LTE System Toolbox™
Getting Started Guide
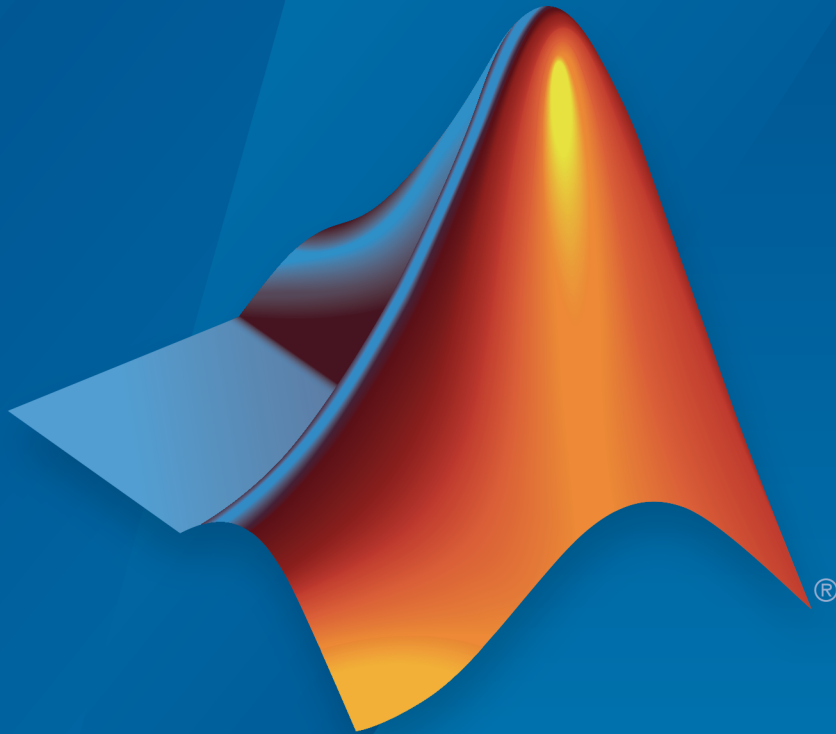
MATLAB®

MathWorks®

How to Contact MathWorks

	Latest news:	www.mathworks.com
	Sales and services:	www.mathworks.com/sales_and_services
	User community:	www.mathworks.com/matlabcentral
	Technical support:	www.mathworks.com/support/contact_us
	Phone:	508-647-7000

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

Contents

Getting Started with LTE System Toolbox Software

1

High-Level Examples

2

System Toolboxes

3

1

Getting Started with LTE System Toolbox Software

LTE System Toolbox Product Description

Simulate the physical layer of LTE and LTE-Advanced wireless communications systems

LTE System Toolbox™ provides standard-compliant functions and apps for the design, simulation, and verification of LTE and LTE-Advanced communications systems. The system toolbox accelerates LTE algorithm and physical layer (PHY) development, supports golden reference verification and conformance testing, and enables test waveform generation. With the system toolbox, you can configure, simulate, measure, and analyze end-to-end communications links. You can also create and reuse a conformance test bench to verify that designs, prototypes, and implementations comply with the LTE standard.

Key Features

- Standard-compliant models for LTE and LTE-Advanced (Releases 8, 9, 10, and 11)
- Link-level transmit and receive processing functions, support for downlink transmission modes 1 to 10, and reference designs, including coordinated multipoint (CoMP)
- Test models (E-TM) and reference measurement channel (RMC) for LTE, LTE-A, and UMTS waveform generation
- Interactive tools for conformance and BER testing
- Waveform transmission and reception with radio devices and instruments for over-the-air testing
- System and control parameter recovery from captured signals, including cell identity, MIB, and SIB1
- Channel estimation, synchronization, MIMO receiver functions, and propagation channel models

What Is LTE?

In this section...
"Long-Term Evolution" on page 1-3
"LTE Releases" on page 1-4
"LTE Physical Layer" on page 1-6

Long-Term Evolution

Long-Term Evolution (LTE) is the air interface supporting fourth generation cellular networks. LTE is specifically designed for packet data communications, where the emphasis of the technology is high spectral efficiency, high peak data rates, low latency, and frequency flexibility. The LTE specifications were developed by the Third Generation Partnership Project (3GPP).

GSM and UMTS are the predecessors of the LTE air interface and are referred to as second generation (2G) and third generation (3G) technologies, respectively. GSM was developed as a circuit switched network meaning that radio services are configured at the user's request and resources remain allocated until terminated by the network controller. This type of operation is well suited to supporting voice calls. Eventually, GSM was enhanced to support low data rate services with packet switching capability but data rates were limited by GSM's air interface, time division multiple access (TDMA). In TDMA, each user is assigned to a particular channel (frequency band) and time slot which serves to limit capacity as the channel spacing is only 200 kHz.

UMTS uses code division multiple access (CDMA) as its air interface. In CDMA, active users transmit simultaneously over the allocated bandwidth, typically 5 MHz. Signals are separated from each other by the use of orthogonal variable spreading factor (OVSF) spreading codes. The advantage of OVSF codes is that resources can be allocated asymmetrically among the active users. UMTS supports both circuits switched services for voice calls and packet switched for data sessions. Due to its larger bandwidth and superior spectral efficiency, UMTS can support higher data rates than GSM.

Unlike GSM and UMTS, LTE is a purely packet switched network in which both voice and data services are carried by IP. LTE uses orthogonal frequency division multiple access (OFDMA) in which the spectrum is divided into resource blocks (RB) that are composed of twelve 15 kHz subcarriers. By dividing the spectrum in such a way,

complicated equalizers are no longer necessary to mitigate frequency selective fading. LTE supports higher order modulation schemes up to 64-QAM along with bandwidth allocations that can be as large as 20 MHz. In addition, LTE makes use of MIMO so that very high theoretical data rates can be achieved (75 Mbps in the uplink and 300 Mbps in the downlink for release 8).

Second and third generation cellular networks consist of an interface to the public telephone or IP network, a radio network controller (RNC) that allocates radio resources among the users, a base station (referred to as a Node B in UMTS) that transmits and receives signals to and from the users, and user devices (MS for GSM and UE for UMTS). The LTE access network is similar with the exception that the RNC functionality has been pushed down into the enhanced Node B (eNB). The flatter architecture reduces the time required to establish data services resulting in lower latency. The architecture is shown below.
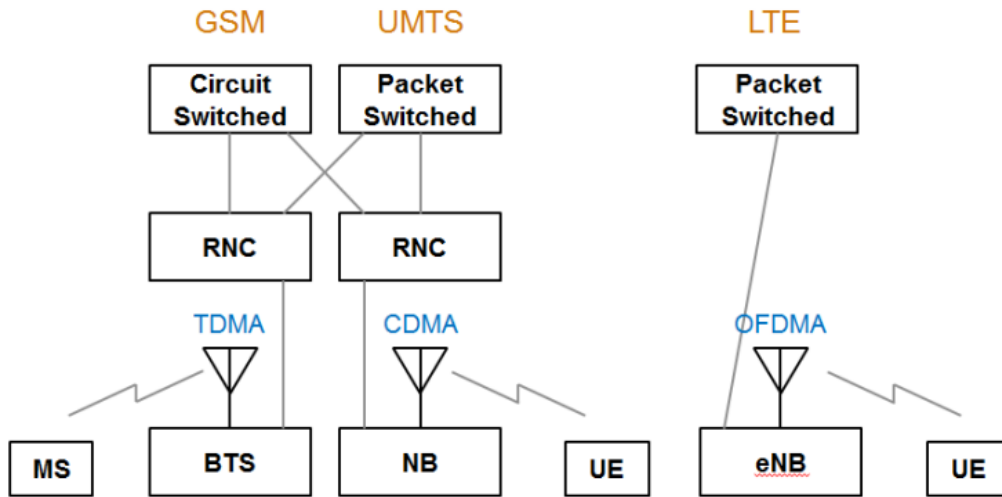
LTE Releases

Initially standardized in 3GPP release 8, the LTE standards continue to evolve over multiple releases to capture requirements that lead to improved data throughput, lower latencies, and increasingly flexible configurations.

LTE Release 8

LTE release 8 introduced LTE for the first time in 2008. It consisted of a completely new radio interface and core network which enabled substantially improved data performance compared with previous systems. Highlights from release 8 include the following features.

- Up to 300 Mbps downlink and 75 Mbps uplink
- Latency as low as 10 ms
- Bandwidth sized in 1.4, 3, 5, 10, 15, or 20 MHz blocks to allow for a variety of deployment scenarios
- Orthogonal frequency domain multiple access (OFDMA) downlink
- Single-carrier frequency domain multiple access (SC-FDMA) uplink
- Multiple-input multiple-output (MIMO) antennas
- Flat radio network architecture, with no equivalent to the GSM base station controller (BSC) or UMTS radio network controller (RNC), and functionality distributed among the base stations (enhanced NodeBs)
- All IP core network, the System Architecture Evolution (SAE)

LTE Release 9

LTE release 9 brought refinements to LTE release 8 as well as introducing some new service features and network architecture improvements. Highlights from release 9 include the following noteworthy modifications.

- Evolved multimedia broadcast and multicast service (eMBMS) for the efficient delivery of the same multimedia content to multiple destinations
- Location services (LCS) to pinpoint the location of a mobile device
- Dual layer beamforming

LTE Release 10

LTE release 10 is considered to be the start of LTE-Advanced. It significantly improved data throughput and extended cell coverage. Highlights from release 10 include the following features.

- Higher order MIMO antenna configurations supporting up to 8×8 downlinks and 4×4 uplinks

- Data throughput of up to 3 Gbps downlink and 1.5 Gbps uplink
- Carrier aggregation (CA), allowing the combination of up to five separate carriers to enable bandwidths up to 100 MHz
- Relay nodes to support Heterogeneous Networks (HetNets) containing a wide variety of cell sizes
- Enhanced inter-cell interference coordination (eICIC) to improve performance towards the edge of cells

LTE Release 11 and Beyond

LTE release 11 includes such features as enhancements to Carrier Aggregation, MIMO, relay nodes, and eICIC, introduction of new frequency bands, coordinated multipoint transmission and reception to enable simultaneous communication with multiple cells, and advanced receivers.

LTE Physical Layer

The LTE radio access network is comprised of the following protocol entities.

- Packet Data Convergence Protocol (PDCP)
- Radio Link Control (RLC)
- Medium Access Control (MAC)
- The Physical layer (PHY)

The first three protocol entities handle tasks such as header compression, ciphering, segmentation and concatenation, and multiplexing and demultiplexing. The physical layer handles coding and decoding, modulation and demodulation, and antenna mapping. The delineation between the physical layer and higher layers is indicated below.
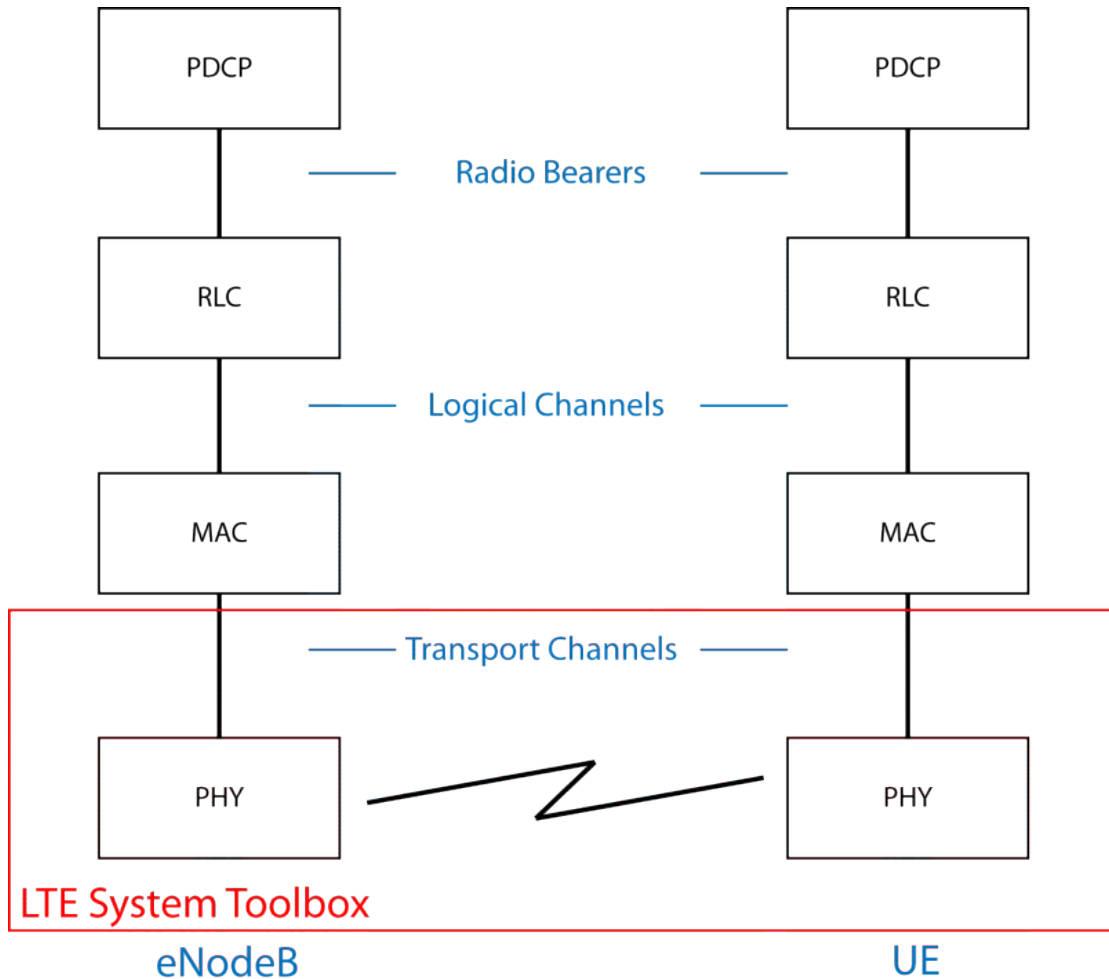
The LTE System Toolbox product focuses on the physical layer, which is highlighted in red in the preceding figure, but also supports interfacing with portions of the RLC and MAC layers, which are highlighted in blue. The primary features of the LTE physical layer are OFDM modulation, including the time-frequency structure of the resource blocks, adaptive modulation and coding, hybrid-ARQ, and MIMO.

The relationship between the downlink transport channels and physical channels is shown in the figure.
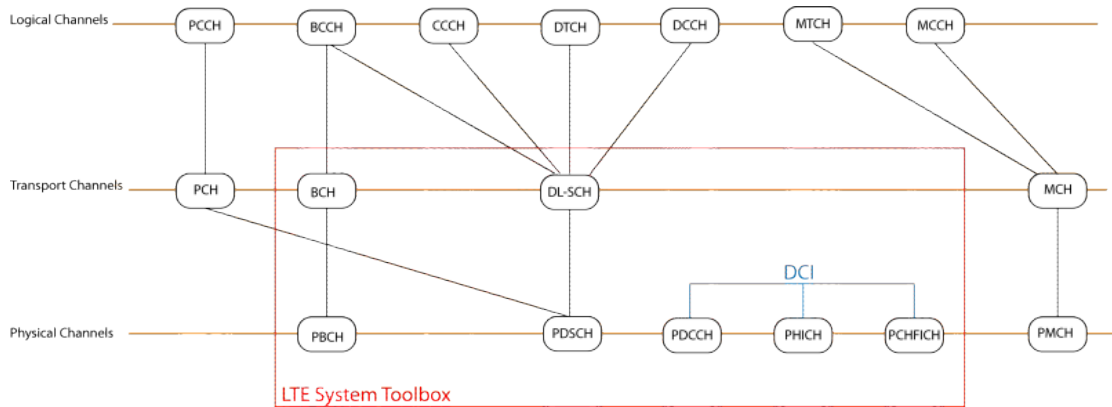
The LTE System Toolbox product offers functionality for the downlink physical channels, transport channels, and control information outlined in red in the preceding figure. For more information, see the following categories.

- "PBCH"
- "PDSCH"
- "PDCCH"
- "PCFICH"
- "PHICH"
- "Control Information"
- "Transport Channels"

The relationship between the uplink transport channels and physical channels is shown in the figure.
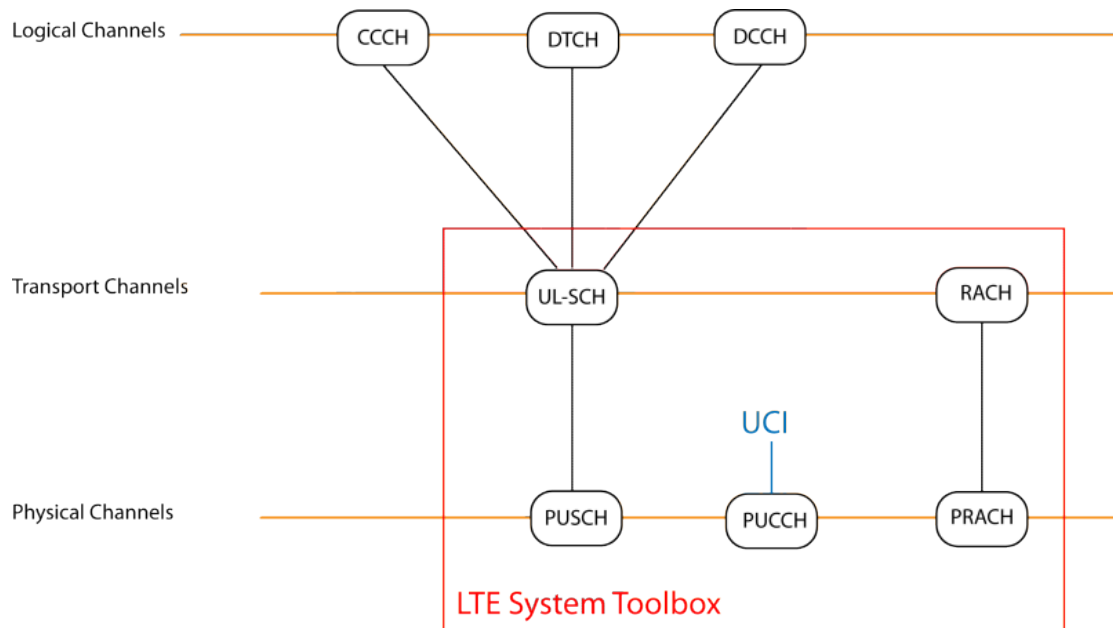
The LTE System Toolbox product offers functionality for the uplink physical channels, transport channels, and control information outlined in red in the preceding figure. For more information, see the following categories.

- "PUSCH"
- "PUCCH Format 1"
- "PUCCH Format 2"
- "PUCCH Format 3"
- "PRACH"
- "Transport Channels"
- "Control Information"

References

[1] Nohrborg, Magdalena, for 3GPP. "LTE Overview." *3GPP, A Global Initiative, THE Mobile Broadband Standard*, August 2013. http://www.3gpp.org/LTE.

[2] Dahlman, E., Parkvall, S., and Sköld, J.. *4G LTE / LTE-Advanced for Mobile Broadband*. Kidlington, Oxford: Academic Press, 2011. pp. 112–118.

Limitations

MATLAB Compiler Support

The LTE System Toolbox product does not support the MATLAB® Compiler™. You cannot compile any functions in the toolbox.

Code Generation Support

The LTE System Toolbox product does not support automatic generation of C code or HDL code. You cannot generate code from the functions in the toolbox.

Fixed-Point Support

The LTE System Toolbox product does not support fixed-point data types.

Block and System object Support

The LTE System Toolbox product does not contain Simulink® blocks or MATLAB System objects.

LTE Advanced

In this section...
"Release 9 Positioning Reference Signal" on page 1-12
"Release 9 Dual-Layer UE-Specific Beamforming" on page 1-12
"Release 10 Downlink Enhanced MIMO" on page 1-14
"Release 10 Uplink MIMO" on page 1-15
"Release 10 Spatial Orthogonal Resource Transmit Diversity (SORTD)" on page 1-16
"Release 10 PUCCH Format 3" on page 1-17
"Release 10 Carrier Aggregation" on page 1-17
"Release 11 Enhanced Physical Downlink Control Channel (EPDCCH)" on page 1-17

The LTE System Toolbox product supports LTE Advanced and represents it through functions and parameter settings. This section describes how LTE Advanced is represented, including Releases 9, 10 and 11 of the LTE Standard.

Release 9 Positioning Reference Signal

Release 9 defines a number of changes to the provision for positioning within the LTE standard. This includes the UE reception of a new downlink positioning signal (the Positioning Reference Signal or PRS) transmitted by the eNodeB and transmission of the time difference of arrival to the eNodeB as a measurement, allowing the network to compute the position of the UE.

The LTE System Toolbox product supports the PRS with the `ltePRS` and `ltePRSIndices` functions.

For a demonstration on how to use the PRS to perform time-difference of arrival (TDOA) position estimation using the LTE System Toolbox product, see "Time Difference Of Arrival Positioning Using PRS".

Release 9 Dual-Layer UE-Specific Beamforming

Release 9 provides a dual-layer UE-specific beamforming mode. Two UE-specific reference signals are defined (antenna ports 7 and 8). Two independent streams of data can be sent, one on each layer; these streams of data could be to a single UE (rank 2 transmission) or to two UEs (two rank 1 transmissions).

The LTE System Toolbox product supports the antenna port 7 and 8 reference signals with the `lteDMRS` and `lteDMRSIndices` functions. These functions support the transmission of UE-specific reference signals for Release 8, 9 and 10. The particular UE-specific reference signals created are controlled by the transmission scheme parameter, `TxScheme`. For Release 9, `TxScheme` can be set to the following transmission schemes.

Parameter Setting	Description
`'Port7-8'`	Rel-9 single-antenna port, port 7 (if `NLayers`=1). Rel-9 dual layer transmission, ports 7&8 (if `NLayers`=2)
`'Port8'`	Single-antenna port, Port 8

Note: The functions `LteUeRS` and `LteUeRSIndices`, which provided the Release 8 Port 5 reference signal, are now deprecated. In this case, use the `lteDMRS` and `lteDMRSIndices` functions instead with the `'Port5'` transmission scheme.

PDSCH transmissions associated with antenna ports 7 and 8 can be made, as for any transmission scheme, using the `ltePDSCH` and `ltePDSCHIndices` functions. These functions accept settings for the `TxScheme` parameter as described in the preceding table.

The UE-specific beamforming of the reference signals and PDSCH transmission is specified by the parameter `W` provided to `lteDMRS` and `ltePDSCH`. The functions `lteDMRSIndices` and `ltePDSCHIndices` use the parameter `NTxAnts` to specify the number of transmission antennas. See the function reference pages for details.

At the receiver, `ltePDSCHDecode` decodes PDSCH transmissions made on ports 7 and 8, under the assumption that the input will be equalized back to the transmission layers; hence, no deprecoding is required. This behavior is consistent with the operation of `lteDLChannelEstimate`, which cannot assume knowledge of the UE-specific beamforming used at the transmitter and thus produces the channel matrices between transmission layers and receive antennas. Therefore, the MMSE equalization carried out within `ltePDSCHDecode` outputs the PDSCH layers, which are then layer demapped, demodulated, and descrambled to produce soft bit estimates.

Note: A number of other functions that are aware of the transmission scheme have been updated to behave correctly for Release 9 UE-specific beamforming,

including `lteDLDeprecode`, `lteDLPrecode`, `lteDLSCH`, `lteDLSCHDecode`, `lteRateMatchTurbo`, and `lteRateRecoverTurbo`.

Release 9 transmissions on antenna port 7 and 8 are associated with DCI Format 2B, which is supported by the `lteDCI`, `lteDCIDecode`, `lteDCIInfo`, `ltePDCCHSearch`, and `lteDCIResourceAllocation` functions.

Release 10 Downlink Enhanced MIMO

Release 10 provides a further extension to downlink UE-specific beamforming with reference signals (antenna ports) for up to 8 layers. These reference signals are termed demodulation reference signals (DMRS) in the standard. To support channel estimation for up to 8 layers (noting the Cell-specific Reference Signals only support 4 antenna ports) a new channel state information reference signal (CSI-RS) set has been added, with 8 antenna ports specifically designed for CSI estimation.

The DMRS antenna ports are number 7 through 14, with ports 7 and 8 being compatible with the dual-layer UE-specific beamforming capability of Release 9. The LTE System Toolbox product supports these reference signals with the `lteDMRS` and `lteDMRSIndices` functions. These functions support the transmission of UE-specific reference signals for Release 8, 9 and 10. The particular UE-specific reference signals created are controlled by the transmission scheme parameter, `TxScheme`. For Release 10, `TxScheme` can be set to the following transmission scheme.

Parameter Setting	Description
`'Port7-14'`	Rel-10 up to 8-layer transmission, ports 7-14 (`NLayers`=1...8)

Note: The functions `LteUeRS` and `LteUeRSIndices`, which provided the Release 8 Port 5 reference signal, are now deprecated. In this case, use the `lteDMRS` and `lteDMRSIndices` functions instead with the `'Port5'` transmission scheme.

PDSCH transmissions associated with antenna ports 7 through 14 can be made, as for any transmission scheme, using the `ltePDSCH` and `ltePDSCHIndices` functions. These functions accept settings for the `TxScheme` parameter as described in the preceding table.

The UE-specific beamforming of the reference signals and PDSCH transmission is specified by the parameter W provided to `lteDMRS` and `ltePDSCH`. The functions

`lteDMRSIndices` and `ltePDSCHIndices` use the parameter `NTxAnts` to specify the number of transmission antennas. See the function reference pages for details.

At the receiver, `ltePDSCHDecode` decodes PDSCH transmissions made on ports 7 through 14, under the assumption that the input will be equalized back to the transmission layers; hence, no deprecoding is required. This behavior is consistent with the operation of `lteDLChannelEstimate`, which cannot assume knowledge of the UE-specific beamforming used at the transmitter and thus produces the channel matrices between transmission layers and receive antennas. Therefore, the MMSE equalization carried out within `ltePDSCHDecode` outputs the PDSCH layers, which are then layer demapped, demodulated, and descrambled to produce soft bit estimates.

For PMI feedback, `lteDLChannelEstimate` can optionally perform channel estimation against the CSI-RS. To do so, set the `Reference` parameter to `'CSIRS'`. Then, provide this channel estimate to `ltePMISelect` to perform PMI selection based on the codebook for CSI reporting, which is implemented using the `lteCSICodebook` function.

Note: A number of other functions that are aware of the transmission scheme have been updated to behave correctly for Release 10, including `lteDLDeprecode`, `lteDLPrecode`, `lteDLSCH`, `lteDLSCHDecode`, `lteRateMatchTurbo`, and `lteRateRecoverTurbo`.

Release 10 transmissions on antenna ports 7 through 14 are associated with DCI Format 2C, which is supported by the `lteDCI`, `lteDCIDecode`, `lteDCIInfo`, `ltePDCCHSearch`, and `lteDCIResourceAllocation` functions.

Release 10 Uplink MIMO

Release 10 supports Uplink MIMO, with 2 codewords transmitted on up to 4 layers on 4 antennas for the PUSCH. The Toolbox supports this in a very similar fashion to the downlink, with cell arrays being used to represent multiple codeword vectors, and multiple column matrices used to represent multiple layers and transmission antennas.

Uplink MIMO transmission is provided by the `ltePUSCH` and `lteULSCH` functions. In the receiver, the timing offset function `lteULFrameOffset` searches its input across all configured DRS signals. By default, `lteULChannelEstimate` provides channel estimates to the precoded DRS signals, or transmission antennas, and `ltePUSCHDecode` uses knowledge of the precoding matrices used to perform MIMO equalization. Alternatively, `lteULChannelEstimate` can be configured to provide channel estimates

to the DRS layers. To do so, set the `Reference` parameter to `'Layers'`. In this case, `ltePUSCHDecode` equalizes back to transmission layers.

The functions `ltePUSCHPrecode` and `ltePUSCHDeprecode` have been added to perform MIMO precoding and deprecoding for the PUSCH. The `lteLayerMap` and `lteLayerDemap` functions have been generalized to provide support for the uplink.

The functions `lteACKDecode`, `lteACKEncode`, `lteRIDecode`, and `lteRIEncode` have been updated to reflect the increased number of bits that can be coded in Release 10.

For a demonstration on how to create and simulate an uplink MIMO PUSCH performance test using the LTE System Toolbox product, see "Release 10 PUSCH Multiple Codeword Transmit and Receive Modeling".

Release 10 Spatial Orthogonal Resource Transmit Diversity (SORTD)

Release 10 incorporates spatial orthogonal resource transmit diversity (SORTD) transmission on the PUCCH and SRS channels. SORTD transmits independent versions of an encoded and modulated signal on each transmission antenna, by using a different orthogonal resource for each transmission antenna. For the PUCCH, the different orthogonal resources are different PUCCH resource indices, *n1_pucch*, *n2_pucch*, and *n3_pucch*. For the SRS, the different orthogonal resources are different reference signal cyclic shifts, *alpha*.

SORTD transmission is supported by the `ltePUCCH1`, `ltePUCCH1DRS`, `ltePUCCH1DRSIndices`, `ltePUCCH1Indices`, `ltePUCCH2`, `ltePUCCH2DRS`, `ltePUCCH2DRSIndices`, `ltePUCCH2Indices`, `lteSRS`, and `lteSRSIndices` functions.

In the case of the PUCCH formats 1 and 2 and their DRS signals, the SORTD is specified by the parameter `ResourceIdx`, which is a vector of indices, rather than a scalar index as in Release 8. For the SRS, the SORTD is specified by the `NTxAnts` parameter.

In the receiver, the timing offset functions, `lteULFrameOffsetPUCCH1` and `lteULFrameOffsetPUCCH2`, search their input across all configured DRS signals. The channel estimators, `lteULChannelEstimatePUCCH1` and `lteULChannelEstimatePUCCH2`, make a channel estimate against all DRS signals, or transmission antennas. If a pilot averaging frequency window size, which is a multiple of 12, is used, orthogonal despreading of different DRS signals is supported.

Release 10 PUCCH Format 3

Release 10 introduces a new PUCCH format, format 3, designed to transmit a large number of ACK indications in a single subframe. The LTE System Toolbox product implements PUCCH format 3 with the `ltePUCCH3`, `ltePUCCH3Decode`, `ltePUCCH3DRS`, `ltePUCCH3DRSIndices`, `ltePUCCH3Indices`, `ltePUCCH3PRBS`, `lteULChannelEstimatePUCCH3`, and `lteULFrameOffsetPUCCH3` functions.

Release 10 Carrier Aggregation

For a demonstration on how to create a signal covering multiple LTE carriers using carrier aggregation, see "Release 10 Downlink Carrier Aggregation Waveform Generation".

Release 11 Enhanced Physical Downlink Control Channel (EPDCCH)

Release 11 introduces the Enhanced Physical Downlink Control Channel, EPDCCH, which is designed to achieve improved spectral reuse of control channel resources. It supports CoMP, downlink MIMO as well as beamforming and frequency domain Inter-Cell Interference Coordination (ICIC). The LTE System Toolbox product implements EPDCCH with the `lteEPDCCH`, `lteEPDCCHIndices`, `lteEPDCCHDMRS`, `lteEPDCCHDMRSIndices`, and `lteEPDCCHPRBS` functions.

Data Structures

In this section...
"Overview" on page 1-18
"Multidimensional Arrays" on page 1-18
"Creating an Empty Resource Grid" on page 1-20

Overview

This section describes the data structures used to represent the resource grid in the LTE System Toolbox product.

Multidimensional Arrays

Prior to OFDM modulation (IFFT), physical channels and signals in LTE are mapped to different portions of the resource grid. The LTE System Toolbox product represents the resource grid as a multidimensional array, as shown in the following figure.
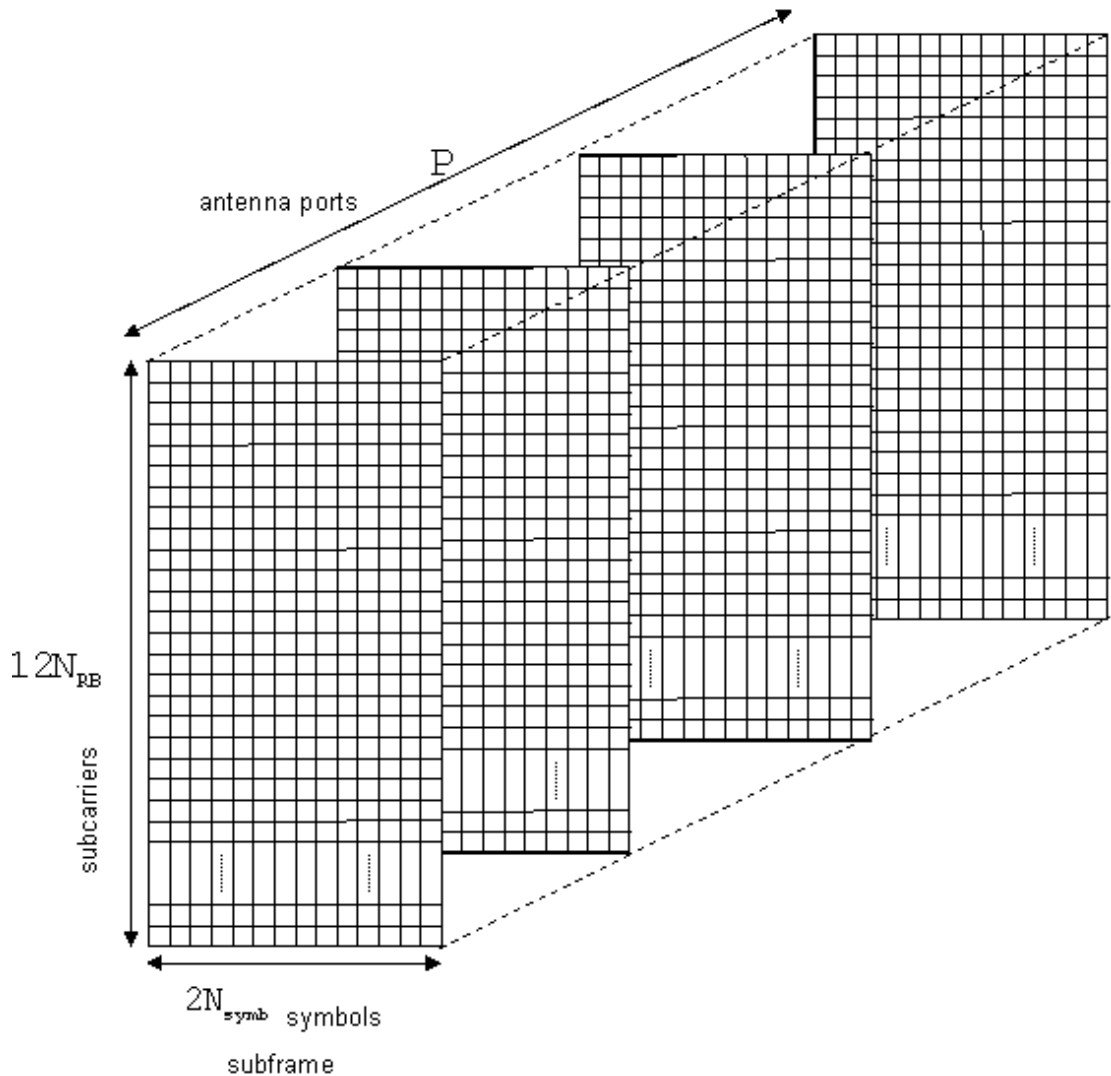
The rows of this array represent the subcarrier, while the columns map the OFDM or SC-FDMA symbols in the downlink and uplink respectively. The third dimension or plane represents the antenna ports. In the LTE System Toolbox product, the resource grid spans a subframe in the time-domain, instead of a slot. Hence, the documentation uses the term *subframe resource grid*. The size of this multidimensional array is

$12N_{RB}{\times}2N_{symb}{\times}P$, where N_{RB} is the number of resource blocks spanning the available bandwidth, N_{symb} is the number of OFDM, or SC-FDMA in the uplink, symbols per slot, and P is the number of antenna ports. Therefore, the resource grid represents a subframe, 2 slots, and whole bandwidth since there are 12 subcarriers per resource block. For the single antenna case, you can work with a two-dimensional array of size $12N_{RB}{\times}2N_{symb}$.

Creating an Empty Resource Grid

This example shows how to create an empty resource grid using two different methods. A subframe resource grid can be created using the `lteDLResourceGrid` function. Alternatively, you can use the MATLAB `zeros` function. Both approaches are valid and equivalent.

First, create the parameter structure.

```
enb.CyclicPrefix = 'Normal';
enb.NDLRB = 9;
enb.CellRefP = 1;
noSymbolsSlot = 7;
```

This structure represents the normal cyclic prefix.

Next, create an empty subframe resource grid, using both methods.

```
resourceGrid1 = zeros(enb.NDLRB*12, noSymbolsSlot*2, enb.CellRefP);
resourceGrid2 = lteDLResourceGrid(enb);
```

Compare the two grid variables for equality using the MATLAB `isequal` function.

```
isequal(resourceGrid1,resourceGrid2)

ans =

     1
```

Both approaches generate the same result. Use either approach to create an empty resource grid.

See Also
`isequal` | `lteDLResourceGrid` | `lteResourceGrid` | `lteULResourceGrid` | `zeros`

Related Examples

- "Resource Grid Indexing"

Resource Grid Indexing

Overview

The LTE System Toolbox provides facilities to generate sequences of symbols corresponding to the physical channels and signals. Indices for the mapping of these sequences to the resource grid are also generated. For convenience, the LTE System Toolbox uses the MATLAB linear indexing style to represent these indices.
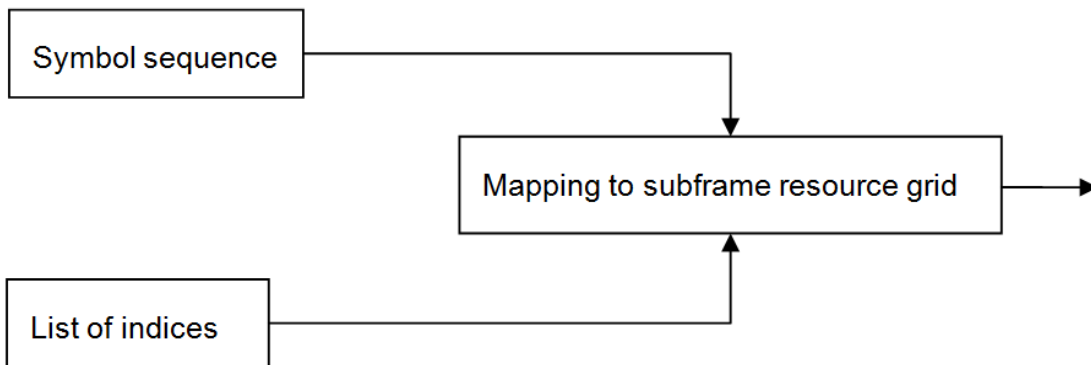
Subframe Resource Grid Size

Prior to OFDM modulation (IFFT), physical channels and signals in LTE are mapped to different portions of the subframe resource grid. The subframe resource grid is represented in the LTE System Toolbox as a multidimensional array of the following size.

$$12N_{RB} \times 2N_{symb} \times P$$

In the preceding expression, N_{RB} is the number of resource blocks spanning the available bandwidth, N_{symb} is the number of OFDM (or SC-FDMA in the uplink) symbols per slot, and P is the number of antenna ports. Therefore, the resource grid represents a subframe (2 slots) and whole bandwidth, since there are 12 subcarriers per resource block. For the single antenna case, a resource grid can be a two-dimensional matrix of the following size.

$$12N_{RB} \times 2N_{symb}$$

Creating an Empty Resource Grid

This example shows how to create an empty resource grid using two different methods. A subframe resource grid can be created using the MATLAB `zeros` function. However, the LTE System Toolbox provides the `lteDLResourceGrid` function to simplify this process. Both approaches are valid and equivalent.

First, create the parameter structure.

```
enb.CyclicPrefix = 'Normal';
enb.NDLRB = 9;
enb.CellRefP = 1;
noSymbolsSlot = 7;
```

This structure represents the normal cyclic prefix.

Next, create an empty subframe resource grid.

```
resourceGrid1 = zeros(enb.NDLRB*12, noSymbolsSlot*2, enb.CellRefP);
```

```
resourceGrid2 = lteDLResourceGrid(enb);
```

Both approaches and valid and equivalent.

Resource Grid Indexing

This example shows how to generate a reference signal and map it to an empty resource grid for the single antenna case. The LTE System Toolbox has been designed to facilitate the mapping of physical channels and signals in the resource grid.

Set up the cell-wide settings. Create a structure and specify the cell-wide settings as its fields.

```
enb.CyclicPrefix = 'Normal';
enb.NDLRB        = 6;
enb.CellRefP     = 1;
enb.NCellID      = 1;
enb.NSubframe    = 0;
antPort          = 0;
```

The enb structure now contains the parameters required by the functions to be called next.

Create an empty subframe resource grid and populate it with reference symbols. To do so, call the lteCellRSIndices and lteCellRS functions.

```
resourceGrid = lteDLResourceGrid(enb);
ind = lteCellRSIndices(enb,antPort);
rs = lteCellRS(enb,antPort);
resourceGrid (ind) = rs;
```

A call to the function lteCellRSIndices generates a list of indices identifying to where the reference signal is to be mapped, whereas the call to lteCellRS generates the reference signal symbols.

Linear Indices and Subscripts

This example shows how to generate indices in linear and subscript form. By default, indices generated by the LTE System Toolbox are in linear indexing style, which means that you can access any element of a matrix with a single index value. This indexing style

allows you to easily map the reference sequence symbols to the appropriate location in the resource grid with just one line of code.

Map the reference signal symbols to the resource grid using linear indices `ind`.

```
resourceGrid(ind) = rs;
```

As an alternative approach, you can use subscripts to access elements in a matrix. For example, for a 2-D matrix, you can access each element with a set of two elements representing the row and column equivalents.

All index generation functions in the LTE System Toolbox can produce both formats by setting the appropriate options strings as shown below. The following two calls generate linear indices.

```
ind = lteCellRSIndices(enb,antPort);
ind = lteCellRSIndices(enb,antPort,'ind')

ind =

         2
         8
        14
        20
       ...
       845
       851
       857
       863
```

Alternatively, generate indices in subscript form by providing a different option string.

```
ind = lteCellRSIndices(enb,antPort,'sub')

ind =

         2           1           1
         8           1           1
        14           1           1
        20           1           1
       ...         ...         ...
        59          12           1
        65          12           1
        71          12           1
```

In this case, the output argument, `ind`, is a matrix with three columns, corresponding to the row (subcarrier), column (OFDM symbol) and page or third dimension (antenna port) of the resource grid. These indices are calculated for antenna port 0.

Converting Between Linear Indices and Subscripts

Conversion between linear indices and subscripts can be achieved using the MATLAB `ind2sub` and `sub2ind` functions. Alternatively, all index generation functions in the LTE System Toolbox can produce both formats.

Multi-Antenna Linear Indices

This example shows how to generate indices in multi-antenna linear form. A number of toolbox functions return indices in this form. This form is a variant of the MATLAB linear indexing style in which the indices corresponding for each antenna port are in a different column. However, all indices are still in linear form.

To illustrate this, call the function `ltePDSCH` for the four antenna case.

```
enb.NCellID = 1;
enb.NSubframe = 0;
enb.CellRefP = 4;
enb.CFI = 1;
pdsch.TxScheme = 'TxDiversity';
pdsch.Modulation = 'QPSK';
pdsch.RNTI = 1;
pdsch.PRBSet = (0:5).';
data = ones(768,1);
symz = ltePDSCH(enb,pdsch,data)

sym =

  -0.5000 - 0.5000i   0.0000 + 0.0000i  -0.5000 - 0.5000i   0.0000 + 0.0000i
   0.5000 - 0.5000i   0.0000 + 0.0000i  -0.5000 + 0.5000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.5000 - 0.5000i   0.0000 + 0.0000i   0.5000 - 0.5000i
          ...                ...                ...                ...
   0.5000 - 0.5000i   0.0000 + 0.0000i  -0.5000 + 0.5000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.5000 - 0.5000i   0.0000 + 0.0000i  -0.5000 + 0.5000i
   0.0000 + 0.0000i   0.5000 + 0.5000i   0.0000 + 0.0000i  -0.5000 + 0.5000i
```

The output argument, `sym`, is a matrix with four columns, in which each column corresponds to each antenna port.

In a similar format, generate the indices for the PDSCH.

```
pdschIndices = ltePDSCHIndices(enb,pdsch,pdsch.PRBSet)

pdschIndices =
          145        1153        2161        3169
          146        1154        2162        3170
          147        1155        2163        3171
          ...         ...         ...         ...
         1006        2014        3022        4030
         1007        2015        3023        4031
         1008        2016        3024        4032
```

Again, each column corresponds to each of the four antenna ports. The concatenation of all four columns produces a column vector of indices using the MATLAB linear indexing style.

Index Base

This example shows how to generate either 0-based or 1-based indices. All mapping operations in the LTE technical specification (TS) documents refer to 0-based indexing. However, MATLAB indices must be 1-based. By default, the LTE System Toolbox generates 1-based indices, but you can generate 0-based indices by setting the appropriate options strings.

Generate 1-based indices by specifying the '1based' flag or leaving it out.

```
ind = lteCellRSIndices(enb,antPort);
ind = lteCellRSIndices(enb,antPort,'1based');
```

Generate 0-based indices by specifying the '0based' flag.

```
ind = lteCellRSIndices(enb,antPort,'0based');
```

Resource Blocks

A *resource block* is defined as a group of resource elements spanning 12 consecutive subcarriers in the frequency domain and one slot in the time domain.

In the LTE System Toolbox , the term resource block is used sometimes to represent 12 consecutive subcarriers spanning in the frequency domain and one subframe in the time domain. For example, the command ltePDSCHIndices requires parameter PRBSET.

```
pdschIndices = ltePDSCHIndices(enb,pdsch,pdsch.PRBSet)
```

The parameter `pdsch.PRBSet` is the set of physical resource block indices. It can be either a column vector or a two-column matrix. If you provide a column vector, the resource allocation is the same in both slots of the subframe, which means that each resource index block refers to both slots in a subframe. On the other hand, if you provide a two-column matrix, the resource indices refer to a slot.

See Also
lteCellRS | lteCellRSIndices | lteDLResourceGrid | ltePDSCH | ltePDSCHIndices | zeros

Related Examples
- "Parameterization"

Parameterization

Some of the functions in the LTE System Toolbox product require a large number of parameters. To simplify the process, the LTE System Toolbox product groups relevant parameters together into structures.

Parameter Structures

Consider, as an example, the task of generating PCFICH symbols and mapping indices. For this task, you can call the functions `ltePCFICH`, and `ltePCFICHIndices`. The `ltePCFICH` function also requires `cw`, an input bit vector. For this input, you can call the `lteCFI` function. All three functions require a parameter structure, `enb`, that represents the eNodeB cell-wide settings.

The function `ltePCFICH` requires `enb` to have at least the following fields.

- `NCellID` — Physical layer cell identity
- `CellRefP` — Number of cell-specific reference signal antenna ports. Valid values are 1, 2, and 4.
- `NSubframe` — Subframe number

In comparison, the function `ltePCFICHIndices` requires `enb` to have at least the following fields.

- `NCellID` — Physical layer cell identity
- `NDLRB` — Number of downlink resource blocks

Finally, the function `lteCFI` only requires `enb` to have one field, `CFI`. In all cases, if additional fields are present and not required, the function ignores them.

Create a Cell-Wide Settings Structure

This example shows how to create a cell-wide settings structure. In particular, you can create a parameter structure, `enb`, that has all the fields required by the `lteCFI`, `ltePCFICH`, and `ltePCFICHIndices` functions.

Create a new parameter structure, `enb`, with only one field, `CFI`.

```
enb.CFI      = 1;
```

Create a 32-element bit vector, `cw`, representing the rate 1/16 block encoding of the control format indicator (CFI) value. To do so, call the `lteCFI` function. Provide `enb` as an input argument.

```
cw  = lteCFI(enb);
```

Add additional fields to the parameter structure, `enb`.

```
enb.NCellID   = 0;
enb.CellRefP  = 1;
enb.NSubframe = 0;
enb.NDLRB     = 9;
```

Generate the PCFICH complex symbols using `enb`. To do so, call the `ltePCFICH` function, providing this structure, `enb`, and the bit vector, `cw`, as input arguments.

```
sym = ltePCFICH(enb,cw);
```

Although `ltePCFICH` does not require that `enb` have the `NDLRB` field, this does not cause a problem. In this case, the function ignores any non-required fields.

Generate the PCFICH mapping indices using `enb`. To do so, call the `ltePCFICHIndices` function, providing this structure as an input argument.

```
ind = ltePCFICHIndices(enb);
```

Although `ltePCFICHIndices` does not require that `enb` have the `NSubframe` field, this does not cause a problem. The function ignores any fields that it does not require.

You can remove fields from a structure using the MATLAB `rmfield` function.

Cell-Wide Parameters

Many functions in the LTE System Toolbox product require a parameter structure called `enb`. This parameter represents the eNodeB, or *cell-wide*, settings which are common

to all user equipments (UEs) in the cell. This structure can include the following fields, which are among the most common.

- `NCellID` — Physical layer cell identity
- `CellRefP` — Number of cell-specific reference signal antenna ports. Valid values are 1, 2, and 4.
- `CyclicPrefix` — Length of cyclic prefix. Valid values are `'Normal'` and `'Extended'`.
- `NSubframe` — Subframe number
- `NDLRB` — Number of downlink resource blocks

Different functions require different fields. Not all functions that require the `enb` structure need all the fields listed above. Some functions require only a subset of those listed above. In this case, any non-required fields are ignored.

When optional parameter fields are not specified, a function in the LTE System Toolbox product may assume default settings. In this case, the product produces warning messages to specify the default values that it is using. You may control these warnings using the `lteWarning` function.

Option Strings

This example shows how to pass option strings to certain functions to change some of the attributes of a function.

For example, consider the case where a list of indices for a certain physical channel is generated.

```
ind = ltePCFICHIndices(enb);
```

The input argument, `enb`, is a structure with the appropriate fields. By default, these indices are 1-based, as opposed to the 0-based indices specified in the technical specification (TS) documentation.

Change the base number used in the index generation by providing an additional *switch* string input argument.

```
ind = ltePCFICHIndices(enb, 'Obased');
ind = ltePCFICHIndices(enb, '1based');
```

If you select a switch string value of `'0based'`, the function generates 0-based indices. If you select a switch string value of `'1based'`, the function generates 1-based indices. The switch string is not required; if you do not specify the switch string, the function uses a default value.

Specify multiple switch strings for a function by providing a cell array input argument.

```
enb.Ng = 'One';
phichInd = ltePHICHIndices(enb, {'sub', '1based', 'reg'});
```

In this example, the generated PHICH indices are in subscript indexing style, 1-based, and refer to resource element groups. The cell array of strings that you specify indicates the format of the returned indices.

Alternatively, you can vary the order of the switch strings. Varying the order produces the same result.

```
phichInd = ltePHICHIndices(enb, {'1based', 'sub', 'reg'});
```

Thus, the order in which you provide the switch strings is not relevant. Both cases produce the same values in the output argument, `phichInd`.

See Also
`lteCFI` | `ltePCFICH` | `ltePCFICHIndices` | `lteWarning` | `rmfield`

Related Examples
- "UL-SCH Parameterization"

UL-SCH Parameterization

In this section...
"Set UL-SCH Parameters in Scalar Structure" on page 1-33
"Set UL-SCH Parameters in Structure Array" on page 1-34

A number of the uplink shared channel (UL-SCH) and PUSCH related functions offer two different ways of parameterizing multiple codewords in the UL-SCH or PUSCH-specific parameter structure. As with many functions in the LTE System Toolbox product, the parameters associated with codewords can be combined together in the individual fields of a single scalar (1-by-1) structure. However, many UL-SCH-specific functions also allow each codeword to be defined by separate independent elements of a (1-by-2) structure array. This feature offers additional flexibility and results in more compact code when explicit fine-grained parameterization of the individual processing steps is required.

Set UL-SCH Parameters in Scalar Structure

This example shows how to parameterize an UL-SCH or PUSCH-specific parameter structure using two different representations. Consider creating a parameter structure for the `lteULSCHDeinterleave` function.

When UCI is being transmitted on the UL-SCH, the deinterleaving and UCI demultiplexing operations require explicit knowledge of number of control channel symbols within the codeword. For example, for a single LTE Release 8 codeword, the UL-SCH specific parameters can be defined by a scalar (1-by-1) structure.

```
ulsch1.Modulation = 'QPSK';
ulsch1.QdCQI = 4;
ulsch1.QdRI  = 2;
ulsch1.QdACK = 2;
```

In this case, there are four CQI, two RI, and two HARQ-ACK symbols within the QPSK-modulated codeword.

When moving to a full LTE-Advanced uplink transmission, you must consider a second possible codeword and the impact of the additional PUSCH layering. This layering can be achieved either by adding values in the structure field values above or by using a 1-by-2 element structure array to define the codeword pair. For example, transmit a second 16-QAM-modulated codeword also, which now carries the CQI and both codewords are sent on a total of 3 spatial layers.

```
ulsch2.Modulation = {'QPSK','16QAM'};
ulsch2.NLayers = 3;
ulsch2.QdCQI = [0,4];
ulsch2.QdRI  = 2;
ulsch2.QdACK = 2;
```

Since the CQI should only be transmitted on one of the codewords (the second one here) this symbol allocation is signaled by setting ulsch2.QdCQI = [0,4].

You must explicitly specify some parameters for each codeword. However, in general, when using a single 1-by-1 structure for multi-codeword parameterization, scalar parameter field values are assigned to all codewords. The structure `ulsch2` sets the number of RI and HARQ-ACK coded modulation symbols per layer per codeword to 2. Make this number of symbols explicit for each codeword by defining the `QdRI` and `QdACK` fields as 1-by-2 vectors.

```
ulsch2.QdRI  = [2,2];
ulsch2.QdACK = [2,2];
```

One special case is the parameter field which controls the number of spatial layers, `NLayers`, which has slightly different semantics. If this field value is scalar, it defines the total number of layers across all codewords. Following the LTE standard formulae, when you set the total number of layers to 3, the LTE System Toolbox product partitions 1 layer for the first codeword and 2 layers for the second codeword. Make this layer allocation per codeword explicit by defining the `NLayers` field as a 1-by-2 vector.

```
ulsch2.NLayers = [1,2];
```

In summary, you can write the overall parameter structure by declaring all of the parameter fields at once.

```
ulsch2.Modulation = {'QPSK','16QAM'};
ulsch2.NLayers = [1,2];
ulsch2.QdCQI = [0,4];
ulsch2.QdRI  = [2,2];
ulsch2.QdACK = [2,2];
```

This structure is equivalent to the ones created earlier.

Set UL-SCH Parameters in Structure Array

This example shows how to parameterize an UL-SCH or PUSCH-specific parameter structure using two different representations. Consider creating a parameter structure for the `lteULSCHDeinterleave` function.

The UL-SCH-specific structure also allows each codeword to be defined by separate, independent elements of a 1-by-2 structure array. In this case, the important distinction is that no parameter field values are implicitly shared between the codewords. Each field value applies only to the codeword associated with that structure array element. For example, redefine the single codeword structure by creating a new 1-by-2 structure array containing 2 identical elements.

```
clear ulsch2;
ulsch2(1:2) = ulsch1

ulsch2 =

1x2 struct array with fields:

    Modulation
    QdCQI
    QdRI
    QdACK
```

Next, update only the parameters which are different for each codeword.

```
ulsch2(1).QdCQI = 0;
ulsch2(2).Modulation = '16QAM';
```

Finally, add the explicit number of layers per codeword parameter, NLayers, to the elements of the structure array.

```
[ulsch2.NLayers] = deal(1,2);
```

The first element of the final ulsch2 structure array has contents as follows.

```
ulsch2(1)

ans =

    Modulation: 'QPSK'
         QdCQI: 0
          QdRI: 2
         QdACK: 2
       NLayers: 1
```

The second element of the final ulsch2 structure array has contents as follows.

```
ulsch2(2)
```

```
ans =

    Modulation: '16QAM'
         QdCQI: 4
          QdRI: 2
         QdACK: 2
       NLayers: 2
```

Both of these forms of UL-SCH parameter representation can be used in many of the UL-SCH- and PUSCH-related functions. In addition, the `lteULSCHInfo` function can return its output structure in either form. To receive a structure array, set the second element of the 1-by-2 `opts` cell array to `'cwseparate'`. To receive a scalar structure, set it to `'cwcombined'`.

See Also
`lteULSCHDeinterleave` | `lteULSCHInfo`

Related Examples
- "Parameterization"

Obsolete LTE Toolbox Interface

Previous versions of the LTE System Toolbox product contained a different set of function names. The term *Obsolete LTE Toolbox interface* refers to these previous versions, prior to version 1.0. If you wrote scripts using any of the old function names used in previous versions, you should modify the scripts to use the new function names. Also, you should modify many of the scripts to expect column vectors for output arguments where row vectors were previously returned.

Alternatively, to enable the Obsolete LTE Toolbox interface, in the MATLAB toolstrip, select the Preferences button (⚙ Preferences). Then, in the left navigation bar, select **LTE System Toolbox**. The **LTE System Toolbox Preferences** panel appears.
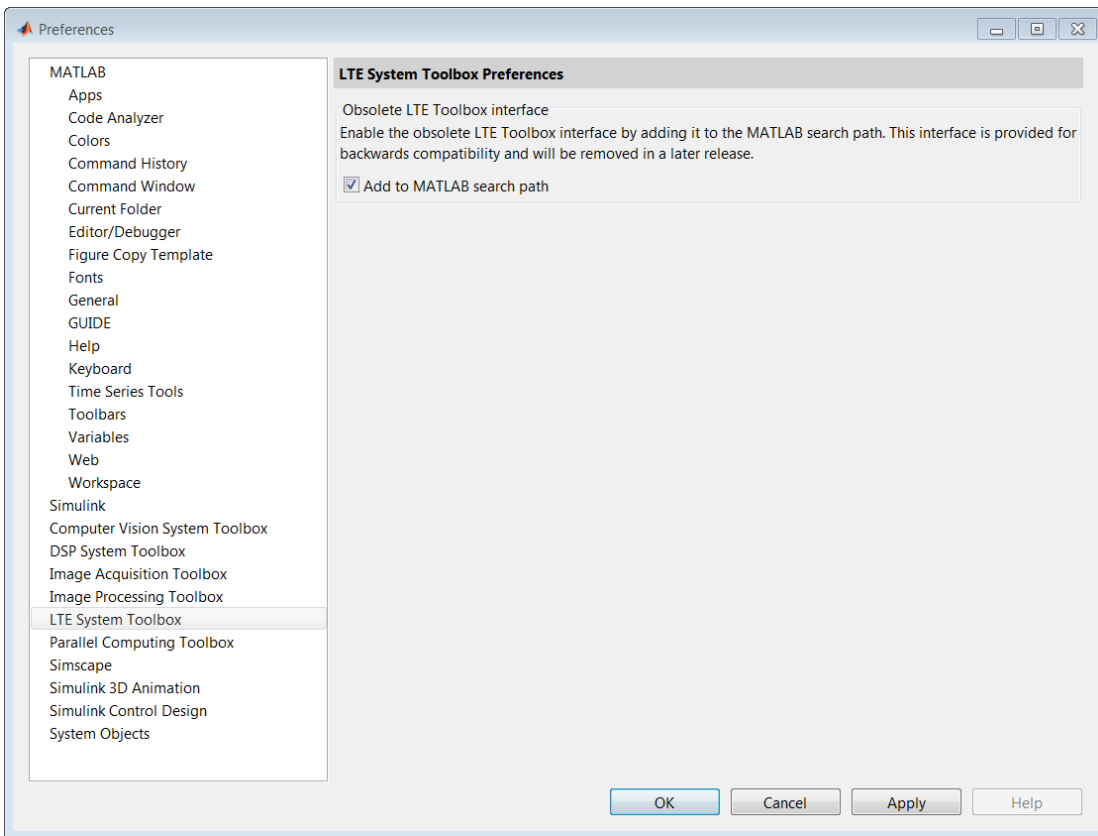
To enable the Obsolete LTE Toolbox interface, select the **Add to MATLAB search path** check box . This setting is the default and is equivalent to running the `addlteobsolete` function. To disable the Obsolete LTE Toolbox interface, clear the **Add to MATLAB search path** check box . This setting is equivalent to running the `rmlteobsolete` function. Use these functions to modify the product preferences programmatically; do not use the `setpref` function.

Note: The Obsolete LTE Toolbox interface is provided for backwards compatibility and will be removed in a later release.

When **Add to MATLAB search path** is selected, all the functions listed in the **Previous Function Name** column of the following table are available on the MATLAB path. Refer to the following table for a mapping of the previous function names to their new equivalent function names.

Previous Function Name	New Function Name
LteACKDecode	lteACKDecode
LteACKEncode	lteACKEncode
LteBCH	lteBCH
LteBCHDecode	lteBCHDecode
LteCFI	lteCFI
LteCFIDecode	lteCFIDecode
LteCQIDecode	lteCQIDecode
LteCQIEncode	lteCQIEncode
LteCRC	lteCRCEncode
LteCRCDecode	lteCRCDecode
LteCSICodebook	lteCSICodebook
LteCSIRS	lteCSIRS
LteCSIRSIndices	lteCSIRSIndices
LteCellRS	lteCellRS
LteCellRSIndices	lteCellRSIndices
LteCellSearch	lteCellSearch

Previous Function Name	New Function Name
`LteCodeBlkDeseg`	`lteCodeBlockDesegment`
`LteCodeBlkSeg`	`lteCodeBlockSegment`
`LteConvCode`	`lteConvolutionalEncode`
`LteConvDecode`	`lteConvolutionalDecode`
`LteDCI`	`lteDCI`
`LteDCIDecode`	`lteDCIDecode`
`LteDCIDims`	`lteDCIInfo`
`LteDCIEncode`	`lteDCIEncode`
`LteDLChannelEstimation`	`lteDLChannelEstimate`
`LteDLConformanceTestBench`	`lteDLConformanceTestTool`
`LteDLDeprecoder`	`lteDLDeprecode`
`LteDLFrameOffset`	`lteDLFrameOffset`
`LteDLPerfectChannelEstimation`	`lteDLPerfectChannelEstimate`
`LteDLPrecoder`	`lteDLPrecode`
`LteDLResourceGrid`	`lteDLResourceGrid`
`LteDLResourceGridDims`	`lteDLResourceGridSize`
`LteDLSCH`	`lteDLSCH`
`LteDLSCHDecode`	`lteDLSCHDecode`
`LteDLSCHDims`	`lteDLSCHInfo`
`LteDMRS`	`lteDMRS`
`LteDMRSIndices`	`lteDMRSIndices`
`LteDuplexDims`	`lteDuplexingInfo`
`LteEVM`	`lteEVM`
`LteEqualizeMIMO`	`lteEqualizeMIMO`
`LteEqualizeMMSE`	`lteEqualizeMMSE`
`LteEqualizeULMIMO`	`lteEqualizeULMIMO`
`LteEqualizeZF`	`lteEqualizeZF`

Previous Function Name	New Function Name
LteFadingChan	lteFadingChannel
LteFreqCorrect	lteFrequencyCorrect
LteFreqOffset	lteFrequencyOffset
LteHSTChan	lteHSTChannel
LteLayerDemapper	lteLayerDemap
LteLayerMapper	lteLayerMap
LteMIB	lteMIB
LteMovingChan	lteMovingChannel
LteOFDM	lteOFDMModulate
LteOFDMDemod	lteOFDMDemodulate
LteOFDMDims	lteOFDMInfo
LtePBCH	ltePBCH
LtePBCHDecode	ltePBCHDecode
LtePBCHIndices	ltePBCHIndices
LtePBCHPRBS	ltePBCHPRBS
LtePCFICH	ltePCFICH
LtePCFICHDecode	ltePCFICHDecode
LtePCFICHDims	ltePCFICHInfo
LtePCFICHIndices	ltePCFICHIndices
LtePCFICHPRBS	ltePCFICHPRBS
LtePDCCH	ltePDCCH
LtePDCCHDecode	ltePDCCHDecode
LtePDCCHDeinterleave	ltePDCCHDeinterleave
LtePDCCHDims	ltePDCCHInfo
LtePDCCHIndices	ltePDCCHIndices
LtePDCCHInterleave	ltePDCCHInterleave
LtePDCCHPRBS	ltePDCCHPRBS

Previous Function Name	New Function Name
LtePDCCHSearch	ltePDCCHSearch
LtePDCCHSpace	ltePDCCHSpace
LtePDSCH	ltePDSCH
LtePDSCHDecode	ltePDSCHDecode
LtePDSCHIndices	ltePDSCHIndices
LtePDSCHPRBS	ltePDSCHPRBS
LtePHICH	ltePHICH
LtePHICHDecode	ltePHICHDecode
LtePHICHDeprecoder	ltePHICHDeprecode
LtePHICHDims	ltePHICHInfo
LtePHICHIndices	ltePHICHIndices
LtePHICHPRBS	ltePHICHPRBS
LtePHICHPrecoder	ltePHICHPrecode
LtePHICHTxDivDecode	ltePHICHTransmitDiversityDecode
LtePMIDims	ltePMIInfo
LtePMISelection	ltePMISelect
LtePRACH	ltePRACH
LtePRACHDetect	ltePRACHDetect
LtePRACHDims	ltePRACHInfo
LtePRBFromDCI	lteDCIResourceAllocation
LtePRBS	ltePRBS
LtePRS	ltePRS
LtePRSIndices	ltePRSIndices
LtePSS	ltePSS
LtePSSIndices	ltePSSIndices
LtePUCCH1	ltePUCCH1
LtePUCCH1DRS	ltePUCCH1DRS

Previous Function Name	New Function Name
LtePUCCH1DRSIndices	ltePUCCH1DRSIndices
LtePUCCH1Decode	ltePUCCH1Decode
LtePUCCH1Indices	ltePUCCH1Indices
LtePUCCH2	ltePUCCH2
LtePUCCH2DRS	ltePUCCH2DRS
LtePUCCH2DRSDecode	ltePUCCH2DRSDecode
LtePUCCH2DRSIndices	ltePUCCH2DRSIndices
LtePUCCH2Decode	ltePUCCH2Decode
LtePUCCH2Indices	ltePUCCH2Indices
LtePUCCH2PRBS	ltePUCCH2PRBS
LtePUCCH3	ltePUCCH3
LtePUCCH3DRS	ltePUCCH3DRS
LtePUCCH3DRSIndices	ltePUCCH3DRSIndices
LtePUCCH3Decode	ltePUCCH3Decode
LtePUCCH3Indices	ltePUCCH3Indices
LtePUCCH3PRBS	ltePUCCH3PRBS
LtePUSCH	ltePUSCH
LtePUSCHDRS	ltePUSCHDRS
LtePUSCHDRSIndices	ltePUSCHDRSIndices
LtePUSCHDecode	ltePUSCHDecode
LtePUSCHDeprecoder	ltePUSCHDeprecode
LtePUSCHIndices	ltePUSCHIndices
LtePUSCHPrecoder	ltePUSCHPrecode
LteRIDecode	lteRIDecode
LteRIEncode	lteRIEncode
LteRMCDL	lteRMCDL
LteRMCDLTool	lteRMCDLTool

Previous Function Name	New Function Name
`LteRMCUL`	`lteRMCUL`
`LteRMCULTool`	`lteRMCULTool`
`LteRateMatchConv`	`lteRateMatchConvolutional`
`LteRateMatchTurbo`	`lteRateMatchTurbo`
`LteRateRecoverConv`	`lteRateRecoverConvolutional`
`LteRateRecoverTurbo`	`lteRateRecoverTurbo`
`LteResourceGrid`	`lteResourceGrid`
`LteResourceGridDims`	`lteResourceGridSize`
`LteSCFDMA`	`lteSCFDMAModulate`
`LteSCFDMADemod`	`lteSCFDMADemodulate`
`LteSCFDMADims`	`lteSCFDMAInfo`
`LteSRS`	`lteSRS`
`LteSRSDims`	`lteSRSInfo`
`LteSRSIndices`	`lteSRSIndices`
`LteSSS`	`lteSSS`
`LteSSSIndices`	`lteSSSIndices`
`LteSymbolDemod`	`lteSymbolDemodulate`
`LteSymbolMod`	`lteSymbolModulate`
`LteTBS`	`lteTBS`
`LteTestModel`	`lteTestModel`
`LteTestModelTool`	`lteTestModelTool`
`LteTurboCode`	`lteTurboEncode`
`LteTurboDecode`	`lteTurboDecode`
`LteTxDiversityDecode`	`lteTransmitDiversityDecode`
`LteUCI3Decode`	`lteUCI3Decode`
`LteUCI3Encode`	`lteUCI3Encode`
`LteUCIDecode`	`lteUCIDecode`

Previous Function Name	New Function Name
`LteUCIEncode`	`lteUCIEncode`
`LteUeRS`	Removed. Use `lteDMRS` instead.
`LteUeRSIndices`	Removed. Use `lteDMRSIndices` instead.
`LteULChannelEstimation`	`lteULChannelEstimate`
`LteULChannelEstimationPUCCH1`	`lteULChannelEstimatePUCCH1`
`LteULChannelEstimationPUCCH2`	`lteULChannelEstimatePUCCH2`
`LteULChannelEstimationPUCCH3`	`lteULChannelEstimatePUCCH3`
`LteULDeprecoder`	`lteULDeprecode`
`LteULDescrambler`	`lteULDescramble`
`LteULFrameOffset`	`lteULFrameOffset`
`LteULFrameOffsetPUCCH1`	`lteULFrameOffsetPUCCH1`
`LteULFrameOffsetPUCCH2`	`lteULFrameOffsetPUCCH2`
`LteULFrameOffsetPUCCH3`	`lteULFrameOffsetPUCCH3`
`LteULPMIDims`	`lteULPMIInfo`
`LteULPMISelection`	`lteULPMISelect`
`LteULPerfectChannelEstimation`	`lteULPerfectChannelEstimate`
`LteULPrecoder`	`lteULPrecode`
`LteULResourceGrid`	`lteULResourceGrid`
`LteULResourceGridDims`	`lteULResourceGridSize`
`LteULSCH`	`lteULSCH`
`LteULSCHDecode`	`lteULSCHDecode`
`LteULSCHDeinterleave`	`lteULSCHDeinterleave`
`LteULSCHDims`	`lteULSCHInfo`
`LteULSCHInterleave`	`lteULSCHInterleave`
`LteULScrambler`	`lteULScramble`
`LteVersion`	Removed. Use the MATLAB `version` function instead.
`LteWarning`	`lteWarning`

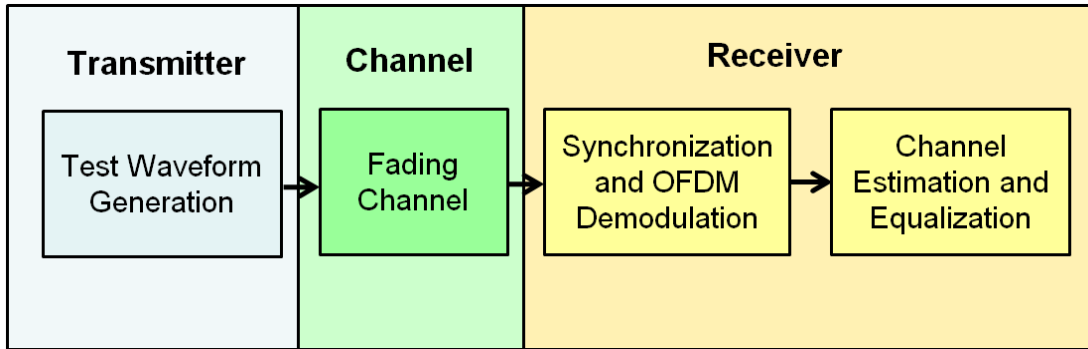| Previous Function Name	New Function Name
LteZadoffChu	Removed. Use lteZadoffChuSeq in the Communications System Toolbox™ product instead.

See Also

addlteobsolete | rmlteobsolete | setpref

High-Level Examples

Transmit-Receive Chain

This example shows how to implement the transmit and receive chain, as shown in the following figure.

Generate an E-UTRA test model (E-TM) configuration. Use this configuration to generate the waveform and populate the resource grid.

```
enb = lteTestModel('1.1','1.4MHz');
[txwave,txgrid,info] = lteTestModelTool(enb);
```

Plot a graphical representation of the transmit resource grid.

```
figure('Color','w');
helperPlotTransmitResourceGrid(enb,txgrid);
```

The resource grid is shown in the following figure.

Transmitted resource grid
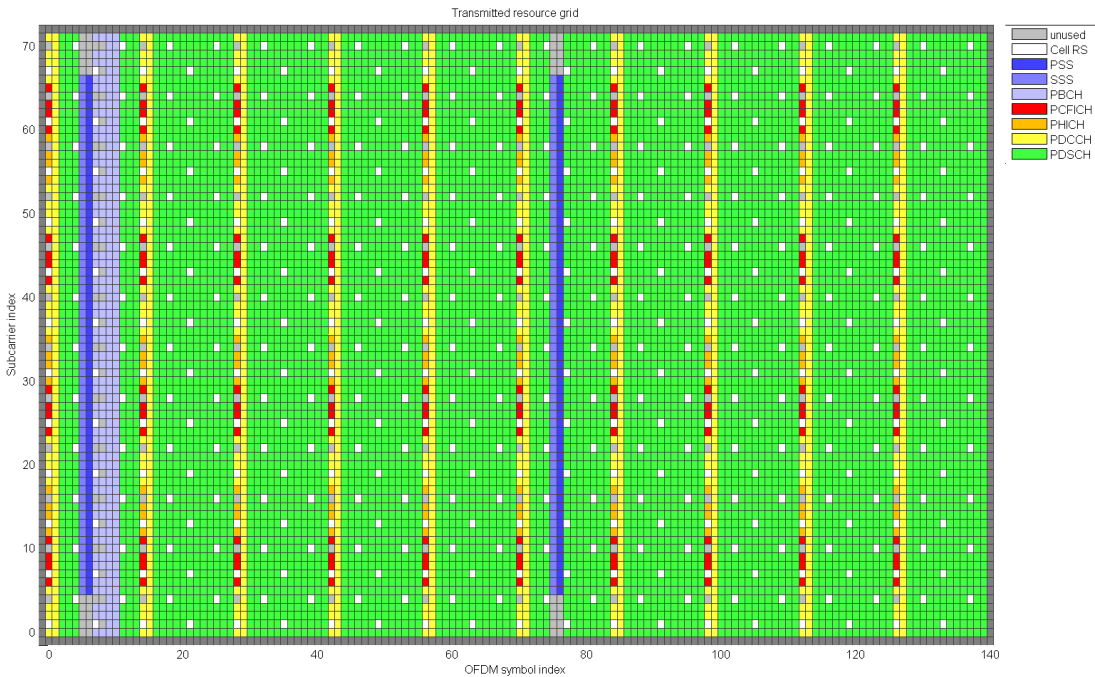
Simulate transmission through a fading channel propagation model.

```
channel.ModelType = 'GMEDS';
channel.DelayProfile = 'EVA';
channel.DopplerFreq = 70;
channel.MIMOCorrelation = 'Medium';
channel.NRxAnts = 1;
channel.InitTime = 0;
channel.InitPhase = 'Random';
channel.Seed = 17;
channel.NormalizePathGains = 'On';
channel.NormalizeTxAnts = 'On';
channel.SamplingRate = info.SamplingRate;
channel.NTerms = 16;
rxwave = lteFadingChannel(channel,[txwave;zeros(25,1)]);
```

Plot the time-varying power of the received waveform.

```
figure('Color','w');
helperPlotReceiveWaveform(info,rxwave);
```

The waveform power over time is shown in the following figure.
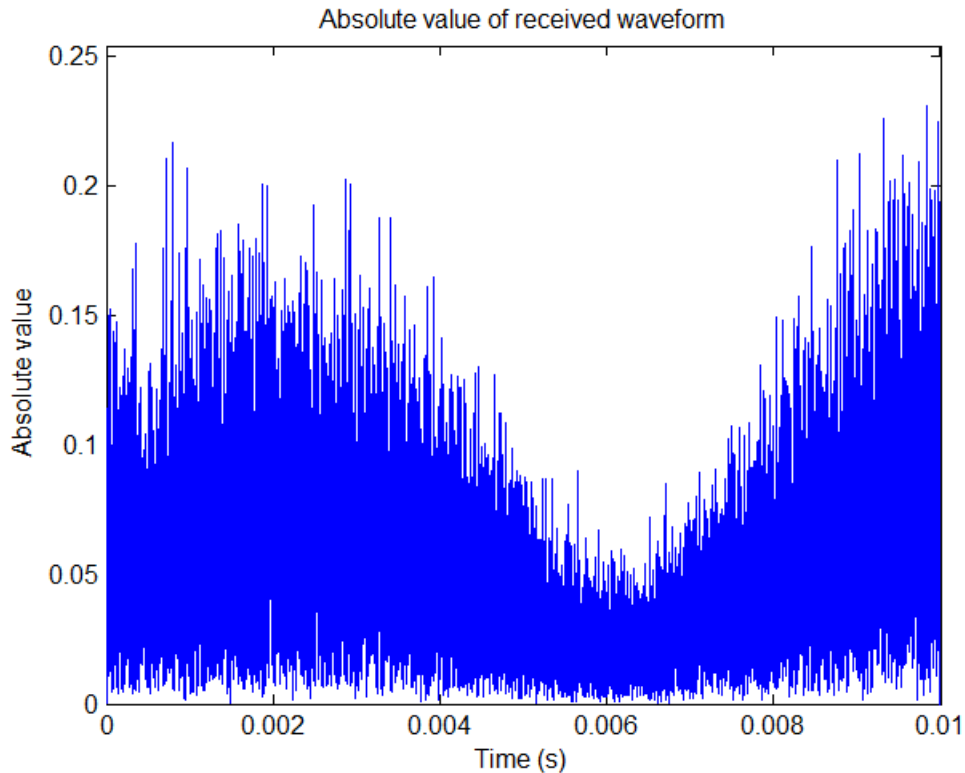
Perform frame synchronization.

```
offset = lteDLFrameOffset(enb,rxwave);
rxwave = rxwave(offset:end,:);
```
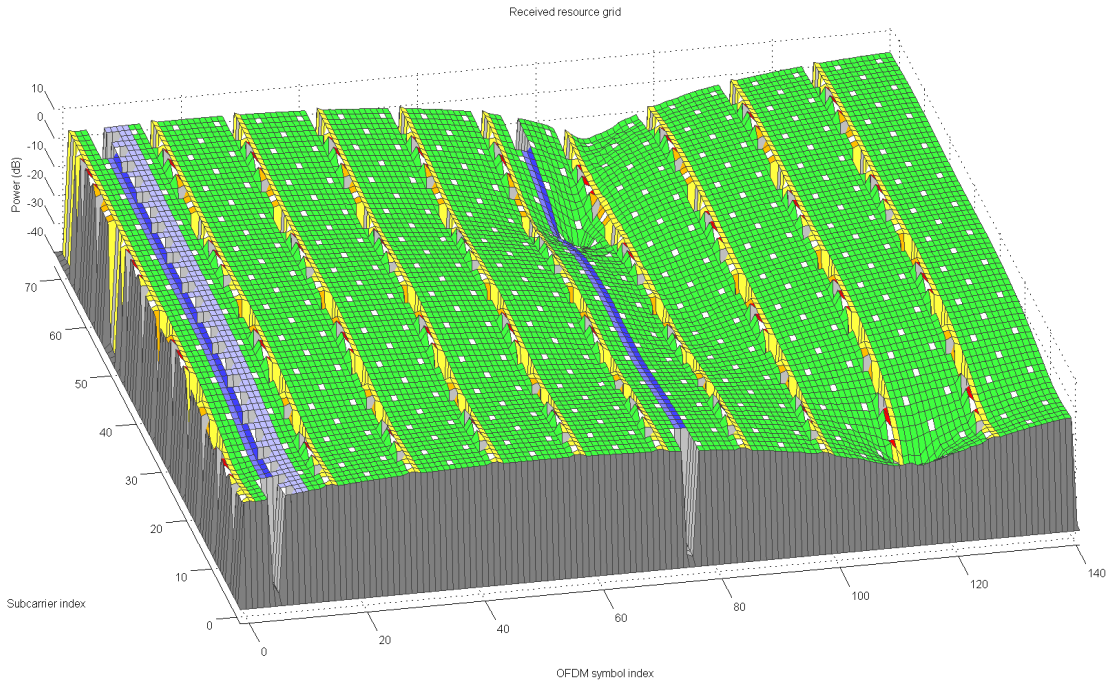
Perform OFDM demodulation.

```
rxgrid = lteOFDMDemodulate(enb,rxwave);
```

Create a surface plot of the power of the received grid for each subcarrier and OFDM symbol.

```
figure('Color','w');
helperPlotReceiveResourceGrid(enb,rxgrid);
```

The received grid power is shown in the following figure.
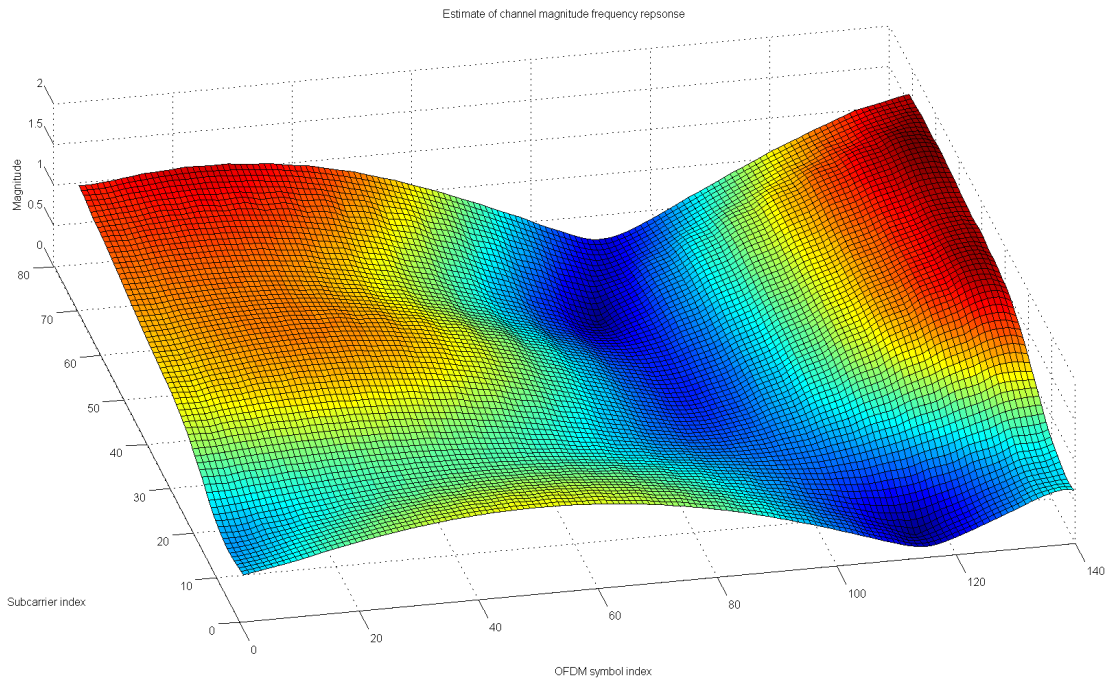

Received resource grid

Estimate the channel and noise.

```
cec.PilotAverage = 'UserDefined';
cec.FreqWindow = 9;
cec.TimeWindow = 9;
cec.InterpType = 'Cubic';
cec.InterpWindow = 'Centered';
cec.InterpWinSize = 3;
[hest,nest] = lteDLChannelEstimate(enb,cec,rxgrid);
```

Create a surface plot of the estimate the channel surface gain for each subcarrier and OFDM symbol.

```
figure('Color','w');
helperPlotChannelEstimate(hest);
```

The estimate of channel magnitude frequency response is shown in the following figure.

Estimate of channel magnitude frequency repsonse

Finally, perform minimum mean-square error (MMSE) equalization on the received grid.

```
eqgrid = lteEqualizeMMSE(rxgrid,hest,nest);
```

Create a surface plot of the power of the equalized resource grid, in dB.

```
figure('Color','w');
helperPlotEqualizedResourceGrid(enb,eqgrid);
```

The equalized resource grid power is shown in the following figure.

Equalized resource grid
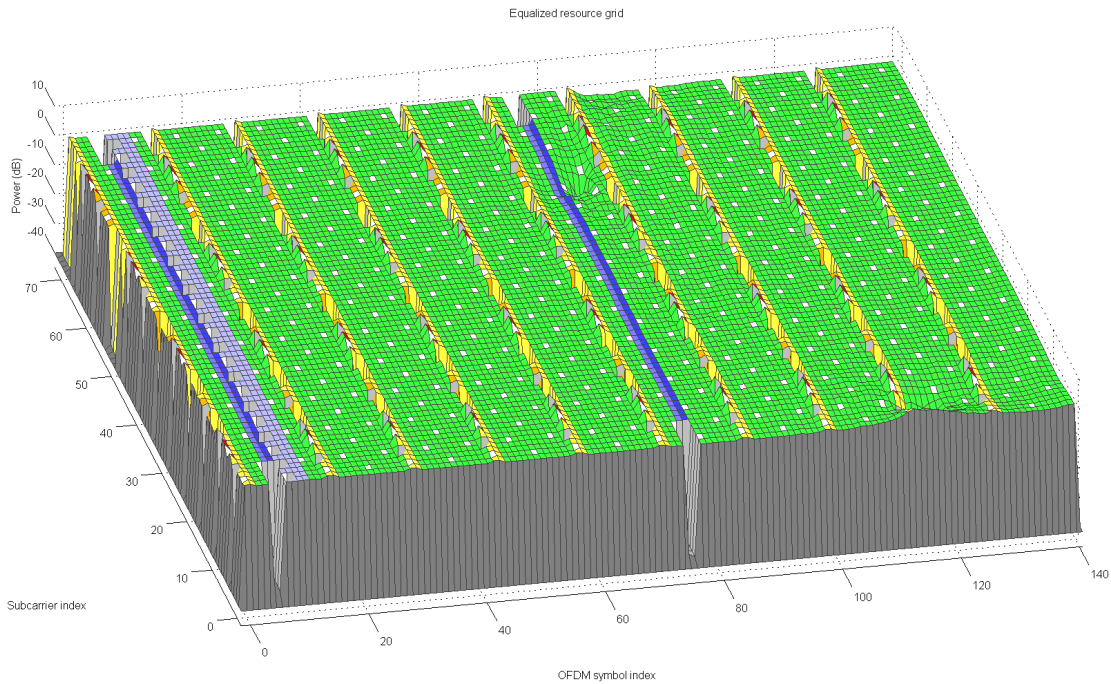
See Also
`lteDLChannelEstimate` | `lteDLFrameOffset` | `lteEqualizeMMSE` | `lteFadingChannel` | `lteOFDMDemodulate` | `lteTestModel` | `lteTestModelTool`

Related Examples
- "Generate a Test Model"
- "Simulate Propagation Channels"
- "Find Channel Impulse Response"

More About
- "Propagation Channel Models"
- "Channel Estimation"

System Toolboxes

What Is a System Toolbox?

System Toolbox products provide algorithms and tools for designing, simulating, and deploying dynamic systems in MATLAB and Simulink. These toolboxes contain MATLAB functions, System objects, and Simulink blocks that deliver the same design and verification capabilities across MATLAB and Simulink, enabling more effective collaboration among system designers. Available System Toolbox products include:

· DSP System Toolbox
· Communications System Toolbox
· Computer Vision System Toolbox
· LTE System Toolbox
· Phased Array System Toolbox

System Toolboxes support floating-point and fixed-point streaming data simulation for both sample- and frame-based data. They provide a programming environment for defining and executing code for various aspects of a system, such as initialization and reset. System Toolboxes also support code generation for a range of system development tasks and workflows, such as:

· Rapid development of reusable IP and test benches
· Sharing of component libraries and systems models across teams
· Large system simulation
· C-code generation for embedded processors
· Finite wordlength effects modeling and optimization
· Ability to prototype and test on real-time hardware