress data
  switch(mode)
  {
  case 'c': //compress data
    return_value=compressc(ptrIN, INlength, ptrOUT);
    break;
  case 'u': //decompress data
    return_value=expandc(ptrIN, INlength, ptrOUT);
    break;
  default:
    mexErrMsgTxt(LZWENG_SYNTX);
  };
  mxFree(code_value);
  mxFree(prefix_code);
  mxFree(append_character);
  return(return_value);
}

/*****************************************************************************/
//Compression functions
/*****************************************************************************/
unsigned int compressc(const symbol *fidIN, unsigned int INlength, symbol *fidOUT)
{
  //function prototypes
  unsigned int char_findmatch(unsigned int hash_prefix,unsigned int hash_character);
  unsigned int char_outcode(symbol **fidOUT, unsigned int code, bool clr);

  //variables
  unsigned int next_code;
  unsigned int character;
  unsigned int string_code;
  unsigned int index;
  unsigned int count=0, i;

  next_code=65536;            /* Next code is the next available string code*/
  for (i=0;i<TABLE_SIZE;i++)  /* Clear out the string table before starting */
    code_value[i]=-1;
  memgetc(fidIN,INlength,1); //initialize memgetc
  string_code=memgetc(fidIN++,INlength,0);    /* Get the first code    */

  /*
  ** This is the main loop where it all happens.  This loop runs until all of
  ** the input has been exhausted.  Note that it stops adding codes to the
  ** table after all of the possible codes have been defined.
  */
  while ((character=memgetc(fidIN++,INlength,0)) != EOF_FLAG)
  {
```

```
    index=char_findmatch(string_code,character);// See if the string is in
    if (code_value[index] != -1)              // the table.  If it is,
      string_code=code_value[index];        // get the code value.  If
    else                                      // the string is not in the
    {                                         // table, try to add it.
      if (next_code <= MAX_CODE)
      {
        code_value[index]=next_code++;
        prefix_code[index]=string_code;
        append_character[index]=character;
      }
      count+=char_outcode(&fidOUT,string_code,0);  // When a string is found
      string_code=character;                // that is not in the table
    }                                       // I output the last string
  }                                         // after adding the new one

  count+=char_outcode(&fidOUT,string_code,0); // Output the last code
  count+=char_outcode(&fidOUT,MAX_VALUE,0);   // Output the end of buffer code
  count+=char_outcode(&fidOUT,0,0);           // This code flushes the output buffer
  count+=char_outcode(&fidOUT,0,1);       // This code clears the static variables in
      char_outcode
  return(count);
}

// This is the hashing routine.  It tries to find a match for the prefix+char
// string in the string table.  If it finds it, the index is returned.  If
// the string is not found, the first available index in the string table is
// returned instead.
unsigned int char_findmatch(unsigned int hash_prefix,unsigned int hash_character)
{
  int index;
  int offset;
  index = (hash_character << HASHING_SHIFT) ^ hash_prefix;
  if (index == 0)
    offset = 1;
  else
    offset = TABLE_SIZE - index;
  while (1)
  {
    if (code_value[index] == -1)
      return(index);
    if (prefix_code[index] == (unsigned) hash_prefix && append_character[index] ==
        hash_character)
      return(index);
    index -= offset;
    if (index < 0)
      index += TABLE_SIZE;
  }
  return(0);
}
```

```
//This function gets the next correct output code for the compressed sequence using
//unsigned long long int for the bit buffer
unsigned int char_outcode(symbol **ptrfidOUT, unsigned int code, bool clr)
{
  static int output_bit_count=0;
  static unsigned int output_bit_buffer[2];
  unsigned int charcount=0;
  symbol *fidOUT;
  unsigned int lcode[2];

  fidOUT=*ptrfidOUT;  //get the address of next output location
  if(clr)
  {
    output_bit_count=0;
    output_bit_buffer[0]=0;
    output_bit_buffer[1]=0;
  }
  else
  {
    lcode[0]=code;
    lcode[1]=0;
    lsl_long(&lcode[0],8*sizeof(int)*2-BITS-output_bit_count);
    output_bit_buffer[1] |= lcode[1];
    output_bit_buffer[0] |= lcode[0];
    output_bit_count += BITS;
    while (output_bit_count >= SYMb)
    {
      lcode[0]=output_bit_buffer[0];
      lcode[1]=output_bit_buffer[1];
      lsr_long(&lcode[0],8*sizeof(int)*2-SYMb);
      memputc((symbol)lcode[0],fidOUT++);
      charcount++;
      lsl_long(&output_bit_buffer[0],SYMb);
      output_bit_count -= SYMb;
    }
  }
  *ptrfidOUT=fidOUT;
  return(charcount);
}

/*****************************************************************************/
//Expansion functions for 16-bit symbols
/*****************************************************************************/
unsigned int expandc(const symbol *fid, unsigned int INlength, symbol *fidOUT)
{
  //function prototypes
  symbol *char_decode(symbol *buffer,unsigned int code);
  unsigned int char_incode(symbol **fidIN, unsigned int INlength, bool clr);
```

```
//variables
unsigned int next_code;
unsigned int new_code;
unsigned int old_code;
unsigned int character, count=0;
symbol *string, *fidIN;

//initialize memgetc
fidIN=fid;
memgetc(fidIN,INlength,1);

next_code=65536;            // This is the next available code to define
old_code=char_incode(&fidIN, INlength, 0);  // Read in the first code, initialize the
character=old_code;          // character variable, and send the first
memputc((symbol)old_code,fidOUT++);       // code to the output file
count++;

/*
**  This is the main expansion loop.  It reads in characters from the LZW file
**  until it sees the special code used to indicate the end of the data.
*/
while ((new_code=char_incode(&fidIN, INlength, 0)) != MAX_VALUE)
{
  /*
  ** This code checks for the special STRING+CHARACTER+STRING+CHARACTER+STRING
  ** case which generates an undefined code.  It handles it by decoding
  ** the last code, and adding a single character to the end of the decode string.
  */
  if (new_code>=next_code)
  {
    *decode_stack=character;
    string=char_decode(decode_stack+1,old_code);
  }
  /*
  ** Otherwise we do a straight decode of the new code.
  */
  else
    string=char_decode(decode_stack,new_code);
  /*
  ** Now we output the decoded string in reverse order.
  */
  character=*string;

  while (string >= decode_stack)
  {
    memputc(*string--,fidOUT++);
    count++;
  }
  /*
  ** Finally, if possible, add a new code to the string table.
```

```
    */
    if (next_code <= MAX_CODE)
    {
      prefix_code[next_code]=old_code;
      append_character[next_code]=character;
      next_code++;
    }
    old_code=new_code;

  }
  //clear the static variables in char_incode
  char_incode(&fidIN, INlength, 1);
  return(count);
}

// This routine simply decodes a string from the string table, storing
// it in a buffer.  The buffer can then be output in reverse order by
// the expansion program.
symbol *char_decode(symbol *buffer,unsigned int code)
{
  int i;
  i=0;
  while (code > 65535)
  {
    *buffer++ = append_character[code];
    code=prefix_code[code];
    if (i++>=MAX_CODE)
      mexErrMsgTxt(LZWENG_ERREX);
  }
  *buffer=code;
  return(buffer);
}

// The following two routines are used to output variable length
// codes.  They are written strictly for clarity, and are not
// particularly efficient.

unsigned int char_incode(symbol **ptrfidIN, unsigned int INlength, bool clr)
{
  unsigned int return_value;
  static int input_bit_count=0;
  static unsigned int input_bit_buffer[2];
  unsigned int lcode[2];
  unsigned int inputcharacter;
  symbol *fidIN;

  fidIN=*ptrfidIN;
  if(clr)
  {
    input_bit_count=0;
```

```
    input_bit_buffer[0]=0;
    input_bit_buffer[1]=0;
    return_value=0;
  }
  else
  {
    while (input_bit_count <= (8*sizeof(int)*2-SYMb))
    {
      inputcharacter=memgetc(fidIN,INlength,0);
      if(inputcharacter!=EOF_FLAG)
      {
        fidIN++;
        lcode[0]=inputcharacter;
        lcode[1]=0;
        lsl_long(&lcode[0],8*sizeof(int)*2-SYMb-input_bit_count);
        input_bit_buffer[0] |= lcode[0];
        input_bit_buffer[1] |= lcode[1];
        input_bit_count += SYMb;
      }
      else
        break;
    }
    lcode[0]=input_bit_buffer[0];
    lcode[1]=input_bit_buffer[1];
    lsr_long(&lcode[0],8*sizeof(int)*2-BITS);
    return_value=lcode[0];
    lsl_long(&input_bit_buffer[0],BITS);
    input_bit_count -= BITS;
  }
  *ptrfidIN=fidIN;
  return(return_value);
}

/*****************************************************************************/
//Common functions accessed by compression and decompression functions
/*****************************************************************************/

// memgetc is similar to getc except that it reads an element from memory
unsigned int memgetc(const symbol *inptr, unsigned int length, bool clr)
{
  static unsigned int count=0;
  unsigned int return_value;

  if (clr)
  {
    return_value= EOF_FLAG;
    count=0;
  }
  else
  {
```

```
    if(count<length)
    {
      count++;
      return_value=*inptr;
    }
    else
      return_value= EOF_FLAG;
  }
  return(return_value);
}

//memputc is similar to putc except that it writes a 16-bit integer to memory
void memputc(const symbol ch, symbol *outptr)
{
  *outptr=ch;
}

//This function performs a left-shift of a long long int (MATLAB doesn't support normal C
    processing for long long int)
void lsl_long(unsigned int *longword, unsigned int N)
{
  if (N>0)
  {
    if (N>=8*sizeof(int)*2)
    {
      *(longword+1)=0;
      *longword=0;
    }
    else
    {
      if (N>=8*sizeof(int))
      {
        *(longword+1)= *longword << (N-8*sizeof(int));
        *longword=0;
      }
      else
      {
        *(longword+1)= (*(longword+1) << N) | (*longword >> (8*sizeof(int)-N));
        *longword <<= N;
      }
    }
  }
}

//This function performs a right-shift of a long long int (MATLAB doesn't support normal C
    processing for long long int)
void lsr_long(unsigned int *longword, unsigned int N)
{
  if (N>0)
  {
```

```
    if (N>=8*sizeof(int)*2)
    {
      *(longword+1)=0;
      *longword=0;
    }
    else
    {
      if (N>=8*sizeof(int))
      {
        *longword= *(longword+1) >> (N-8*sizeof(int));
        *(longword+1)=0;
      }
      else
      {
        *longword= (*longword >> N) | (*(longword+1) << (8*sizeof(int)-N));
        *(longword+1) >>= N;
      }
    }
  }
}
```

Index

Biography

James Giesbrecht was born in Saskatoon, Canada in 1969. He simultaneously received the BSc in electrical engineering, and BSc in computer science from the University of Saskatchewan, Canada in 1992. He received an MSc in electrical engineering from the same university in 1995. He has core skills in electronics, instrumentation, wireless technologies, digital-signal-processing, embedded real-time systems, software engineering, lecturing, industry training, and project management.

The period from 1987 to 1997 saw him heavily involved in research and lecturing at the University of Saskatchewan. During this time he was awarded the Saskatchewan Government Bursary (1987), the Saskatchewan Valley Teachers Association Scholarhip (1987), and the Mining Automation Award (1994). Between 1994 and 2000 he also worked in industry as an Electronics Engineer developing instrumentation equipment, and hardware and software for spread-spectrum modems. He emigrated to Australia in 2000, and from 2000 to early-2002 he was a Research Engineer in telecommunications, Industry Lecturer, and Operations Manager for the Centre for Telecommunications Information Networking (CTIN) at The University of Adelaide. In 2002 he acted as a telecommunications consultant for the Centre for Information and Telecommunications Research (CITR) at The University of Adelaide. At the same time he was a telecommunications consultant for mNet Corporation. Since late-2002 he has fulfilled the role of Senior Electronics Engineer (Hardware Development) and Project Manager at Ebor Computing. In 2003 he was awarded a Faculty of Engineering Computer and Mathematical Sciences Scholarship from The University of Adelaide where he began a PhD program in the area of automatic modulation recognition for HF communications, under the supervision of Professor Derek Abbott and Dr. Russell Clarke (Ebor Computing). James' work at Ebor Computing directly contributed to the PhD program. He is continuing to fulfil his role at Ebor Computing.

Scientific Genealogy

My scientific genealogy via my primary supervisor is as follows:

James Giesbrecht's Scientific Genealogy			
1774	MA	University of Cambridge	John Cranke
1782	MA	University of Cambridge	Thomas Jones
1811	MA	University of Cambridge	Adam Sedgwick
1830	MA	University of Cambridge	William Hopkins
1857	MA	University of Cambridge	Edward John Routh
1868	MA	University of Cambridge	John William Strutt (Lord Rayleigh)
1883	MA	University of Cambridge	Joseph John Thomson
1903	MA	University of Cambridge	John Sealy Townsend
1923	DPhil	University of Oxford	Victor Albert Bailey
1948	MSc	University of Sydney	Ronald Ernest Aitchison
1964	PhD	University of Sydney	Peter Harold Cole
1980	PhD	University of Adelaide	Kamran Eshraghian
1995	PhD	University of Adelaide	Derek Abbott
2008	PhD submitted	University of Adelaide	James E. Giesbrecht