LabVIEW™

ユーザマニュアル

NATIONAL INSTRUMENTS™

1998年7月版
製品番号 321187B-01

インターネットサポート
サポート電子メール：support@nni.co.jp
電子メール：info@nni.co.jp
FTPサイト：ftp.natinst.com
日本語ホームページ：http://www.natinst.com/nni

Fax-on-Demandサポート（米国）
512 418 1111

電話サポート（日本）
Tel：03-5472-2981
Fax：03-5472-2977

海外オフィス
イスラエル 03 6120092、イタリア02 413091、英国01635 52354、オーストラリア03 9879 5166、
オーストリア 0662 45 79 90 0、オランダ 0348 433466、カナダ(オンタリオ)905 785 0085、
カナダ(ケベック) 514 694 8521、韓国02 596 7456、シンガポール2265886、スイス056 200 51 51、
スウェーデン 08 730 49 70、スペイン91 640 0085、台湾02 377 1200、デンマーク45 76 26 00、
ドイツ 089 741 31 30、ノルウェー32 84 84 00、フィンランド09 725 725 11、フランス01 48 14 24 24、
ベルギー 02 757 00 20、ブラジル011 288 336、香港2645 3186、メキシコ5 520 2635

ナショナルインスツルメンツ米国本社
11500 North Mopac Expressway Austin, Texas 78759 USA Tel: 512 794 0100

日本ナショナルインスツルメンツ(株)
〒105-0011 東京都港区芝公園2-4-1 秀和芝パークビルB館5F Tel:03-5472-2970

必ずお読みください

保証

お買い上げいただいたナショナルインスツルメンツのソフトウェア媒体は、材料およびソフトウェア製造上の欠陥によるプログラミング命令の実行不能に対して、ソフトウェアの受領書または他の文書（登録カード）によって立証された出荷日から 90 日間の保証が適用されます。上記保証期間中に、ソフトウェア媒体がプログラミング命令を実行しない欠陥がある旨の通知がナショナルインスツルメンツに対して行われた場合、ナショナルインスツルメンツは弊社の責任でそのソフトウェアを修理または交換します。ただし、ナショナルインスツルメンツはソフトウェア操作の中断や、エラー発生に関しては保証しません。

保証の対象としてお受けするには、いかなる機器についても工場から返品確認（RMA）番号を取得し、パッケージの外部にその番号を明記していただく必要があります。保証期間内の部品を所有者に返却する際の費用は、ナショナルインスツルメンツが負担します。

本書の内容については万全を期しており、技術面でのチェックも入念に行っております。技術上または印刷上の誤りがあった場合、ナショナルインスツルメンツは本書をお持ちのお客様に事前に通告することなく次回以降の版に変更を加える権利を有します。本書で誤りなどお気づきの点がありましたら弊社までご連絡ください。ナショナルインスツルメンツは、本書およびその内容により、またはそれに関連して発生した損害に対して一切責任を負いません。

ナショナルインスツルメンツは、明示、暗示を問わず、ここに記載された以外の保証は行いません。特に、商品性の保証や特定用途に対する適合性についての保証は行いません。ナショナルインスツルメンツの過失または不注意により発生した損害に対するユーザの賠償請求権は、ユーザが製品に支払われた金額を上限とします。データの消失、利益の逸失、製品の使用から生じた損失や、付随的または結果的に生じた損害に対しては、ナショナルインスツルメンツは、その損害が発生する可能性を通知されていた場合でも、一切の責任を負いません。このナショナルインスツルメンツの限定責任は、契約が遵守された場合でも、契約に違反した場合でも不注意の場合でも、訴訟方式に関係なく適用されます。ナショナルインスツルメンツに対する訴訟は、訴訟の原因の発生より 1 年以内に申し立てる必要があります。ナショナルインスツルメンツは、妥当な管理限界を越えた原因により発生した履行遅延に関しては一切の責任を負いません。ここに定めた保証では、インストール、操作、保守に関連するナショナルインスツルメンツの指示をユーザが守らなかったために生じた損害、欠陥、誤動作、動作故障は対象となりません。さらに、ユーザが製品を改造した場合や、ユーザによる乱用、誤操作、不注意の場合、および停電、電源サージ、火災、洪水、事故、第三者の行為、その他予期せぬ事象も、本保証の対象とはなりません。

著作権

商標

ナショナルインスツルメンツ社製品を医療および臨床用に使用する際の注意

ナショナルインスツルメンツの製品は、人体の治療や診断の用途に適した信頼性を保証することを目的とした部品および試験を使用して設計されておりません。ナショナルインスツルメンツの製品を医療用または臨床用に使用した場合、製品の故障またはユーザやアプリケーション設計者の過失により、身体に害を及ぼす可能性があります。ナショナルインスツルメンツの製品のアプリケーションを医療用または臨床用として使用する場合は、適切な訓練と資格を有する医療専門家が行うものとします。また当該製品の使用に際しては、重大な人身事故や死亡の危険を避けるため、従来医療用安全対策、機器、および手順を引き続き適用してください。ナショナルインスツルメンツの製品は、人体の健康や医療および臨床治療における安全性を保護、監視するための通常プロセス、手順、あるいは機器に代わるものではありません。

目次

本書について

第1章
概要

第I部
Gプログラミングの概要

第2章
VIを作成する

第5章
配列、クラスタ、およびグラフ

第6章
文字列とファイルI/O

第II部
I/Oインタフェース

第7章
LabVIEW計測器ドライバ入門

第8章
LabVIEW VISAチュートリアル

第 9 章

LabVIEW の GPIB 関数の概要

第10章
シルアルポートVI

第III部
解析

第11章
LabVIEWにおける解析の概要

第12章
信号生成

第13章
デジタル信号処理

第14章
スムージングウィンドウ

第15章
スペクトルの解析と測定

第16章
フィルタ処理

第17章
カーブフィット

第18章
線形代数

第19章
確率と統計

第IV部
ネットワークとアプリケーション間通信

第20章
通信の概要

第21章
TCPとUDP

第24章
AppleEvent

第25章
プログラム間通信

第V部
上級Gプログラミング

第26章
VIをカスタマイズする

第27章
フロントパネルオブジェクトの属性

第28章
プログラム設計

第29章
次に学習すべき内容

付録、用語集、および索引の一覧

付録 A
解析に関する参考文献

付録 B
一般的な質問

付録 C
カスタマーコミュニケーション

用語集

索引

図、表、および作業の一覧

図一覧

表一覧

作業一覧

本書について

『LabVIEW ユーザマニュアル（本書）』では、バーチャルインスツルメンツ（VI：仮想計測器）の作成について解説します。データ入出力用のインタフェース、LabVIEW の VI を使用して解析処理を行う方法、LabVIEW でのネットワーク処理やアプリケーション間通信についても説明します。本書をお使いになる前に『LabVIEW リリースノート』をお読みください。

本書の構成

『LabVIEW ユーザマニュアル』は、次のように構成されています。

- 「第 1 章　概要」では、LabVIEW のユニークなプログラミング方式を紹介し、LabVIEW を使用してプログラムの開発を始める方法について説明します。

第 I 部　G プログラミングの概要

第 I 部では、バーチャルインスツルメンツ（VI）の作成方法、VI を他の VI 内で使用する方法、ループなどのプログラミングストラクチャ、配列や文字列などのデータストラクチャについての基本的な内容を説明します。

「第 I 部　G プログラミングの概要」は、以下の章から構成されています。

- 「第 2 章　VI を作成する」では、ユーザインタフェースとなるフロントパネルや、ソースコードとなるブロックダイアグラムなどの VI を作成する方法を説明します。作成した VI は他の VI の中で使用できます。
- 「第 3 章　ループとチャート」では、While ループや For ループを使用してブロックダイアグラムの一部分を繰り返し実行する方法を示します。この章では、複数の点を一度に 1 つずつチャートにグラフィック的に表示する方法も説明します。
- 「第 4 章　Case ストラクチャ、シーケンスストラクチャ、およびフォーミュラノード」では、条件付きストラクチャである Case ストラクチャ、実行順序を規定するためのシーケンスストラクチャ、および数学的公式の実行をサポートするフォーミュラノードの使用方法を説明します。
- 「第 5 章　配列、クラスタ、およびグラフ」では、データ点のグループや配列を 1 つのグラフ上に表示する方法を示します。異なるデータタイプのグループであるクラスタを作成することで、データ点の配列だけでなくスケールパラメータもグラフに渡すことができます。
- 「第 6 章　文字列とファイル I/O」では、文字列制御器と文字列表示器、およびファイルの入出力動作の概要を説明します。

第II部 I/Oインタフェース

第II部では、データ集録、GPIB、シリアル、VXIなどのデータを入出力するためのインタフェースに関する基本的な内容を説明します。リアルタイムデータ集録に関する基本的な情報については、『データ集録ベーシックマニュアル』を参照してください。VISA（Virtual Instruments Software Architecture：仮想計測器ソフトウェアアーキテクチャ）は、GPIB、シリアル、VXI計測器とインターフェイスするための唯一のソフトウェアライブラリです。特定の計測器用に特別に開発されたLabVIEWアプリケーションは、計測器ドライバと呼ばれます。ナショナルインスツルメンツでは、VISA ライブラリを使用したいくつかの計測器ドライバを提供していますが、ユーザ独自の計測器ドライバを作成することもできます。

「第II部 I/Oインタフェース」は、以下の章から構成されています。

「第7章 LabVIEW計測器ドライバ入門」では、ナショナルインスツルメンツの計測器ドライバの作成方法と使用方法を説明します。

「第8章 LabVIEW VISAチュートリアル」では、イベント、ロック、メッセージベースの通信、レジスタベースの通信などを使用して、共通のVISAアプリケーションを実装する方法を示します。

「第9章 LabVIEWのGPIB関数の概要」では、GPIBの動作、およびIEEE 488とIEEE 488.2インターフェースの違いについて説明します。

「第10章 シルアルポートVI」では、シリアル通信に影響を及ぼす重要な要素について説明します。

第III部 解析

第III部では、データ解析、信号処理、信号発生、線形代数、曲線近似、確率、統計に関する基本的な内容を説明します。

「第III部 解析」は、以下の章から構成されています。

- 「第11章 LabVIEWにおける解析の概要」では、サポートされている機能、表記法と命名法に関する規則、サンプリング信号の方法など、すべての解析アプリケーションに適用される概念を紹介します。
- 「第12章 信号生成」「第12章 信号の発生」では、標準周波数を使用して信号を発生する方法、およびシミュレーションによる関数発生器の作成方法を説明します。
- 「第13章 デジタル信号処理」では、高速フーリエ変換（Fast Fourier Transform：FFT）と離散フーリエ変換（Discrete Fourier Transform：DFT）の違いを説明します。
- 「第14章 スムージングウィンドウ」では、ウィンドウを使用してスペクトルの漏洩を防ぎ、集録された信号の解析を改善する方法を説明します。

- 「第15章 スペクトルの解析と測定」では、振幅と位相のスペクトルの測定方法、スペクトルアナライザの開発方法、全高調波歪み（THD）の測定方法を示します。
- 「第16章 フィルタ処理」では、無限パルス応答フィルタ（IIR）、有限パルス応答フィルタ（FIR）、非線形フィルタを使用して、信号から不必要な周波数成分を除去する方法を説明します。
- 「第17章 カーブフィット」では、データセットから情報を抽出してデータ傾向に関する記述を作成する方法を示します。
- 「第18章 線形代数」では、行列演算と解析の実行方法を説明します。
- 「第19章 確率と統計」では、確率と統計に関するいくつかの基本概念を説明し、実際の問題解決におけるこれらの概念の使用方法を示します。

第IV部 ネットワークとアプリケーション間通信

第IV部では、ネットワークとアプリケーション間通信に関する基本的な内容を説明します。

「第IV部 ネットワークとアプリケーション間通信」は、以下の章から構成されています。

- 「第20章 通信の概要」では、LabVIEWにおけるネットワークとアプリケーション間通信の扱いについて紹介します。
- 「第21章 TCPとUDP」では、通信制御プロトコル（Transmission Control Protocol：TCP）、インターネットプロトコル（Internet Protocol：IP）、およびインターネットアドレスに関する基本的な概念を説明します。
- 「第22章 ActiveXのサポート」では、LabVIEWをActiveXサーバとクライアントにする方法を示します。ActiveXは、OLE自動通信と同じです。
- 「第23章 DDEを使用する」では、動的データ交換（Dynamic Data Exchange：DDE）を使用してWindowsアプリケーションとの間でデータ交換を行う方法を説明します。DDEは、クライアント内、サーバ内、ネットワーク経由で使用することができます。
- 「第24章 AppleEvent」では、AppleEventを使用してLabVIEWと他のMacintoshアプリケーションとの間でデータ交換を行う方法を示します。LabVIEWは、AppleEventsのサーバやクライアントにすることができます。
- 「第25章 プログラム間通信」では、プログラム間通信（Program-to-Program Communication：PPC）を使用して、LabVIEWと他のMacintoshアプリケーションとの間でデータ交換を行う方法を説明します。

第Ｖ部 上級Ｇプログラミング

第 V 部では、VI のカスタマイズ、フロントパネルオブジェクト、VI、LabVIEW のプログラム制御について説明し、複雑なアプリケーションの設計に関するヒントを示します。

「第Ｖ部 上級Ｇプログラミング」は、以下の章から構成されています。

- 「第 26 章 VI をカスタマイズする」では、**VI 設定 ...** や**サブ VI ノード設定 ...** を使用して、実行時の VI の外観や動作をカスタマイズする方法を示します。
- 「第 27 章 フロントパネルオブジェクトの属性」では、属性ノードと呼ばれるオブジェクトについて説明します。属性ノードとは、制御器や表示器の外観や機能特性を制御する特殊なブロックダイアグラムノードです。
- 「第 28 章 プログラム設計」では、プログラム作成時に使用するテクニックを説明し、プログラミングスタイルについてのガイドラインを示します。
- 「第 29 章 次に学習すべき内容」では、アプリケーションを正しく作成するために使用可能なリソースについて説明します。

付録、用語集、索引

- 「付録 A 解析に関する参考文献」では、LabVIEW で解析 VI を作成する際に参考となる資料の一覧を示します。これらの参考資料には、解析ライブラリに組み込まれている理論やアルゴリズムについての詳細が示されています。
- 「付録 B 一般的な質問」には、LabVIEW のネットワーク通信、計測器 I/O、特に GPIB とシリアル I/O について、よくお問い合せをいただく質問についての答えが載っています。
- 「付録 C カスタマーコミュニケーション」には、お客様からの技術的問題点を当社が解決するための情報をお客様が収集するのに役立つ用紙、および製品の資料に対するお客様のご意見を送っていただくための用紙が用意されています。
- 「用語集」は、本書で使用されている用語を数字、アルファベット、五十音順に並べ、それぞれの用語の解説を示します。略語、頭字語、メートル法の接頭辞、ニーモニック、記号も含まれています。
- 「索引」は、本書に含まれている重要な用語や内容を数字、アルファベットおよび五十音順に並べ、各項目の掲載ページとともに示します。

本書で使用する表記規則

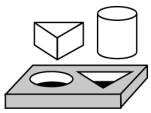
本書では、以下の表記規則を使用します。

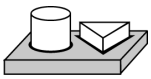
<>　キーボード上のキーの名前は<Shift>のように山括弧で囲みます。DBIO<3...0>のように数字の間に省略部分があり全体が山括弧で囲まれている場合は、1つのビットまたは信号名に対応する値の範囲を示します。

-　<Control-Alt-Delete>のように複数のキーの名前をハイフンでつなぎ、全体を山括弧で囲んだものは、示された名前のキーを同時に押す必要があることを示します。

→　→の記号は、ネストされたメニュー項目やダイアログボックスをたどって最終的な操作に至ることを示します。**ファイル→ページ設定→オプション→代替フォント**という順番で示されている場合は、まず**ファイル**メニューをプルダウンし、次に**ページ設定**、**オプション**という項目をそれぞれ選択し、最後にダイアログボックスから**代替フォント**というオプションを選択します。

『』および「」　参照すべきマニュアルもしくはマニュアルの箇所を表します。

 　**太字**のテキストの左にこのアイコンがある場合は、ここから LabVIEW についてさらに詳しく学習するため、示される指示の順番に従って作業を開始することを示します。

 　**太字**のテキストの左にこのアイコンがある場合は、示される指示の順番に従って LabVIEW についてさらに詳しく学習するための作業がここで終了することを示します。

☞　**太字**のテキストの左にこのアイコンがある場合は、重要な情報ですので注意してください。

⚠　**太字**のテキストの左にこのアイコンがある場合は、人体への損傷、データの損失、システムのクラッシュなどを防止するために必要な注意事項を示します。

`monospace`　この書体は、ディスクドライブ、パス、ディレクトリ、プログラム、サブプログラム、サブルーチン、デバイス、関数、操作、変数などに対応する名前、ファイル名や拡張子、およびプログラムから抽出されたステートメントや注釈などにも使用されます。

`[monospace]`　[　]内のテキストは、説明の通りにコードの一部、プログラム例、シンタックス例など、キーボードから入力する必要があるテキストや文字を示します。

[monospace]［太字］　[　]内の太字のテキストは、画面に自動的に表示されるメッセージや応答を示します。

ゴシック体	このフォントで示すテキストは、メニュー、メニュー項目、パラメータ、ダイアログボックス、ダイアログボックスのボタンやオプション、アイコン、ウィンドウ、Windows 95 のタブ、LED などの名前を示します。また、このフォントのテキストの左側余白にアイコンが表記されている場合には、アイコンに対応してそれぞれ作業の目的、備考、注意、警告を示します。複数のプラットフォームに関する情報が記載されている箇所では、プラットフォーム名がこの書体で示されます。
斜体	この書体は、変数、および Windows 3.*x* のようにユーザがこの中から適切な語や値を入力することを示します。
パス	本書のパスの表示では、ドライブ名、ディレクトリ、フォルダ、ファイルなどの区切りに円コード（¥）を使用します。
太字明朝体	このフォントで示すテキストは、強調部分や重要な概念を示します。

関連資料

『G プログラミングリファレンスマニュアル』

『LabVIEW データ集録ベーシックマニュアル』

『LabVIEW Function and VI Reference Manual』

『LabVIEW クイックスタートガイド』

『LabVIEW オンラインリファレンス』［**ヘルプ→オンラインリファレンス**］を選択します。

『LabVIEW オンラインチュートリアル』（Windows のみ）［LabVIEW］ダイアログボックスから起動します。

『G プログラミングクイックリファレンスカード』

『LabVIEW 始めにお読みください (カード)』

『LabVIEW リリースノート』

『LabVIEW アップグレードノート』

カスタマーコミュニケーション

ナショナルインスツルメンツでは、当社の製品やマニュアルに関するお客様からのご意見をお待ちしております。当社の製品を使用して開発されたアプリケーションについての情報もお寄せください。お客様のアプリケーションに問題がある場合は、弊社がお手伝いさせていただきます。

お客様からのお問い合わせを円滑にするため、本書の「付録C カスタマーコミュニケーション」には、お客様のご意見やシステム構成を記入していただくための用紙が用意されています。

1

概要

この章では、LabVIEW を使用したユニークなプログラミング方法を紹介します。また、LabVIEW を使用してプログラムの開発を始める方法についても説明します。さらに、詳しい説明が参照できる他の章やマニュアルについて説明します。

LabVIEW とは？

LabVIEWは、最新のC、BASIC、ナショナルインスツルメンツのLabWindows/CVIなどの開発環境とよく似たプログラム開発環境ですが、LabVIEW とこれらのアプリケーションとの間には、重要な相違点があります。他のプログラミングシステムではテキストベースの言語を使用してコード行を作成するのに対し、LabVIEW ではグラフィック的なプログラミング言語**G** を使用してブロックダイアグラム形式のプログラムを生成します。

LabVIEW は、C や BASIC と同様、汎用のプログラミングシステムであり、どのようなプログラミングタスクにも対応できるよう、総合的な関数ライブラリが用意されています。LabVIEW には、データ集録、GPIB とシリアル計測器の制御、データ解析、データ表示、データ記憶などのためのライブラリがあります。このほか、従来のプログラム開発ツールも含まれていますので、ブレークポイントを設定したり、実行状態を動画化してプログラム内のデータフローをチェックしたり、シングルステップでプログラムを動作させることができ、プログラムの開発やデバッグが一層容易に行えます。

LabVIEW の動作

LabVIEW は汎用のプログラミングシステムですが、データ集録や計測器制御用に設計された専用の関数ライブラリや開発ツールも備えています。LabVIEW プログラムは、外観と動作状態が実際の計測器に似ているため、**バーチャルインスツルメンツ（仮想計測器）** VI と呼ばれますが、VI は従来の言語プログラムの関数に似ています。

VI は、対話式ユーザインタフェース、ソースコードの役割を持つデータフローダイアグラム、上位 VI から VI を呼び出せるようにするためのアイコン接続から構成されています。VI の構成をさらに詳しく説明すると次のようになります。

- VI の対話式ユーザインタフェースは実際の計測器のパネルを模倣しているため、フロントパネルと呼ばれます。**フロントパネル**には、ノブ、押しボタン、グラフ、その他の制御器や表示器があります。マウスやキーボードでデータを入力すると、コンピュータ画面に結果が表示されます。
- VIは、ユーザがGで作成した**ブロックダイアグラム**から指示を受け取ります。ブロックダイアグラムは、プログラミングすべきテーマに対する解答を図式的に表現したものです。また、ブロックダイアグラムはVIに対するソースコードともなります。
- VIは、階層的なモジュール式構造になっています。VIは、最上位プログラムとしても、他のプログラムのサブプログラムとしても使用できます。他のVIに属するVIは**サブVI**と呼ばれます。VIの**アイコン**と**コネクタ**はグラフィカルパラメータリストと同様に機能しますので、他のVIからサブVIにデータを渡すことができます。

このような特徴を持つLabVIEWは、**モジュール式プログラミング**の概念を採用し、この概念に従って作られています。アプリケーションを一連のタスクに分割し、さらにこれらを分割していくと、複雑なアプリケーションも一連の単純なタスクに分解できます。分解されたそれぞれの部分タスクを実行するVIを作成し、これらのVIを別のブロックダイアグラムに組み込むことで、大規模なタスクを実現できます。最終的には、アプリケーションの機能を表現したサブVIの集合体が最上位VIの中に含まれた形になります。

各サブVIは、アプリケーションの他の部分とは別に単体でも実行できますので、デバッグは非常に簡単です。さらに、下位のサブVIで実行するタスクは複数のアプリケーションに共通な場合が多いため、ある種のサブVI群を開発しておくと、ユーザが作成する複数のアプリケーションに対応できます。

Gプログラミング

Gは、使いやすいグラフィカルデータフロープログラミング言語であり、LabVIEWの基本になっています。Gを使用すると、科学計算、工程の監視と制御、試験や測定などのアプリケーションを単純化できます。そしてGはこれ以外にもさまざまなアプリケーションに使用できます。

「第I部　Gプログラミングの概要」では、Gの機能のうち、ほとんどのLabVIEWアプリケーションを始める際に理解しておくことが必要なものについて説明します。LabVIEWの機能についての詳細は、『Gプログラミングリファレンスマニュアル』を参照してください。

このマニュアルで説明しているGの基本概念の一覧を以下に示します。

- VI — バーチャルインスツルメンツ（VI）には、フロントパネル、ブロックダイアグラム、アイコン／コネクタという3つの主要な構成要素があります。フロントパネルは、VIのユーザインターフェイスを規定します。ブロックダイアグラムは、ユーザがノード、端子、ワイヤを使用して作成した実行可能なコードから構成されます。アイコン／コネクタを使用すると、あるVIを、別のVIのブロックダイアグラム内でサブVIとして使用できます。VIについての詳細は、「第2章　VIを作成する」、「第26章　VIをカスタマイズする」を参照してください。
- ループとチャート — Gには、サブダイアグラムを繰り返し実行するための2つのストラクチャ、すなわち**Whileループ**と**Forループ**があります。どちらのストラクチャも、サイズが自由に変えられる四角形となっています。繰り返し実行したいサブダイアグラムは、ループストラクチャの枠内に入れます。Whileループは、条件端子の値がTRUEである限り実行を続けます。Forループは、設定された回数だけ実行を繰り返します。チャートは、トレンド情報をリアルタイムでオペレータに表示するのに使用されます。ループとチャートについての詳細は、「第3章　ループとチャート」を参照してください。
- Caseストラクチャとシーケンスストラクチャ — **Caseストラクチャ**は条件付き分岐を制御するストラクチャで、特定の入力に基づいてサブダイアグラムを実行します。**シーケンスストラクチャ**はプログラムを制御するストラクチャで、数値的な順番に従ってサブダイアグラムを実行します。Caseストラクチャとシーケンスストラクチャについての詳細は、「第4章　Caseストラクチャ、シーケンスストラクチャ、およびフォーミュラノード」を参照してください。
- 属性ノード — **属性ノード**は、ブロックダイアグラムの特殊なノードであり、これを使用すると制御器や表示器の外観や機能的特性を制御できます。属性ノードについての詳細は、「第27章　フロントパネルオブジェクトの属性」を参照してください。
- 配列、クラスタ、グラフ — **配列**は、同じタイプのデータ要素の集合体であり、サイズを変更することができます。**クラスタ**は、同じタイプまたは異なるタイプのデータ要素の集合体であり、サイズは固定されています。グラフは通常、データの表示に使用されます。配列、クラスタ、グラフについての詳細は、「第5章　配列、クラスタ、およびグラフ」を参照してください。

LabVIEWシステムの構成（Windowsの場合）

ソフトウェアに添付されている『LabVIEW リリースノート』に説明されていますが、インストールが完了した時点で、LabVIEW ディレクトリには以下のファイルが含まれています。

- LabVIEW.EXE — これがLabVIEWのプログラムです。LabVIEWを起動するには、このプログラムを起動します。
- vi.lib ディレクトリ — GPIB、解析、データ集録（DAQ）用など、LabVIEW に付属する VI のライブラリが入っています。これらの大部分は、**関数**パレットから使用できます。
- examples ディレクトリ — サンプルの入った多数のサブディレクトリがあります。またこのディレクトリには、サンプルに関するガイドとして機能するreadme.viというVIも入っています。
- serpdrvとdaqdrv — これらのファイルはそれぞれ、シリアルポートとDAQ通信に対するLabVIEWのインタフェースの一部として機能します。これらのファイルは、vi.libと同じディレクトリ内になければなりません。
- resourceディレクトリ
  - labview.rsc、lvstring.rsc、lvicon.rsc — LabVIEWアプリケーションで使用されるデータファイル
  - （Windows3.1）lvdevice.dll — このファイルはLabVIEWに対するタイミングサービスを提供しますので、vi.libと同じディレクトリ内にないとLabVIEWが動作しません。
  - （Windows3.1）lvimage.dll — このファイルを使用すると、さまざまなグラフィカルプログラムを使用して作成された画像をLabVIEWがロードすることができます。
  - labview50.tlb — このファイルは、LabVIEWがActiveXサーバとして機能できるようにするためのタイプライブラリです。
  - ole_container.dll — このファイルを使用することで、LabVIEWはActiveXコンテナを表示、更新することができます。
  - lvwuti132.dll — このファイルはソリューションウィザードで使用され、ユーザの基準に従ってDAQおよび計測器I/Oのサンプルを作成します。
  - lvjpeg.dllおよびlvpng.dll — これらのファイルは、VI文書をHTMLファイルに印刷するときにHTMLファイル内のJPEGとPNGのグラフィックス表示をサポートします。
- Cintoolsディレクトリ — CのコードをLabVIEWのVIにリンクするためのコードインタフェースノード（Code Interface Nodes：CIN）の作成に必要なファイルが入っています。

- visarcファイル — VISA（Virtual Instrument Software Architecture：VIソフトウェアアーキテクチャ）に対するLabVIEWのインタフェースの一部として機能します。VISAは、VXI、GPIB、シリアル計測器を制御する唯一のインタフェースライブラリを提供します。
- labview.ini — LabVIEW用の構成オプションが含まれています。
- Projectディレクトリ — LabVIEWの**プロジェクト**メニューの項目になるファイルが含まれています。
- menusディレクトリ — **制御器**パレットと**関数**パレットのストラクチャの構成に使用されるファイルが含まれています。
- Instr.libディレクトリ — VXI、GPIB、シリアル計測器の制御に使用される計測器ドライバが含まれています。ナショナルインスツルメンツの計測器ドライバをインストールする場合、計測器ドライバは**関数**パレットに追加されますので、このディレクトリにインストールしてください。
- Helpディレクトリ — 完全なオンラインマニュアルと、ユーザのアプリケーションに共通なサンプルを探すのに役立つサンプル検索ヘルプファイルが含まれています。
- Tutorialディレクトリ — LabVIEW環境の基本概念を説明した対話式のオンラインチュートリアルを実行するのに必要なファイルが含まれています。
- Activityディレクトリ — 本書の各作業の完了時には、作成したVIをこのディレクトリに保存できます。
- User.libディレクトリ — ユーザが作成したVIで一般的に使用するVIはこのディレクトリに保存することができます。このディレクトリ内のVIは**関数**パレットに表示されます。
- Wizardディレクトリ — このディレクトリは**ファイル**メニューの**ソリューションウィザード**オプションを作成します。**ファイル**メニューに項目を追加するには、このディレクトリを使用します。

LabVIEWは、GPIB、データ集録、VXIドライバハードウェア用のドライバソフトウェアをインストールします。構成に関しては、『LabVIEWデータ集録ベーシックマニュアル』の「第2章 データ集録ハードウェアをインストールして構成する」、『VXI VI Reference Manual』、および本書の「第8章 LabVIEW VISAチュートリアル」を参照してください。

Windowsの場合の起動画面

LabVIEWを起動すると、案内用のダイアログボックスが表示されます。このダイアログボックスからは、内容について紹介する資料、よく使用されるコマンド、クイックヒントなどに容易にアクセスできます。この案内用ダイアログボックスを省略したい場合は、ダイアログボックス下部の

チェックボックスを使用してこの機能を無効にできます。この機能を有効に戻すには、環境設定ダイアログボックスを使用します。

すべてのVIを閉じたときにも、同様のダイアログボックスが表示されます。**小さいダイアログボックス**ボタンを使用すると、**新規VI**、**VIを開く**、**終了**ボタンだけが含まれた簡易バージョンのダイアログボックスに切り替わります。

LabVIEWシステムの構成（Macintoshの場合）

ソフトウェアに添付されている『LabVIEWリリースノート』に説明されていますが、インストールが完了した時点で、LabVIEWディレクトリには以下のファイルが含まれているはずです。

- `LabVIEW` ― これがLabVIEWのプログラムです。LabVIEWを起動するには、このプログラムを起動します。
- `vi.lib`フォルダ ― GPIB、解析、データ集録（DAQ）用など、LabVIEWに付属するVIのライブラリが入っています。これらの大部分は、**関数**パレットから使用できます。
- `examples`フォルダ ― サンプルの入った多数のフォルダがあります。またこのフォルダには、サンプルに関するガイドとして機能するreadme.viというVIも入っています。
- `resource`フォルダ
  - `lvstring.rsrc`と`lvicon.rsrc` ― LabVIEWアプリケーションで使用されるデータファイル
  - `lvjpeg.lib`と`lvpng.lib` ― これらのファイルは、VI文書をHTMLファイルに印刷するときにHTMLファイル内のJPEGとPNGのグラフィックス表示をサポートします。
- `cintools`フォルダ ― CのコードをLabVIEWのVIにリンクするためのコードインタフェースノード（Code Interface Nodes：CIN）の作成に必要なファイルが入っています。
- `visarc`ファイル ― VISA（Virtual Instrument Software Architecture：VIソフトウェアアーキテクチャ）に対するLabVIEWのインタフェースの一部として機能します。VISAは、VXI、GPIB、シリアル計測器を制御する唯一のインタフェースライブラリを提供します。
- `Project`フォルダ ― LabVIEWの**プロジェクト**メニューの項目になるファイルが含まれています。
- `menus`フォルダ ― **制御器**パレットと**関数**パレットのストラクチャの構成に使用されるファイルが含まれています。

- instr.lib フォルダ ― VXI、GPIB、シリアル計測器の制御に使用される計測器ドライバが含まれています。ナショナルインスツルメンツの計測器ドライバをインストールする場合、計測器ドライバは**関数**パレットに追加されますので、このディレクトリにインストールしてください。
- help フォルダ ― 完全なオンラインマニュアルと、ユーザのアプリケーションに共通なサンプルを探すのに役立つサンプル検索ヘルプファイルが含まれています。
- activity フォルダ ― 本書の各作業の完了時には、作成したVIをこのディレクトリに保存できます。
- user.lib フォルダ ― ユーザが作成したVIの中で一般的に使用するVIはこのディレクトリに保存することができます。このディレクトリ内のVIは**関数**パレットに表示されます。
- wizard フォルダ ― このディレクトリは**ファイル**メニューの**ソリューションウィザード**オプションを作成します（PCI Macintoshのみ）。**ファイル**メニューに項目を追加するには、このディレクトリを使用します。

また、LabVIEW インストールユーティリティは、GPIB や DAQ プラグインボードを使用できるようにいくつかのドライバファイルもインストールします。

- System Folder:Control Panels:NI-488 INIT ― このコントロールパネルには、GPIB ボード用のドライバが含まれています。これを使用してボードを構成でき、設定の変更はほとんど必要ありません。
- System Folder:Control Panels:NI-DAQ ― このコントロールパネルは、DAQ ドライバをメモリにロードします。これを使用すると、DAQ ボードと SCXI モジュールの位置と動作を構成できます。
- System Folder:Extensions:NI-DMA/DSP ― GPIBとDAQドライバはいずれもこの拡張機能を使用します。この機能は、データのダイレクトメモリアクセス（DMA）をサポートし、高速なデータ転送を実現します。この拡張機能はNI-DSP ボードもサポートします。

LabVIEW は、GPIB とデータ集録ハードウェア用のドライバソフトウェアをインストールします。構成については、『LabVIEW データ集録ベーシックマニュアル』の「第 2 章　データ集録ハードウェアのインストールと構成」を参照してください。

LabVIEWシステムの構成（UNIXの場合）

ソフトウェアに添付されている『LabVIEW リリースノート』に説明されていますが、インストールが完了した時点で、LabVIEW ディレクトリには以下のファイルが含まれています。

- `labview` — これがLabVIEWのプログラムです。LabVIEWを開始するには、このプログラムを起動します。
- `vi.lib`ディレクトリ — GPIB、解析、データ集録（DAQ）用など、LabVIEWに付属するVIのライブラリが入っています。これらの大部分は、**関数**パレットから使用できます。
- `examples`ディレクトリ — サンプルの入った多数のサブディレクトリがあります。またこのディレクトリには、サンプルに関するガイドとして機能する`readme.vi`というVIも入っています。
- `serpdrv` — このファイルは、シリアルポート通信に対するLabVIEWのインタフェースの一部として機能します。このファイルは、`vi.lib`と同じディレクトリ内になければなりません。
- `resource`ディレクトリ
  - `labview.rsc`、`lvstring.rsc`、`lvicon.rsc` — LabVIEWアプリケーションで使用されるデータファイル
  - `lvjpeg.lib`と`lvpng.lib` — これらのファイルは、VI 文書をHTMLファイルに印刷するときにHTMLファイル内のJPEGとPNGのグラフィックス表示をサポートします。
- `cintools`ディレクトリ — CのコードをLabVIEWのVIにリンクするためのコードインタフェースノード（Code Interface Nodes：CIN）の作成に必要なファイルが入っています。
- `visarc`ファイル — VISA（Virtual Instrument Software Architecture：VIソフトウェアアーキテクチャ）に対するLabVIEWのインタフェースの一部として機能します。VISAは、VXI、GPIB、シリアル計測器を制御する唯一のインタフェースライブラリを提供します。
- `Project`ディレクトリ — LabVIEWの**プロジェクト**メニューの項目になるファイルが含まれています。
- `menus`ディレクトリ — **制御器**パレットと**関数**パレットのストラクチャの構成に使用されるファイルが含まれています。
- `instr.lib`ディレクトリ — VXI、GPIB、シリアル計測器の制御に使用される計測器ドライバが含まれています。ナショナルインスツルメンツの計測器ドライバをインストールする場合、計測器ドライバは**関数**パレットに追加されますので、このディレクトリにインストールしてください。

- helpディレクトリ ― 完全なオンラインマニュアルと、ユーザのアプリケーションに共通なサンプルを探すのに役立つサンプル検索ヘルプファイルが含まれています。
- activityディレクトリ ― 本書の各作業の完了時には、作成したVIをこのディレクトリに保存できます。
- user.libディレクトリ ― ユーザが作成したVIで一般的に使用するVIはこのディレクトリに保存することができます。このディレクトリ内のVIは**関数**パレットに表示されます。
- Wizardディレクトリ ― このディレクトリは**ファイル**メニューの**ソリューションウィザード**オプションを作成します。**ファイル**メニューに項目を追加するには、このディレクトリを使用します。
- acrobatディレクトリ ― acrobat（.pdf）形式のオンラインマニュアルが含まれています。
- acroreadディレクトリ ― Adobe Acrobatリーダファイルが含まれています。

ツールキットのサポート

vi.lib¥addonsにインストールされたファイルは、**制御器**パレットと**関数**パレットの最上位に自動的に表示されます。新しいツールキットでこの機能を使用すると、インストール後のアクセスがさらに容易になります。ツールキットでどこか他の場所にファイルをインストールしてある場合でも、addonsディレクトリにこれらを移動するとアクセスが容易になります。自分のVIをパレットに追加する場合は、user.libに入れるか、カスタムパレットセットに追加されることをお奨めします。

次に読む資料

本書はLabVIEWにおけるアプリケーションの構築に関する基本的事項を説明したものです。LabVIEW環境に慣れるため、『LabVIEWオンラインチュートリアル』**（Windowsの場合のみ）**、『LabVIEWクイックスタートガイド』、および本書の「第I部　Gプログラミングの概要」を参照してください。

ほとんどのLabVIEWアプリケーションは、センサや計測器に対するI/Oインタフェース、フロントパネルでのデータ表示、データ解析、データ記憶、ネットワーク経由でのデータ転送などのタスクに分割されます。これらの各タスクについての詳細は、本書の「第II部　I/Oインタフェース」、「第III部　解析」、「第IV部　ネットワークとアプリケーション間通信」を参照してください。またGプログラミングのさらに高度なテクニックについては、「第V部　上級Gプログラミング」を参照してください。

自分のアプリケーションに似たサンプルを生成したり検索する方法については、ソリューションウィザード**（Windows および PCI Macintosh のみ）**、またはサンプル検索**（Windows のみ）**のオンラインヘルプファイルを参照してください。これらは、LabVIEW の起動ダイアログからアクセスできます。

個々の関数や VI に関する詳細は、オンラインヘルプを参照してください。

第I部

Gプログラミングの概要

第I部では、バーチャルインスツルメンツ（VI）の作成方法、VIを別のVI内で使用する方法、ループなどのプログラミングストラクチャ、配列や文字列などのデータストラクチャについての基本的な事項を説明します。

「第I部 Gプログラミングの概要」は、以下の章から構成されています。

- 「第2章 VIを作成する」では、ユーザインタフェースであるフロントパネルや、ソースコードであるブロックダイアグラムなどのVIを作成する方法を説明します。作成したVIは、別のVIの中で使用できます。
- 「第3章 ループとチャート」では、Whileループ や Forループを使用してブロックダイアグラムの一部分を繰り返し実行する方法を示します。この章では、複数の点を1つのチャートに一度に1つずつグラフィック的に表示する方法も説明します。
- 「第4章 Caseストラクチャ、シーケンスストラクチャ、およびフォーミュラノード」では、条件付きストラクチャであるCaseストラクチャ、実行順序を規定するためのシーケンスストラクチャ、および数学的公式の実行をサポートするフォーミュラノードの使用方法を説明します。
- 「第5章 配列、クラスタ、およびグラフ」では、データ点のグループや配列を1つのグラフ上に表示する方法を示します。異なるデータタイプのグループであるクラスタを作成することで、データ点の配列だけでなくスケールパラメータもグラフに渡すことができます。
- 「第6章 文字列とファイルI/O」では、文字列の操作方法、およびASCIIファイルにこれらの文字列を書き込む方法を説明します。

☞ **注** **（Windows 3.1）第I部で作成したVIは、VIライブラリに保存する必要があります。VIライブラリでは、8文字を超えるファイル名を使用することができます。また、第I部で行う作業に必要なVIは、`LabVIEW¥Activity¥Activity.lib`のVIライブラリにあります。VIライブラリに関する詳細は、『Gプログラミングリファレンスマニュアル』の「第2章 VIを編集する」の中の「VIを保存する」の項を参照してください。**

2

VIを作成する

この章では、バーチャルインスツルメンツ（VI：仮想計測器）についての基本概念を紹介し、次の内容を理解するための作業を行います。

- アイコンとコネクタの作成方法
- VIをサブVIとして使用する方法

バーチャルインスツルメンツとは？

バーチャルインスツルメンツ(VI)は、グラフィカルプログラミング言語Gにおけるプログラムです。バーチャルインスツルメンツのフロントパネルのユーザインタフェースは、多くの場合実際の計測器に似ています。また、GにはVIに似た組み込み関数も用意されていますが、これらにはVIのようなフロントパネルやブロックダイアグラムがありません。関数アイコンの背景は常に黄色になります。

VIの作成方法

LabVIEWのアプリケーションを作成する際の重要なポイントの一つは、VIの階層的な性質をよく理解して利用することです。作成したVIは、上位のVIのブロックダイアグラムの中でサブVIとして使用することができます。

VI階層

アプリケーションを作成する場合、まず始めに最上位のVIアプリケーションに対する入出力を定義していきます。そして、ブロックダイアグラムでの流れに従って、データに対して必要な処理を行うサブVIを作成します。ブロックダイアグラムに多数のアイコンが含まれる場合は、これらをまとめて下位のVIにすることにより、簡潔なブロックダイアグラムになります。このようなモジュール方式の採用により、アプリケーションがわかりやすく、デバッグ、保守が容易になります。

他のアプリケーションと同様、自分のVIを通常のディレクトリのファイルに保存することができます。またGでは、複数のVIをVIライブラリという1つのファイルに保存してください。

Windows 3.1 を使用している場合は、複合的な場合について（最大 255 文字までの）長いファイル名を使用することができますので、複数のVIは複数のVIライブラリに保存してください。

Windows 3.1 にVIを転送する必要がある場合以外は、VIライブラリを使用しないでください。VI を個別ファイルに保存した方が VI ライブラリを使用するよりも効果的です。これは、VIライブラリを使用するよりも、ファイルのコピー、名前変更、削除などの操作を容易に行えるからです。VIライブラリを使用する場合と個々のファイルを使用する場合のメリットとデメリットについては、『Gプログラミングリファレンスマニュアル』の「第2章 VIを編集する」の「VIを保存する」の項の一覧表を参照してください。

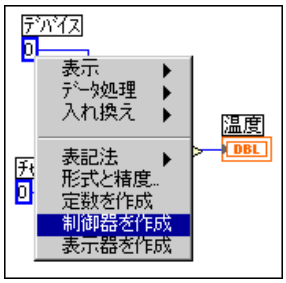
VIライブラリはロード、保存、開くなどの操作を行える点は他のディレクトリと同じですが、VI ライブラリは階層的ではありません。したがって、別のVIライブラリ内にVIライブラリを作成することや、VIライブラリ内に新しいディレクトリを作成することができません。このため、LabVIEW環境外でVIライブラリ内のVIをリストすることはできません。

作成されたVIライブラリは、LabVIEWファイルダイアログボックスのフォルダアイコンの上に、VIを含むフォルダとして表示されます。通常のディレクトリは、VIのラベルのないフォルダとして表示されます。

VIライブラリに自分のVIを保存することはないと思いますが、VIライブラリの動作については理解しておいてください。本書で進める作業では、`LabVIEW¥Activity`ディレクトリにVIを保存します。これらの作業の解答は、`LabVIEW¥Activity¥solution`ディレクトリに入っています。

制御器、定数、表示器

制御器はフロントパネル上に配置され、データを対話形式の操作でVIに入力したりプログラムによりサブVIに入力するためのオブジェクトです。表示器は、フロントパネル上に配置され、出力を表示するためのオブジェクトです。G の制御器と表示器は、それぞれ、従来のプログラム言語における入力パラメータと出力パラメータに似ています。制御器や表示器をフロントパネルに配置してブロックダイアグラム上で関数や VI に配線する方法のほかに、ブロックダイアグラム上から制御器や表示器を直接作成する方法もあります。このためには、ブロックダイアグラム上の関数やVIの入力端子を**ポップアップ**して**制御器を作成**を選択します。これにより、正しいデータタイプの制御器が作成され、端子に配線されます。



出力端子をポップアップして**表示器を作成**を選択すると、表示器が作成されその出力端子に配線されます。ブロックダイアグラム上に定数を配置して関数やVIに配線する方法のほかに、関数やVI端子をポップアップして**定数を作成**を選択する方法もあります。ブロックダイアグラムから制御器や表示器を削除することはできません。すべてのフロントパネルオブジェクトの場合と同様、フロントパネルに移動して位置決めツールを選択してからオブジェクトを削除する必要があります。

フロントパネル上で新しい制御器や表示器を作成するたびに、LabVIEWはブロックダイアグラム内に対応する端子を作成します。端子の記号は、制御器や表示器のデータタイプを示します。たとえば、DBL端子は倍精度の浮動小数点数、TF端子はブール数、I16端子は通常の16ビット整数、ABC端子は文字列であることを示します。Gにおけるデータタイプとグラフィック上の表記法についての詳細は、『Gプログラミングクィックリファレンスカード』を参照してください。

端子

端子は、VIや関数上でデータの受け渡しが行われる領域で、テキストベースのプログラミング言語におけるパラメータに相当します。関数やVIの端子に正しく配線することが重要です。アイコンコネクタを表示すると、正しい配線を容易に行うことができます。このためには、関数またはVIをポップアップして**表示→端子**を選択します。アイコンに戻すには、もう一度関数またはVIをポップアップして**表示→端子**を選択します。

ワイヤ

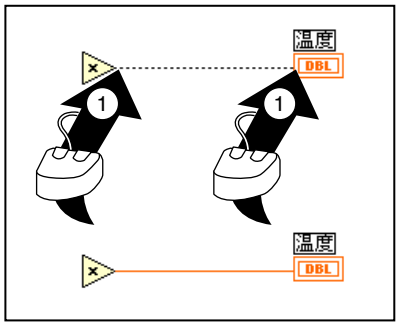
ワイヤはノード間のデータパスです。ワイヤが受け渡すデータの種類に従って、各ワイヤは色分けされています。青いワイヤは整数を、オレンジのワイヤは浮動小数点数を、緑のワイヤはブール数を、ピンクのワイヤは文字列を受け渡します。ワイヤのスタイルと色に関する詳細は、『Gプログラミングクィックリファレンスカード』を参照してください。

ホットスポット 

ある端子から別の端子に配線するには、配線ツールで第1の端子をクリックし、第2の端子にツールを移動して第2の端子をクリックします。どちらの端子から操作を始めても構いません。リールから引き出された配線部分の先端が、配線ツールのホットスポットとなります。

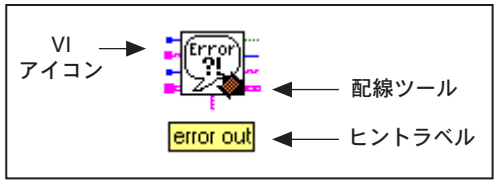
この項に出てくる配線図で、左図のマウス記号の先端の矢印はクリックすべき場所を、矢印に付けられた数字はマウスボタンを押す回数を示します。

配線ツールが端子上にある場合は端子の部分が点滅し、クリックするとその端子にワイヤが接続されることを示します。ある端子から別の端子に配線ツールを動かすときは、マウスボタンを押したままにしないでください。マウスを現在の方向に対して直角に動かすとワイヤを一度曲げることができます。ワイヤを何度も曲げるにはマウスボタンをクリックします。ワイヤの方向を変えるにはスペースバーを押します。ワイヤを固定してからマウスを直角に動かすには、マウスボタンをクリックします。



ヒントラベル

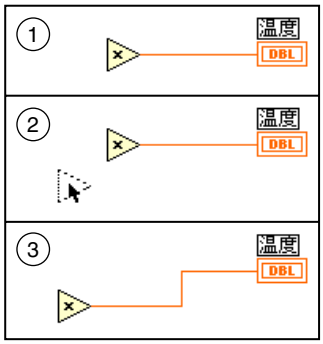
ノード端子上に配線ツールを移動すると、その端子に対応するヒントラベルが現れます。**ヒントラベル**は、各端子の名前が表示された小さい黄色のテキストラベルです。これらのヒントラベルは端子を配線するのに役立ちます。Simple Error Handler VIの出力の上に配線ツールを移動したときに表示されるヒントラベルを、次の図に示します。



☞ **注**　配線ツールをノード上に移動すると、G はそれぞれの入力と出力を示すワイヤスタブを表示します。ノードへの入力となるワイヤスタブの先端にはドットが付きます。

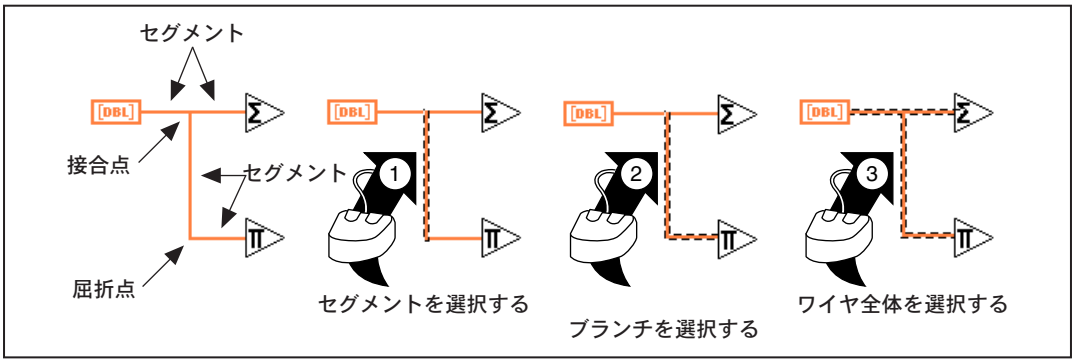
ワイヤの延長

選択されたオブジェクトを、位置決めツールを使用して新しい位置までドラッグすると、配線されたオブジェクトを個々に、または一括して移動することができます。



ワイヤの選択と削除

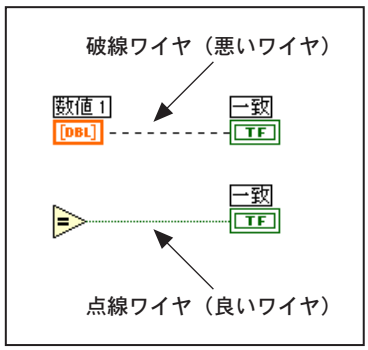
ノードを間違って配線してしまうことがあります。このような場合は、削除したいワイヤを選択して<Delete>を押します。横または縦方向の1本のワイヤをワイヤセグメントといいます。3本または4本のワイヤセグメントがつながる点を**ワイヤ接合点**といいます。ある接合点から別の接合点へのワイヤセグメント、ある端子から次の接合点へのワイヤセグメント、間に接合点がない場合のある端子から別の端子へのワイヤセグメントはすべて、1つのワイヤブランチに含まれます。位置決めツールでワイヤセグメントをクリックするとワイヤセグメントが選択されます。ダブルクリックするとブランチが選択され、3回クリックするとワイヤ全体が選択されます。

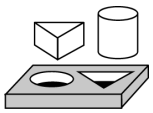


不良ワイヤ

破線のワイヤは不良ワイヤであることを示します。不良ワイヤとなる原因はいくつかあり、たとえば2つの制御器を接続した場合や、データタイプが一致しないのにソース端子を接続先端子に接続した場合（数値をブール値に接続するなど）に不良ワイヤとなります。不良ワイヤを取り除くには、位置決めツールで不良ワイヤをクリックして<Delete>を押します。**編集→不良ワイヤの削除**を選択するか<Ctrl-B>を押すと、ブロックダイアグラム内のすべての不良ワイヤが削除されます。VIが動作しない場合や、［**ワイヤ：未接続配線があります**］というエラーメッセージが表示される場合は、この操作が問題を素早く解決するのに最も有効です。

☞ **注**　**黒い破線のワイヤと点線のワイヤを混同しないでください。次の図のように、点線のワイヤはブール値データタイプを示します。**





作業 2-1.　VIを作成する

ここでは、VIを作成するのが目的です。

温度と体積の測定値を電圧として読み込むセンサがあるとします。LabVIEW¥Activityディレクトリに入っているVIを使用して、ボルト単位の温度と体積の測定をシミュレーションします。これらの測定値を華氏温度とリットルに変換するVIを作成します。

1. **ファイル→新規**を選択して新しいフロントパネルを開きます。すべてのVIが閉じられている場合は、［LabVIEW］ダイアログボックスから**新規VI**を選択します。

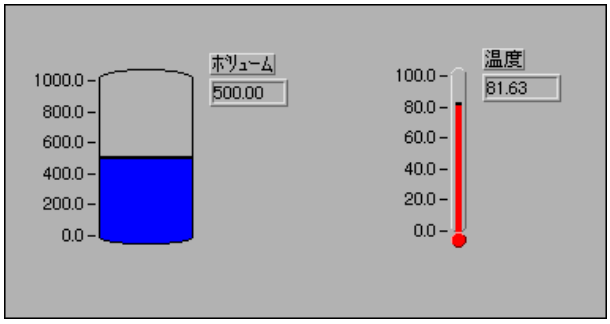
☞ **注**　**制御器パレットが表示されていない場合は、ウィンドウ→制御器パレットを表示を選択してパレットを表示します。また、フロントパネルの空白部にポップアップメニューを表示すると、制御器パレットにアクセスできます。ポップアップメニューを表示するにはマウスを右クリックします（Macintoshの場合は<Option>クリックします）。**

2. **制御器→数値**から**タンク**を選択してフロントパネル上に配置します。
3. ラベルのテキストボックスに［ボリューム］と入力し、フロントパネルのどこかをクリックします。

☞ **注**　**テキストを入力せずにテキストボックスの外側をクリックすると、ラベルが消えます。もう一度ラベルを表示するには、制御器をポップアップして表示→ラベルを選択します。**

4. 0.0～1000.0の間のタンク体積を表示できるようにタンク表示器の目盛りを変更します。
   a. ラベリングツールを使用し、タンクの目盛りの［10.0］をダブルクリックしてハイライト表示にします。
   b. 目盛りに［1000］と入力して、フロントパネルのどこかをマウスボタンでクリックします。中間の目盛りが自動的に付けられます。
5. **制御器→数値**の温度計をフロントパネル上に配置します。この温度計に温度というラベルを付け、目盛りを0～100に変更します。

6. フロントパネルは次の図のようになるはずです。



7. **ウィンドウ→ダイアグラムを表示**を選択してブロックダイアグラムを開きます。**関数**パレットから以下のオブジェクトを選択し、ブロックダイアグラム上に配置します。

☞ **注**　**関数パレットが表示されていない場合は、ウィンドウ→関数パレットを表示を選択してパレットを表示します。また、ブロックダイアグラムの空白部をポップアップすると、関数パレットにアクセスできます。**

8. 以下のそれぞれのオブジェクトをブロックダイアグラム上に配置します。

Process Monitor（`LabVIEW¥Activity`ディレクトリから**関数→VIを選択する**を選択する）— センサやトランスデューサから温度を示す電圧や体積の値を読み込む動作をシミュレーションします。

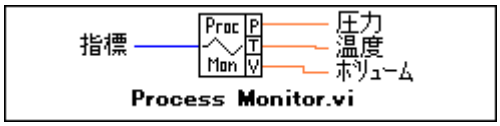
Random Number Generator関数（**関数→数値**）— 0～1の間の数を発生します。

Multiply関数（**関数→数値**）— 2つの数値を乗じて積を返します。この作業では、この関数が2つ必要です。パレットから1つをドロップし、コピーとペーストによりもう1つを作成します。

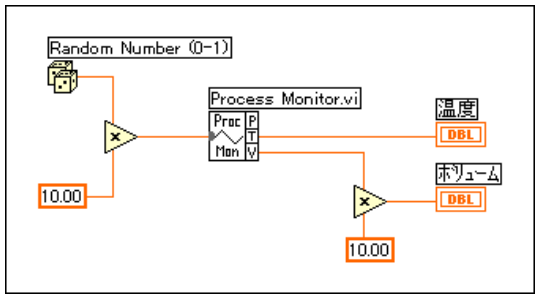
数値定数（**関数→数値**）— この関数も2つ必要です。まず、パレットから1つをドロップします。ラベリングツールを使用して値を`[10.00]`に変更し、コピーしてペーストします。

☞ **注**　**このほかに、配線ツールを使用して関数やVIの端子をポップアップして定数を作成する方法もあります。フローティングメニューから定数を作成を選択すると、適切なデータタイプの定数が表示されます。**

9. 関数やVIの入力と出力を表示するには、**ヘルプ**メニューから**ヘルプを表示**を選択してから各関数やVIの上にカーソルをドラッグします。プロセスモニタVIの場合のヘルプウィンドウを次に示します。



10. 次の図のように、配線ツールを使用してオブジェクトを配線します。



☞ **注**　**ブロックダイアグラム上でオブジェクトを移動するには、ツールパレットの位置決めツールをクリックします。**

11. **ファイル→保存**を選択し、`Temp & Vol.vi`という名前で`LabVIEW¥Activity`ディレクトリにVIを保存します。
12. **実行**ボタンをクリックして、フロントパネルからVIを実行します。ボリュームと温度の値がフロントパネルに表示されるのがわかります。
13. **ファイル→閉じる**を選択してVIを閉じます。

これで作業 2-1 は完了です。

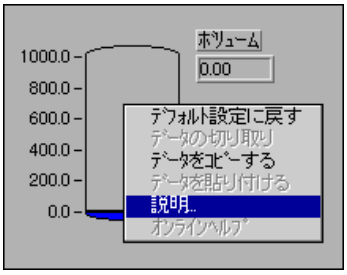
VIの文書作成

VIを文書化するには、**ウィンドウ→VI情報を表示...**を選択してVI情報ダイアログボックスにVIについての説明を入力します。この後、**ウィンドウ→VI情報を表示...**をもう一度選択すると説明を呼び出すことができます。

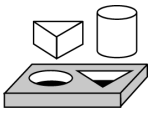
フロントパネルのオブジェクト（またはブロックダイアグラム上のこれらに対応する端子）の説明を編集するには、オブジェクトをポップアップして**データ処理→説明...**を選択します。

☞ **注**　**VIの動作中は、VIや対応するフロントパネルオブジェクトの説明を変更することはできません。**

次の図は、VIの動作中に表示されるポップアップメニューの例です。VIの動作中は、説明を追加したり変更することはできませんが、前に入力した情報はどれでも表示できます。



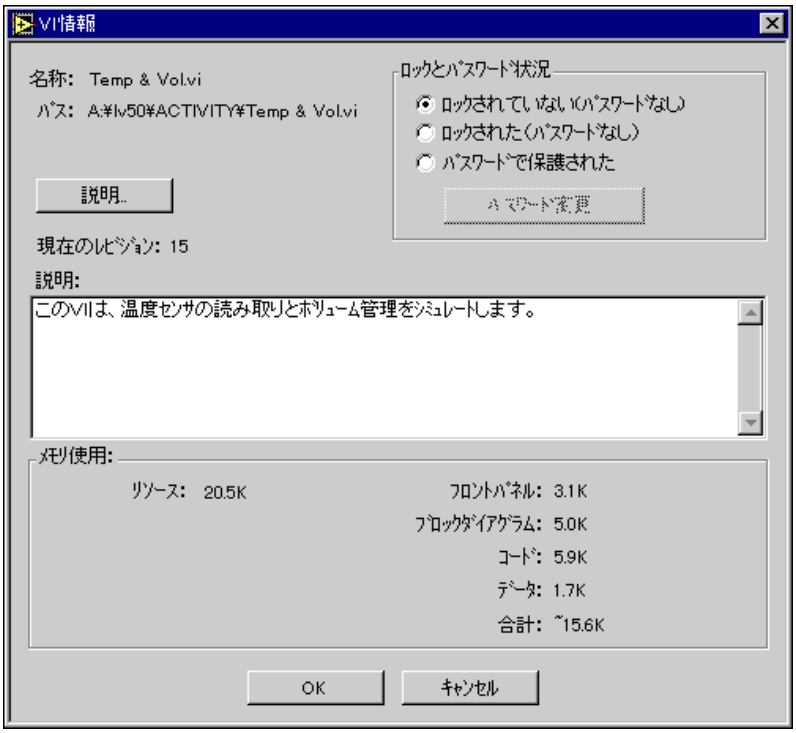
また、フロントパネルオブジェクトの説明を表示するには、ヘルプウィンドウ（**ヘルプ→ヘルプを表示**）を表示してオブジェクトの上にカーソルを移動します。



作業 2-2.　VIを文書化する

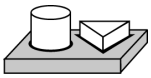
ここでは、前に作成したVIを文書化するのが目的です。

1. LabVIEW¥Activityディレクトリに入っている、作業2-1で作成した`Temp & Vol.vi`を開きます。
2. **ウィンドウ→VI情報を表示...**を選択し、次の図のようにVIに関する説明を入力して**OK**をクリックします。



3. タンクをポップアップして**データ処理→説明 ...** を選択し、次の図のように表示器に関する説明を入力して**OK** をクリックします。

4. 温度計をポップアップして**データ処理→説明...**を選択し、[シミュレーションした測定温度を表示する（℃）]という説明を入力して**OK**をクリックします。
5. **ヘルプ**メニューから**ヘルプを表示**を選択します。ボリューム、温度の順にカーソルを移動します。入力した説明がヘルプウィンドウに表示されることを確認してください。
6. VIを保存して閉じます。



これで作業 2-2 は完了です。

サブVIとは？

サブVIは、テキストベースのプログラミング言語におけるサブルーチンとよく似ています。サブVIとは、他のVIのブロックダイアグラム内で使用されるVIのことです。

アイコンとコネクタを持つVIはどれでも、他のVIの中でサブVIとして使用することができます。ブロックダイアグラムでサブVIとして使用するVIを選択するには、**関数→VIを選択...**を選択します。このオプションを選択するとファイルダイアログボックスが表示され、システム内の任意のVIを選択できます。アイコンやコネクタを含まないVIを開くと、空の正方形のボックスが、呼び出し側のVIのブロックダイアグラムに表示されます。このノードに対して配線することはできません。アイコンとコネクタに関する詳細は、起動時のダイアログボックスからアクセスできる『LabVIEWオンラインチュートリアル』を参照してください。

サブVIはサブルーチンに、サブVIノードはサブルーチン呼び出しに相当します。プログラムにおけるサブルーチン呼び出し命令がサブルーチンそのものではないのと同様、サブVIノード自身はサブVIではありません。1つのブロックダイアグラムに全く同じサブVIノードが複数含まれている場合は、同じサブVIが複数回呼び出されます。

階層ウィンドウ

階層ウィンドウは、メモリ内のすべてのVIに対する呼び出しの階層構造をグラフィック的に表現したものです。この中にはタイプの定義やグローバル変数が含まれます。VIの依存関係を表示するには、VIの呼び出し側とサブVIに関する情報が得られる階層ウィンドウ（**プロジェクト→階層ウィンドウを表示**）を使用します。このウィンドウのツールバーを使用すると、表示された項目に対して何種類もの設定値を構成することができます。次の図に、VI階層ツールバーの例を示します。

以下のオプションにアクセスするには、階層ウィンドウのツールバーのボタン、表示メニュー、またはウィンドウの空白部をポップアップします。階層ウィンドウに関する詳細は、『G プログラミングリファレンスマニュアル』の「第3章　サブ VI を使用する」の「階層ウィンドウを使用する」の項を参照してください。

レイアウトのやり直し — 線の交差が少なく、外観ができるだけ対称になるようにする必要がある場合は、階層ノードに対して連続的に操作した後、ノードを再配置します。フォーカスノードが存在する場合は、サブ VI を示す第1ルートが表示されるよう、ウィンドウをスクロールします。

縦レイアウト — ルートが最も上になるようにして、ノードを上から下に向かって並べ替えます。

横レイアウト — ルートが左側になるようにして、ノードを左から右に向かって並べ替えます。

VI Lib を含む — 階層グラフを、VI ライブラリを含む状態と、VI ライブラリから VI を除外する状態との間で切り替えます。

グローバル変数を含む — 階層グラフを、グローバル変数を含む状態と含まない状態との間で切り替えます。グローバル変数には、複数の VI により使用されるデータが入っています。

Type Def を含む — 階層グラフを、タイプ定義を含む状態と含まない状態との間で切り替えます。タイプ定義は、複数の VI が使用可能なカスタム制御器のマスタコピーです。

また、表示メニューとポップアップメニューには、ツールバーではアクセスできない**すべての VI を表示**オプションと**ラベル**に **VI のフルパスを表示**オプションがあります。

LabVIEW では、階層ウィンドウのオブジェクト上で操作ツールを動かすと、VI アイコンの下にその VI の名前が表示されます。

位置決めツールとスクロールウィンドウツールを切り替えるには、<Tab> キーを使用します。階層ウィンドウからブロックダイアグラムにノードを移動する場合に、この機能が役に立ちます。

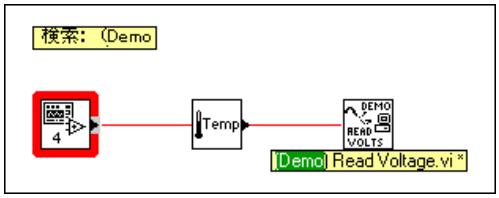
ノードをクリックすると、VI やサブ VI ノードをブロックダイアグラムにドラッグしたり、VI やサブ VI ノードをクリップボードにコピーすることができます。複数のオブジェクトを選択して他のブロックダイアグラムやフロントパネルにコピーするには、VI やサブ VI ノードを <Shift> クリック

します。VIやサブVIノードをダブルクリックすると、そのノードのフロントパネルが開きます。

VIにサブVIが含まれている場合はVIの横に矢印ボタンが表示され、このボタンを使用してサブVIを表示するかどうかを選択できます。赤い矢印ボタンをクリックするか、VIをダブルクリックすると、そのVIに含まれるサブVIが表示されます。VIノード上の矢印ボタンが黒くなっている場合は、すべてのサブVIが表示されていることを示します。また、VIやサブVIノードをポップアップして、サブVIの表示／非表示を切り替える、VIまたはサブVIのフロントパネルを開く、VIアイコンを編集する、などのオプションを含むメニューにアクセスすることもできます。

検索階層

階層ウィンドウ内で現在表示可能なノードを、名前で検索することができます。検索を開始するには、ウィンドウ上のどこかにノード名を入力します。テキストを入力すると、入力したテキストを示す検索文字列が表示され、同時に階層構造内の検索が行われます。次の図に、検索階層を示します。

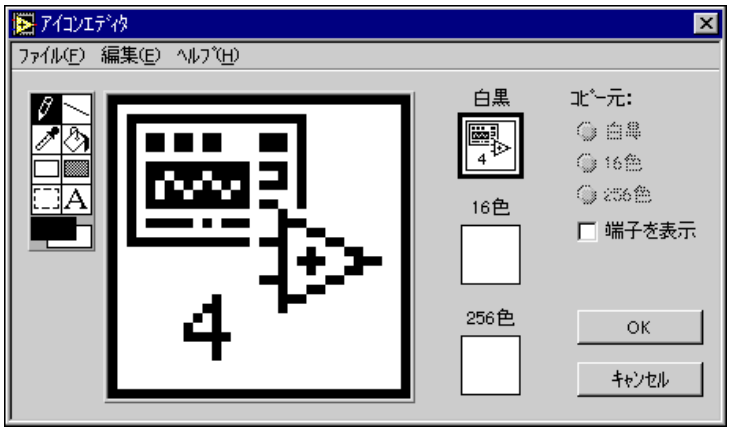


該当するノードが見つかった後、<Enter>を押すと検索文字列に一致する次のノードが検索され、<Shift-Enter>を押すと検索文字列に一致する前のノードが検索されます。

アイコンとコネクタ

どのVIにもデフォルトのアイコンがあり、フロントパネルウィンドウとダイアグラムウィンドウの右上隅に表示されます。VIの場合のデフォルトはLabVIEW VIアイコンであり、数字は、LabVIEWを起動してから開いた新しいVIの数を示します。アイコンをカスタマイズするには、アイコンエディタを使用して個々のピクセルをON/OFFします。アイコンエディタを起動するには、パネルウィンドウの右上隅のデフォルトアイコンをポップアップして**アイコン編集**を選択します。

アイコンエディタウィンドウを次の図に示します。左側のツールを使用して、ピクセル編集領域でアイコンを作成します。編集領域の右側のいずれかのボックスに、実サイズのアイコンの画像が表示されます。



編集領域の左側のツールは、以下の機能を実行します。

鉛筆ツール ― ピクセルごとに描画または消去します。

線ツール ― 直線を描画します。<Shift>を押してこのツールをドラッグすると、縦、横、斜めの線を引くことができます。

スポイトツール ― アイコン内の要素から前景色をコピーします。

フィルバケツツール ― 線で囲んだ部分を前景色で塗りつぶします。

長方形ツール ― 四角形の枠を前景色で描画します。このツールをダブルクリックすると、アイコンの周囲に前景色の枠が描画されます。

フィルされたツール ― 四角形の枠を前景色で描画し、内部を背景色で塗りつぶします。ダブルクリックするとアイコンの周囲に前景色の枠が描画され、内部が背景色で塗りつぶされます。

選択ツール ― 移動、クローン化、その他の変更を行うためにアイコンの領域を選択します。

テキストツール ― アイコンのデザインの中にテキストを入力します。

前景色／背景色 ― 現在の前景色と背景色を表示します。それぞれをクリックするとカラーパレットが表示され、このパレットから新しい色を選択できます。

編集画面右側のボタンは、以下の機能を実行します。

- **OK** ― ユーザが描画したものをVIアイコンとして保存し、フロントパネルに戻ります。
- **キャンセル** ― 変更内容を保存せずにフロントパネルに戻ります。

使用しているモニタのタイプにより、白黒モード、16色モード、256色モード用のアイコンを個別にデザインすることができます。それぞれのバージョンのアイコンを別々にデザインして保存します。エディタのデフォルトは**白黒**になっていますが、他のカラーオプションのいずれかをクリックすればモードを切り替えることができます。

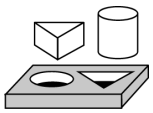
☞ **注**　**カラーアイコンだけをデザインする場合に、VIを`*.lib`ディレクトリ内に入れると、関数パレットのサブパレットにアイコンが表示されません。また、アイコンはモノクロモニタに表示されず、印刷もされません。**

コネクタは、プログラムによるVIへのインタフェースです。パネルの制御器や表示器を使用してサブVIとの間でデータを受け渡す場合、これらの制御器や表示器は、コネクタペーン上の端子を必要とします。接続を定義するには、VIに必要な端子の数を選択し、フロントパネルの制御器や表示器を各端子に割り当てます。

コネクタを定義するには、フロントパネルウィンドウのアイコンペーンのポップアップメニューから**コネクタを表示**を選択します。

コネクタアイコンは、フロントパネルウィンドウの右上隅のアイコンを置換します。LabVIEWは、表示器に対するコネクタペーンの左側の制御器用の端子と右側の表示器用の端子を持つユーザVIに適した端子パターンを選択します。選択される端子の数は、フロントパネル上の制御器と表示器の数によって異なります。

コネクタの各四角形は端子部分であり、VIからの入力にも出力にも使用できます。必要ならば、VI用に別の端子パターンを選択することもできます。このためには、アイコンをポップアップして**コネクタを表示**を選択し、もう一度ポップアップして**パターン**を選択します。



作業 2-3.　アイコンとコネクタを作成する

ここでは、VI用のアイコンとコネクタを作成するのが目的です。

VIをサブVIとして使用するには、VIを他のVIのブロックダイアグラム上に表示するためのアイコンを作成し、さらに入力と出力を接続するためのコネクタペーンを作成する必要があります。LabVIEWには、VIのアイコンを作成したり編集するためのいくつかのツールが用意されています。

VIのアイコンは、他のVIのブロックダイアグラム内ではサブVIとして表示されます。アイコンでは、VIの目的を図記号で表現したり、VIをテキストで説明することができます。

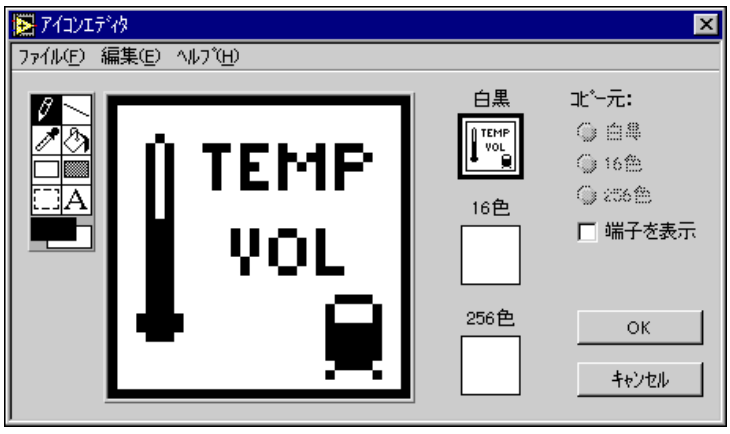
1. `LabVIEW¥Activity`ディレクトリの`Temp & Vol.vi`を開きます。
2. フロントパネルから、右上隅のアイコンをポップアップして**アイコン編集 ...**を選択します。また、アイコンをダブルクリックしてアイコンエディタを起動することもできます。

☞ **注**　　**VI用のアイコン／コネクタにアクセスできるのはダイアグラムのみです。**

3. デフォルトのアイコンを消去します。点線の四角形で表示される選択ツールを使用して、削除する部分をクリックしてドラッグし、<Delete>キーを押します。また、ツールボックス内の網かけの四角形をダブルクリックしてアイコンを消去することもできます。

4. 鉛筆ツールを使用して温度計を描画します。

5. テキストツールを使用してテキストを作成します。フォントを変更するにはテキストツールをダブルクリックします。アイコンは、次の図のようになります。

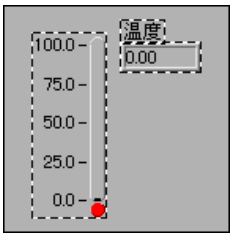


6. OKをクリックしてアイコンエディタを閉じると、新しいアイコンがアイコンペーンに表示されます。

7. フロントパネルのアイコンペーンをポップアップして**コネクタを表示**を選択し、コネクタの端子パターンを定義します。デフォルトでは、フロントパネル上の制御器と表示器の数に基づいて端子パターンを選択します。フロントパネルには2つのオブジェクトがあるため、左の図のようにコネクタには2つの端子があります。

8. コネクタペーンをポップアップして**90度回転**を選択します。コネクタペーンが左の図のように変わることを確認してください。
9. 端子を温度とボリュームに割り当てます。

   a. コネクタの一番上の端子をクリックします。カーソルが自動的に配線ツールに変わり、端子が黒くなります。
   b. 温度表示器をクリックします。次の図のように、表示器の枠が点滅します。選択された端子が、選択された制御器／表示器のデータタイプに対応する色に変わります。



フロントパネルの空いている部分をクリックすると、破線が消えて選択された端子が淡色表示になり、表示器が端子に割り当てられたことを示します。端子が白い場合は、正しく接続されなかったことを示します。

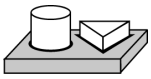
   c. ステップaとbを繰り返して、下の端子をボリューム表示器に割り当てます。
   d. コネクタをポップアップして**アイコンを表示...**を選択します。

10. **ファイル→保存**を選択してVIを保存します。

これで、このVIは完成しましたので、他のVIのサブVIとして使用できます。アイコンは、呼び出し側VIのブロックダイアグラム内のVIを示します。（2つの端子を持つ）コネクタは、温度と体積を出力します。

☞ **注**　コネクタは、サブVIとして使用するときのVIの入力と出力を指定します。フロントパネルの制御器は入力として、表示器は出力としてしか使用できないことを覚えておいてください。

11. **ファイル→閉じる**を選択してVIを閉じます。

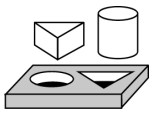


これで作業 2-3 は完了です。

サブVIのオープン、操作、変更

サブVIとして使用されているVIを、呼び出し側VIのブロックダイアグラムから開くには、サブVIアイコンをダブルクリックするか、**プロジェクト→このVIのサブVI**を選択します。呼び出し側VIのすべてのサブVIを含むパレットが表示されますので、開きたいサブVIを選択します。

サブVIを保存するまでは、サブVIに対して行った変更操作により影響を受けるのはメモリ内のVIだけです。変更は、VIの編集に使用したノードだけでなく、サブVIのすべてのバージョンに反映されます。



作業 2-4.　サブVIを呼び出す

ここでは、`Temp & Vol.vi`をサブVIとして使用するVIを作成するのが目的です。

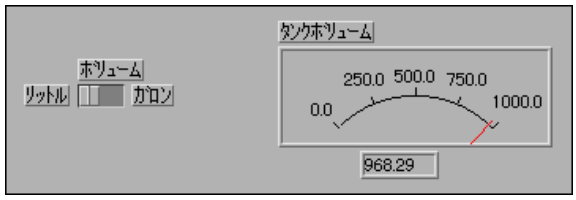
作業2-1で作成したTemp & Vol.VIは、温度と体積を返します。ここでは、スイッチが押されたとき、体積の測定値を読み込んでガロンに変換します。

フロントパネル

1. **ファイル→新規**を選択して新しいフロントパネルを開きます。

2. **制御器→ブール**パレットから横スイッチを選択し、[ボリューム]というラベルを付けます。ラベリングツールを使用して、[リットル]と[ガロン]を表示するフリーラベルをフロントパネルに配置します。

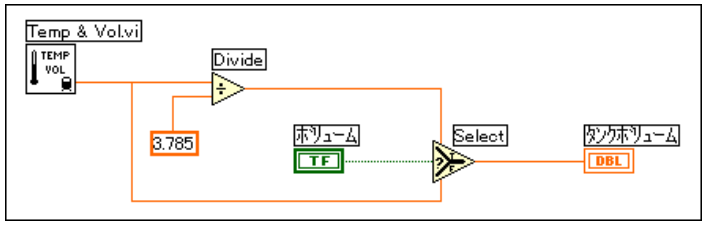
3. **制御器→数値**から計器を選択してフロントパネルに配置し、[タンクボリューム]というラベルを付けます。



4. 0.0～1000.0の範囲の値に合うように、計器の範囲を変更します。操作ツールで上限値をダブルクリックして、10.0から1000.0に変更します。位置決めツールに切り替えて、いずれかの角を外側にドラッグし制御器を拡大することにより、計器のサイズを変更します。

ブロックダイアグラム

5. **ウィンドウ→ダイアグラムを表示**を選択してブロックダイアグラムに移動します。
6. ブロックダイアグラムの空白部をポップアップして**関数→ VI を選択 ...** を選択すると、ダイアログボックスが表示されますので、`LabVIEW¥Activity`ディレクトリの中にある`Temp & Vol.vi`を選択します。ダイアログボックスで**開く**をクリックすると、Temp & Vol.VIはブロックダイアグラム上に配置されます。
7. 次の図のように、他のオブジェクトをブロックダイアグラムに追加します。



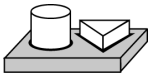
数値定数（**関数→数値**）— 数値定数をブロックダイアグラムに追加します。ラベリングツールを使用して定数に`[3.785]`という値を割り当てます。これは、リットルからガロンに切り替えるときの変換係数です。

Select関数（**関数→比較**）— ブール入力に従ってTRUE入力またはFALSE入力に配線された値を返します。

Divide関数（**関数→数値**）— リットル単位の値を3.785で割ってガロンに変換します。

8. 図のようにダイアグラムオブジェクトを配線します。

9. フロントパネルに戻り、ツールバーの**実行**ボタンをクリックすると、リットル単位の値が計器に表示されます。
10. スイッチをクリックしてガロンを選択し、**実行**ボタンをクリックすると、ガロン単位の値が計器に表示されます。
11. このVIを、`Using Temp & Vol.vi`という名前で`LabVIEW¥Activity`ディレクトリに保存します。



これで作業 2-4 は完了です。

VIのデバッグ方法

VIに不良がある場合はコンパイルや実行ができません。通常、VIに不良が発生するのはVIの作成時と編集時であり、ダイアグラム内のすべてのアイコンを配線すると正常に戻ります。完成したVIにまだ不良がある場合は、**編集**メニューの**不良ワイヤの削除**を選択してみてください。多くの場合、これでVIの不良が解決できます。

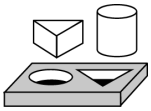
それでもVIが実行可能にならないときは、**実行**ボタンの代わりに**壊れた**矢印が表示されます。エラーの一覧を表示するには、壊れた**実行**ボタンをクリックします。リストに含まれるエラーのいずれかをクリックしてから**検索**をクリックすると、エラーを報告してきたオブジェクトまたは端子がハイライト表示になります。

VIのブロックダイアグラムの実行状態を動画表示するには、**実行のハイライト**ボタンをクリックします。実行ハイライトは、シングルステップモードでブロックダイアグラム内のデータフローをトレースする際によく使用されます。

デバッグをする場合ブロックダイアグラムをノードごとに実行したいことがあります。このような方法をシングルステップといいます。シングルステップモードを有効にするには、**中に入る**ボタンか**飛び越える**ボタンをクリックします。この操作により最初のノードが点滅し、実行準備ができたことを示します。**中に入る**ボタンまたは**飛び越える**ボタンをもう一度クリックして、ノードを実行しこのノードに進みます。ノードがストラクチャまたはVIである場合は、**飛び越える**ボタンを選択すると、ノード内では**シングルステップ動作を行わずに**ノードを**実行**することができます。たとえば、ノードがサブVIである場合に**飛び越える**ボタンをクリックすると、サブVIを実行して次のノードに進みますが、サブVIの実行状態は確認でき

ません。ストラクチャやVIの中もシングルステップ動作させる場合は、**中に入る**ボタンを選択します。

ブロックダイアグラムノードの実行を終了する場合やシングルステップ動作を終了する場合は、**外に出る**ボタンをクリックします。デバッグに関する詳細は、『Gプログラミングリファレンスマニュアル』の「第4章　VIとサブVIの実行およびデバッグ」を参照してください。

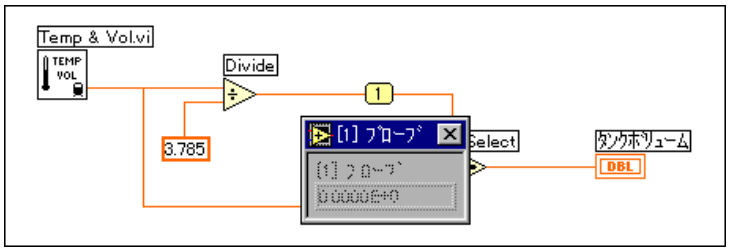


作業 2-5.　LabVIEWでVIをデバッグする

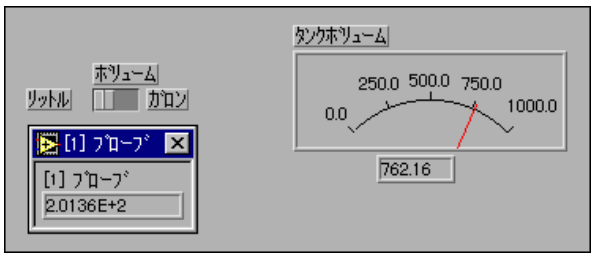
ここでは、プローブツールとプローブウィンドウを使用すること、および実行ハイライト機能を使用してブロックダイアグラム内のデータフローをチェックすることが目的です。

1. `LabVIEW¥Activity`ディレクトリから`Using Temp & Vol.vi`を開きます。
2. **ウィンドウ→ダイアグラムを表示**を選択します。
3. ツールパレットが開いていない場合は、**ウィンドウ→ツールパレットを表示**を選択します。

4. **ツール**パレットからプローブツールを選択します。プローブツールを使用して、Divide 関数から出ているワイヤをクリックします。次の図のように、[[1] プローブ] というタイトルとプローブ番号の付いた黄色のグリフを含むプローブウィンドウが現れます。フロントパネルに切り替えても、プローブウィンドウは開いたままになっています。



5. フロントパネルに戻ります。次の図のようにプローブと体積値の両方を表示できるようにプローブウィンドウを移動します。VI を実行します。[タンクボリューム]にはリットル単位の値が表示されるのに対し、プローブウィンドウにはガロン単位の体積が表示されます。



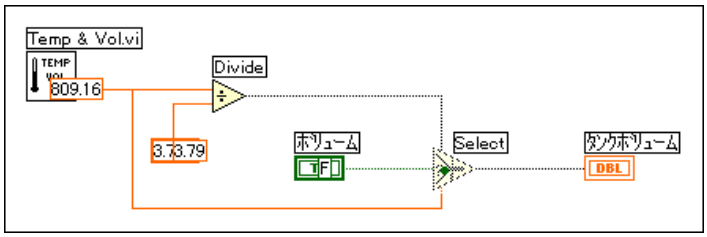
☞ **注**　**画面に表示される体積値が図と異なることがあります。詳しくは、「第３章** ループとチャート」**の「数値の変換」の項を参照してください。**

6. プローブウィンドウのタイトルバーの最上部のクローズボックスをクリックして、プローブウィンドウを閉じます。

もう 1 つの有効なデバッグ方法として、実行ハイライト機能を使用してブロックダイアグラム内のデータフローをチェックする方法があります。

7. VIのブロックダイアグラムに戻ります。

8. ツールバーの**実行のハイライト**ボタンをクリックして実行ハイライトを開始します。**実行のハイライト**ボタンが点灯状態の電球に変わります。
9. **実行**ボタンをクリックしてVIを動作させ、実行ハイライトによってVIのブロックダイアグラムの実行状態が動画で表示されるのを確認してください。バブルが移動して、VI を通るデータフローを示します。また、次のブロックダイアグラムに示す通り、ちょうどワイヤにプローブを当てた場合と同様に、ワイヤ上にデータ値が現れ、そのときワイヤに含まれている値が表示されることを確認してください。

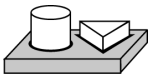


また、グラフィカルコードを一度に 1 ステップずつ進めたい場合は、シングルステップボタンも使用できます。

10. シングルステップ動作を開始するには、ツールバーの**飛び越える**ボタンをクリックします。

11. Temp & Vol サブ VI の中も調べる場合は、ツールバーの**中に入る**ボタンをクリックします。このボタンをクリックすると、ユーザの Temp & Vol サブ VI のフロントパネルとブロックダイアグラムが開きます。VI の実行が終わるまで、**飛び越える**ボタンをクリックします。

12. ブロックダイアグラムの実行を終了するには、ツールバーの**外に出る**ボタンをクリックします。このボタンをクリックすると、ブロックダイアグラムの残りのシーケンスはすべて完了します。



これで作業 2-5 は完了です。

3

ループとチャート

この章では、ストラクチャについて紹介し、チャート、While ループ、For ループについての基本概念を説明します。またこの章では、作業を行いながら以下の内容を学習していきます。

- さまざまなチャートモードについて学習する
- While ループとチャートを使用する
- ブールスイッチの機械的な動作を変更する
- ループのタイミングを調整する
- シフトレジスタを使用する
- マルチプロットチャートを作成する
- For ループを使用する

ストラクチャとは？

ストラクチャはプログラムを制御する要素です。ストラクチャは VI におけるデータフローを制御します。G には、While ループ、For ループ、Case ストラクチャ、シーケンスストラクチャ、フォーミュラノードという 5 つのストラクチャがあります。この章では、While ループストラクチャ、For ループストラクチャ、チャート、シフトレジスタについて紹介します。Case ストラクチャ、シーケンスストラクチャ、フォーミュラノードについては、「第 4 章　Case ストラクチャ、シーケンスストラクチャ、およびフォーミュラノード」で説明します。

While ループと For ループは G によるプログラミングにおける基本的なストラクチャであり、本書の作業や G のサンプルにはほとんど、これらのストラクチャが出てきます。『G プログラミングリファレンスマニュアル』の「第 19 章　ストラクチャ」でも、ループについて詳しく説明しています。

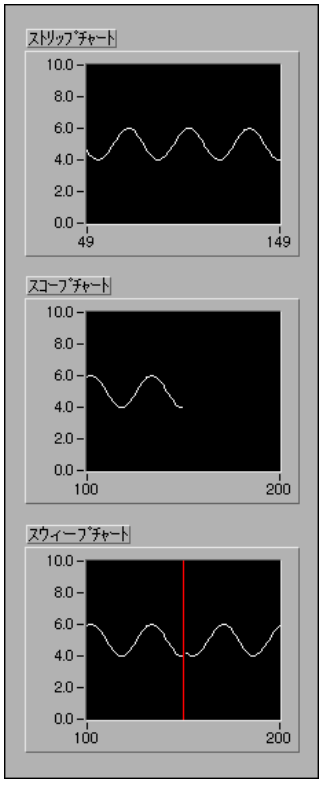
ストラクチャの例については `Examples¥General¥structs.llb` を、チャートの例については `Examples¥General¥Graphs¥charts.llb` を参照してください。

チャート

チャートは、数字をプロットする表示器であり、定期的に新しいデータに更新されます。**制御器→グラフ**パレットには、波形チャートと強度チャートという 2 種類のチャートがあります。データの表示条件に合わせて、あるいは、より多くの情報を表示できるように、チャートをカスタマイズすることができます。チャートでは、スクロールバー、凡例、パレット、デジタル表示、時間目盛りの表示などの機能を使用できます。チャートに関する詳細は、『G プログラミングリファレンスマニュアル』の「第 15 章　グラフ、チャート制御器、および表示器」を参照してください。

チャートモード

次の図は、**データ処理→更新モード**サブメニューで使用可能な 3 つのチャート表示オプション**ストリップチャート**、**スコープチャート**、**スウィープチャート**を示します。デフォルトではストリップチャートモードになっています。

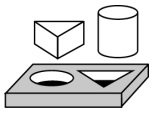


より早くチャートを更新するには

複数の値の配列をチャートに渡すことができます。チャートはこれらの入力を1つのプロット用の新しいデータとして扱います。Examples¥General¥Graphs¥charts.llb に入っている charts.vi の例を参照してください。

重合プロットと積層プロット

1 つのチャートに複数のプロットを表示するとき、1 つの縦軸を使用して行う方法を重合プロットと言い、複数の縦軸を使用して行う方法を積層プロットと言います。Examples¥General¥Graphs¥charts.llb に入っている charts.vi の例を参照してください。



作業 3-1.　チャートのモードに関する実験

ここではVIを、ストリップチャートモード、スコープチャートモード、スウィープチャートモードで動作させた場合のチャートを確認することが目的です。

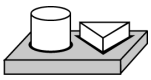
1. 次のライブラリに入っている Charts.vi を開きます。
   LabVIEW¥Examples¥General¥Graphs¥charts.llb
2. VI を実行します。

ストリップチャートモードでは、テープ紙を使用した記録計と同様な目盛りが表示されます。新しい値を受け取るたびに、その値が右側の余白にプロットされ、古い値は左に送られます。

スコープチャートモードでは、オシロスコープのように繰り返しトレース表示が行われます。VI が新しい値を受け取るたびに、その値は最後の値の右側にプロットされます。プロットがプロット領域の右側の端まで達すると、VI はプロットを消去し、左端からもう一度プロットを始めます。スコープチャートは、スクロールに関係する処理のオーバーヘッドがないためストリップチャートに比べるとかなり高速です。

スウィープチャートモードの動作はスコープチャートとよく似ていますが、データが右端に達しても表示は消去されません。その代わりに、VI が新しいデータを追加すると、移動する縦の線が新しいデータの開始部分を示すマーク表示の役割を果たし、この線が画面上を動いていきます。

3. そのままVIを動作させた状態で、いずれかのチャートをポップアップして**更新モード**を選択し、現在のモードを他のチャートのモードに変更します。チャートやモードによる表示の違いを確認してください。
4. VIを停止して閉じます。

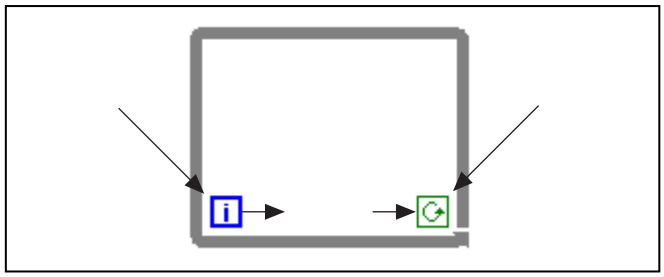


これで作業 3-1 は完了です。

Whileループ

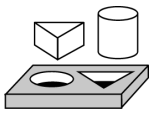
Whileループは、条件が満足されるまでコードの一部分を繰り返し実行するストラクチャで、従来のプログラミング言語では、Doループや Repeat-Until ループに相当します。

次の図に示すWhileループはサイズ変更が可能なボックスで、これを使用すると条件端子（入力端子）に渡されるブール値がFALSEになるまでボックス内のダイアグラムを実行することができます。VIは、繰り返しの各回の終わりで条件端子をチェックします。したがって、Whileループは必ず1回は実行されます。繰り返し端子は出力の数値端子であり、ループが実行された回数を出力します。ただし、繰り返しカウントは常に0から始まりますので、ループが一度実行された時点での繰り返し端子の出力は0になります。



Whileループは、次のような疑似コードに相当します。

```
Do
Execute Diagram Inside the Loop (which sets the
condition)
While Condition is TRUE
```

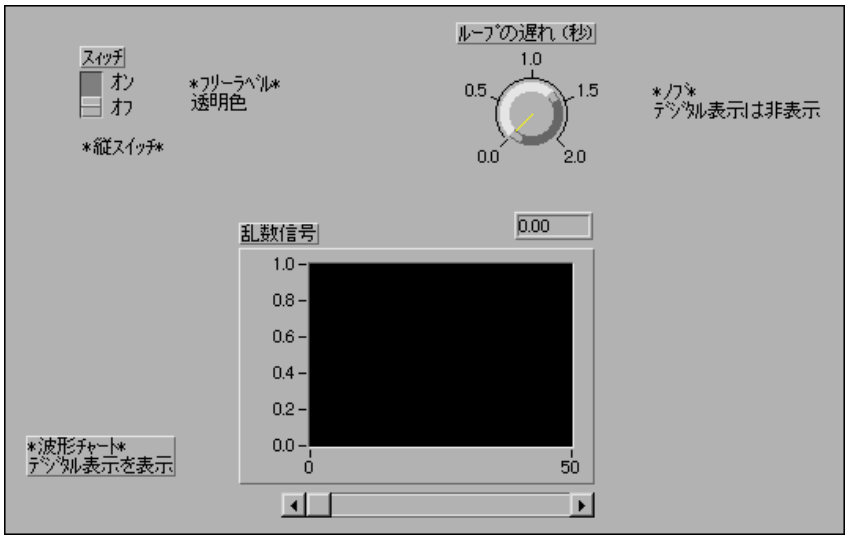


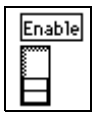
作業 3-2.　Whileループとチャートを使用する

ここでは、While ループとチャートを使用して、リアルタイムでデータを集録して表示することが目的です。

乱数データを発生してチャート上に表示するVIを作成します。フロントパネルのノブ制御器はループ速度が0～2秒の間になるように調整し、スイッチはVIを停止します。VIを実行するたびにスイッチをONにしなくても済むように、スイッチの機械的な動作を変更します。最初は、次の図に示すフロントパネルから始めます。

フロントパネル





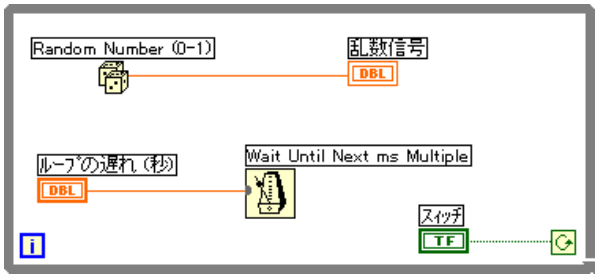
1. **ファイル→新規**を選択して新しいフロントパネルを開きます。
2. 縦のスイッチ（**制御器→ブール**）をフロントパネル上に配置します。スイッチにスイッチというラベルを付けます。
3. ラベリングツールを使用してON/OFF用のフリーラベルを作成します。ラベリングツールを選択してラベルテキストを入力します。左側に表示されているカラーツールを使用し、カラーパレットの左下隅の T を選択してフリーラベルの枠を透明にします。
4. 波形チャート（**制御器→グラフ**）をフロントパネル上に配置します。チャートに [乱数信号] というラベルを付けます。チャートにはリアルタイムでデータが表示されます。

注　必ず波形グラフではなく波形チャートを選択するようにしてください。波形チャートはグラフパレットの一番左側に表示されます。

5. チャートをポップアップして**表示→パレット**、および**表示→凡例**を選択し、パレットと凡例を非表示にします。デジタル表示には最新の値が表示されます。続いて、チャートをポップアップして**表示→デジタル表示**、および**表示→スクロールバー**を選択します。
6. チャートの目盛りを[0.0]～[1.0]に修正します。ラベリングツールを使用して上限値の[10.0]を[1.0]に変更します。
7. ノブ（**制御器→数値**）をフロントパネル上に配置し、[ループの遅れ（秒）]というラベルを付けます。このノブはWhileループのタイミングを制御します。ノブをポップアップして**表示→デジタル表示**を選択し、デジタル表示を非表示にします。
8. ノブの目盛りを変更します。ラベリングツールを使用してノブの外側にある目盛りの[10.0]をダブルクリックして[2.0]に変更します。

ブロックダイアグラム

9. ブロックダイアグラムを開き、次の図のようなダイアグラムを作成します。



a. **関数→ストラクチャ**からWhileループを選択してブロックダイアグラム上に配置します。Whileループはサイズ変更可能な四角形で、ダイアグラムには直接ドロップされませんが、これを配置してサイズを変更することができます。このためには、すべての端子より上でかつ左の部分をクリックします。マウスボタンを押したまま、端子を囲む四角形を外側にドラッグします。

b. **関数→数値**から Random Number (0-1) に関数を選択します。

c. ブロックダイアグラムに示された通り、ダイヤグラムを配線します。Random Number (0-1) 関数を乱数信号チャートの端子に、スイッチを While ループの条件端子に接続します。ループの遅れ端子は、ここでは配線しないでおきます。

10. フロントパネルに戻り、操作ツールで縦のスイッチをクリックして ON にします。

11. VI を、`Random Signal.vi` という名前で `LabVIEW¥Activity` ディレクトリに保存します。

12. VI を実行します。

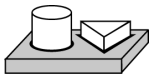
While ループは無限のループストラクチャです。指定された条件が TRUE である限り、ループ内のダイアグラムが実行されます。この例では、スイッチが ON (TRUE) になっている限り、ダイヤグラムは乱数を発生してチャートに表示する動作を続けます。

13. 縦のスイッチをクリックして VI を停止します。スイッチを OFF にすると、ループ条件端子に値 FALSE が送られ、ループが停止します。

14. チャートをスクロールします。マウスボタンでいずれかの矢印をクリックして押し続けます。

15. 表示バッファをクリアしてチャートをリセットするには、チャートをポップアップして**データ処理→チャートをクリア**を選択します。

☞ **注**　**デフォルトの表示バッファサイズは 1024 点です。このバッファサイズを増減するには、チャートをポップアップして*チャート記録の長さ ...* を選択します。この機能は、VI が動作していない時のみ使用できます。**



これで作業 3-2 は完了です。

ブールスイッチの機械的な動作

VIを実行するたびに、縦スイッチをONにしてからツールバーの**実行**ボタンをクリックしなければなりません。Gでは、ブール制御器の機械的な動作を修正することができます。

ブール制御器の機械的な動作としては、次の6つの選択肢が用意されています。

- 押したときにスイッチが切り替わる
- 放したときにスイッチが切り替わる
- 押している間だけスイッチが切り替わる
- 押したときにスイッチがラッチされる
- 放したときにスイッチがラッチされる
- 押している間だけスイッチがラッチされる

各ブールスイッチとそれぞれの機械的な動作の説明を、以下に示します。

押されたときにスイッチが切り替わる動作 ― 操作ツールで制御器をクリックするたびに制御器の値が変わります。この動作は、天井の照明のスイッチと似ており、VIが制御器を読み込む頻度には影響されません。

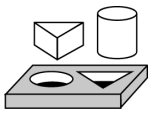
放されたときにスイッチが切り替わる動作 ― 制御器のグラフィック上の範囲内でマウスをクリックする際、マウスボタンを放した後に制御器の値が変わります。この動作は、VIが制御器を読み込む頻度には影響されません。この動作はダイアログボックスのチェックマークをクリックする場合と似ており、ハイライト表示にはなりますが、マウスボタンを放すまでは変わりません。

押している間だけスイッチが切り替わる動作 ― 制御器をクリックすると制御器の値が変わります。マウスボタンを放すまでは新しい値が保持され、マウスボタンを放すと元の値に戻ります。この動作は、ドアのベルに似ており、VIが制御器を読み込む頻度には影響されません。

押したときにスイッチがラッチされる動作 ― 制御器をクリックすると制御器の値が変わります。VIが一度制御器を読み込むまで新しい値が保持され、VIが一度制御器を読み込むと制御器はデフォルト値に戻ります（マウスボタンが押したままになっているかどうかにかかわらず、この動作が行われます）。この動作は、回路ブレーカの動作に似ており、Whileループを停止する場合や、制御器を設定したときに1回だけVIに動作を何か行わせる場合に役立ちます。

放したときにスイッチがラッチされる動作 ― マウスボタンを放した後だけ、制御器の値が変わります。VIが一度値を読み込むと、制御器は元の値に戻ります。この動作では、少くとも 1 回は必ず新しい値になります。放したときにスイッチが切り替わる動作の場合と同様、この動作もダイアログボックスのボタンの動作に似ており、この動作でクリックするとボタンがハイライト表示になり、マウスボタンを放すと測定値がロックされます。

押している間だけスイッチがラッチされる動作 ― 制御器をクリックすると制御器の値が変わります。この値はVIが一度値を読み込むか、またはマウスボタンを放すかのどちらか遅い方の動作が行われるまで保持されます。

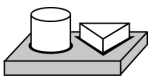


作業 3-3.　ブールスイッチの機械的な動作を変更する

ここでは、ブールスイッチのさまざまな機械的動作を実験することが目的です。

1. `LabVIEW¥Activity` ディレクトリから、作業 3-2 で保存した `Random signal.vi` を開きます。スイッチのデフォルト値はFALSE です。
2. VI を停止する場合にだけ使用されるように縦スイッチを修正します。VI を実行するたびにスイッチを ON にしなくて済むように、スイッチを変更します。
   a. 操作ツールで縦スイッチを ON にします。
   b. スイッチをポップアップして**データ処理→現在の設定をデフォルト設定にする**を選択します。これにより、ON 位置がデフォルト値となります。
   c. スイッチをポップアップして**機械的動作→押されたらラッチ**を選択します。
3. VI を実行します。スイッチをクリックすると集録動作が停止します。スイッチは一時的に OFF 位置に変わり、ON 位置にリセットされます。
4. VI を保存します。

☞ **注**　**LabVIEW には、これらの動作を示す例が含まれており、これは `Examples¥General¥Controls¥booleans.llb` の中に、`Mechanical Action of Booleans.vi` という名前で入っています。**



これで作業 3-3 は完了です。

タイミング

前の作業でVIを実行したとき、Whileループは最高の速度で実行されましたが、**関数→時間＆ダイアログ**パレットに含まれる関数を使用すると、実行速度を遅くして特定の周期で繰り返すようにできます。

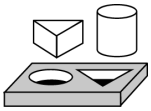
タイミング関数では、時間はミリ秒（ms）単位で表現されますが、ユーザのオペレーティングシステムによっては、このレベルのタイミング精度を保持できないものもあります。

•（Windows 95/NT）タイマの分解能は1msですが、この値はハードウェアによって異なり、80386のような遅い速度のシステムではタイミングの分解能が低くなることがあります。

•（Windows 3.1）タイマのデフォルトの分解能は55msです。分解能が1msになるようにLabVIEWを構成するには、**編集→環境設定**...を選択し、パスリングから性能＆ディスクを選択し、Use Default Timerチェックボックスをチェック解除します。オペレーティングシステムの負荷が大きくなるLabVIEWはデフォルトでは1msの分解能を使用しません。

•（Macintosh）QuickTimeの拡張機能がない68Kシステムの場合、タイマの分解能は16 2/3ms（1/60秒）です。Power Macintoshの場合やQuickTimeがインストールされている場合、タイマの分解能は1msとなります。

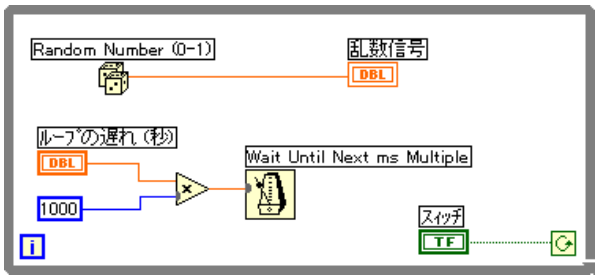
•（UNIX）タイマの分解能は1msです。



作業 3-4.　ループのタイミングを調整する

ここでは、繰り返し処理の時間がミリ秒単位で指定した時間よりも短くならないように、ループのタイミングを調整することが目的です。

1. `LabVIEW¥Activity` ディレクトリから、作業 3-3 で修正し保存した `Random Signal.vi` を開きます。
2. 次の図に示す通り、ノブで指定した時間間隔で新しい乱数を発生するように、VIを修正します。

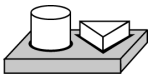


Wait Until Next ms Multiple 関数（**関数→時間＆ダイアログ**）— ノブ端子に 1,000 をかけて秒単位のノブの値をミリ秒単位に変換します。この値は、Wait Until Next ms Multiple 関数の入力として使用されます。

Multiply 関数（**関数→数値**）— Multiply 関数はノブの値に 1000 をかけて秒をミリ秒に変換します。

数値定数（**関数→数値**）— 数値定数は、ミリ秒単位の値を得るためにノブの値にかける定数を保持しています。したがって、ノブの値が 1.0 である場合、ループは 1000 ミリ秒ごとに（1 秒に 1 回ずつ）実行されます。

3. VI を実行します。ノブを回すとループの遅延の値が変化します。ループの遅延の影響で乱数信号の表示が変化することを確認してください。
4. この VI を、`Random Signal with Delay.vi` という名前で、`LabVIEW¥Activity` ディレクトリに保存し、VI を閉じます。

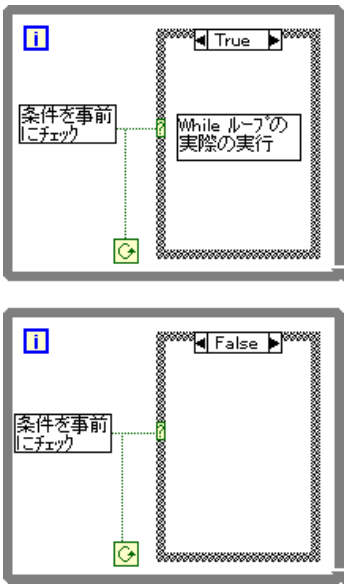


これで作業 3-4 は完了です。

最初の繰り返しでのコード実行を防止する

While ループは少くとも必ず 1 回は実行されますが、これは、G ではダイアグラムの実行後に、ループを継続するかどうかのテストを実行するからです。条件端子を事前にテストする While ループを作成するには、ループの内側に Case ストラクチャを入れます。While ループ内のコードを実行してはならない場合に FALSE 条件用のサブダイアグラムが実行されるよう、ブール入力を Case ストラクチャに配線します。Case ストラクチャに関する詳細は、「第 4 章　Case ストラクチャ、シーケンスストラクチャ、およびフォーミュラノード」を参照してください。

TRUE 条件用のサブダイアグラムには、While ループの動作が含まれます。継続を判定するテストは Case ストラクチャの外側で行われ、結果は While ループの条件端子と Case ストラクチャのセレクタ端子に配線されます。次の図で、ラベルが事前にテストされる条件を示します。

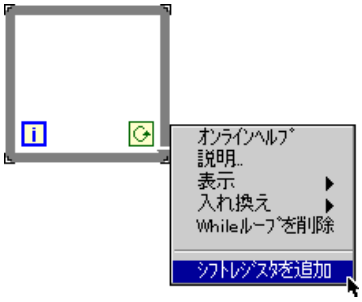


この例は、次に示す疑似コードと同じ結果になります。

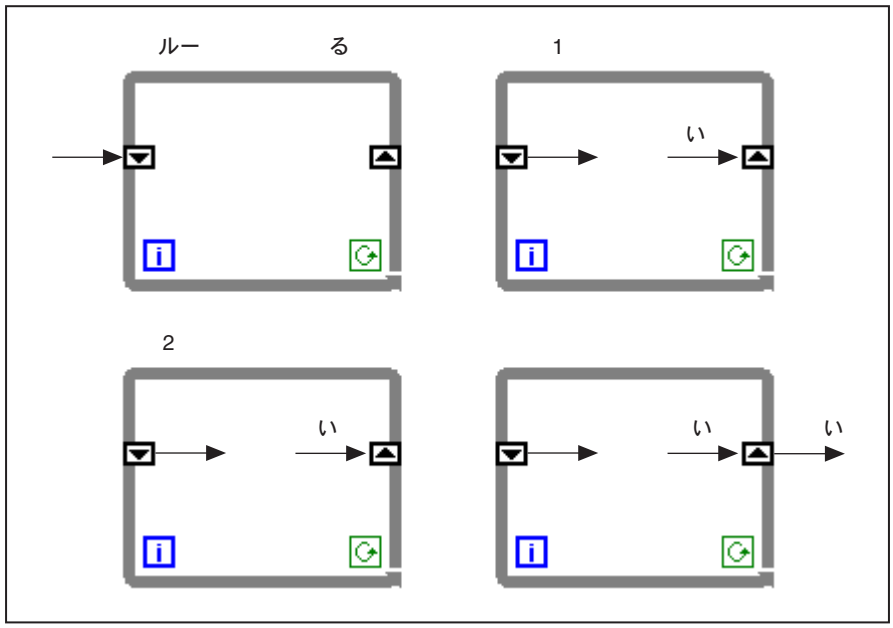
```
While (pretest condition)
Do actual work of While Loop
Loop
```

シフトレジスタ

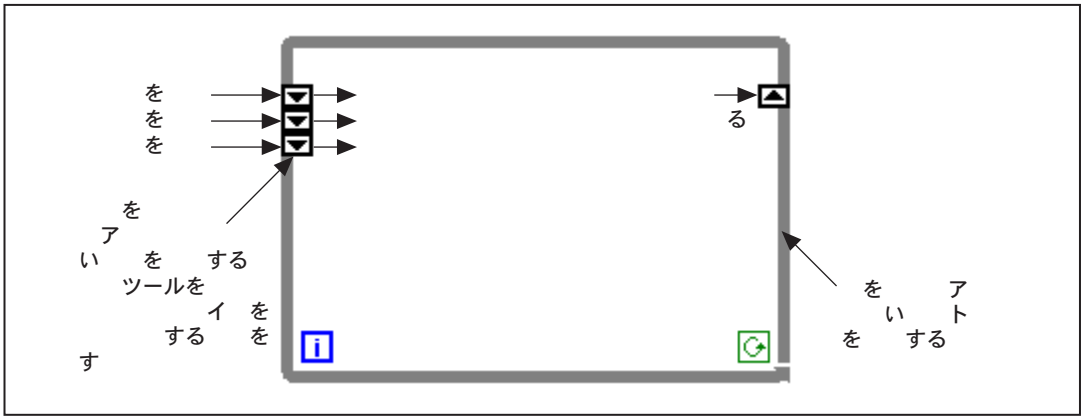
シフトレジスタは（While ループと For ループで使用可能）、1 回の繰り返しから得た値を次の繰り返しに転送します。シフトレジスタを作成するには、ループの左右いずれかの枠をポップアップして**シフトレジスタを追加**を選択します。

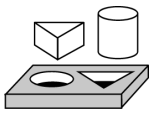


シフトレジスタにはループ枠の左右に一対の端子があり、互いに向かい合っています。右側の端子には、繰り返し処理の完了時のデータが格納されます。次の図に示すように、そのデータは繰り返しの終わりにシフトされ、次の繰り返しの開始時に左側の端子に表示されます。シフトレジスタは、数値、ブール値、文字列、配列など、どのデータタイプでも保持できます。シフトレジスタは、ユーザがシフトレジスタに配線した最初のオブジェクトのデータタイプに自動的に適応します。



シフトレジスタの構成を変更して、前の数回の繰り返しでの値を記憶させておくことができます。この機能は、データポイントの平均を計算する場合に役立ちます。さらに別の端子を作成して前の繰り返した時の値にアクセスするには、左右いずれかの端子をポップアップして**要素を追加**を選択します。たとえば次の図のように、シフトレジスタの左の端子に 3 つの要素がある場合は、最後の3回の繰り返しの値にアクセスできます。



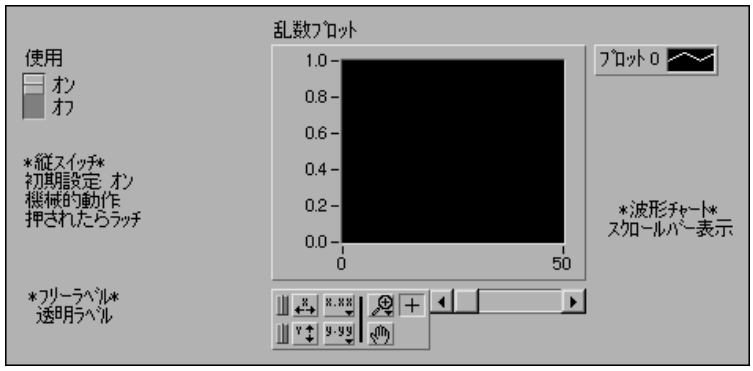


作業 3-5.　シフトレジスタを使用する

ここでは、移動平均をチャートに表示するVIを作成することが目的です。

フロントパネル

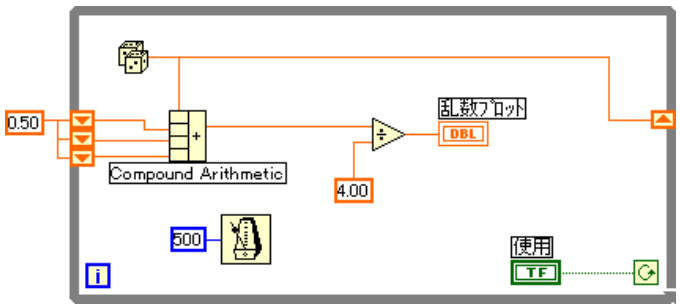
1. 新しいフロントパネルを開き、次の図のようなオブジェクトを作成します。



2. 波形チャートの目盛りの範囲を[0.0]～[2.0]に変更します。
3. 縦のスイッチを追加した後、このスイッチをポップアップして**機械的動作→押されたらラッチ**を選択し、さらに**操作→現在の設定をデフォルトに設定する**を選択してON状態をデフォルトとして設定します。

ブロックダイアグラム

4. 次の図のようなブロックダイアグラムを作成します。



5. ブロックダイアグラムにWhileループ（**関数→ストラクチャ**）を追加して、シフトレジスタを作成します。

   a. Whileループの左右いずれかの枠をポップアップして**シフトレジスタを追加**を選択します。

   b. シフトレジスタの左の端子をポップアップして**要素を追加**を選択し別の要素を追加します。これと同じ方法で3番目の要素を追加します。

Random Number (0-1)関数（**関数→数値**）— この関数は0～1の範囲の乱数データを発生します。

Compound Arithmetic 関数（**関数→数値**）— この作業で、Compound Arithmetic関数は2回の繰り返しからの乱数の合計を返します。より多くの入力を追加するには、入力をポップアップして入力端子を追加を選択します。

Divide関数（**関数→数値**）— この作業では、Divide関数は最後の4つの乱数の平均を返します。

数値定数（**関数→数値**）— Whileループの繰り返しの各回において、Random Number (0-1)関数は1つの乱数値を発生します。VIは、この値をシフトレジスタの左の端子に格納された最後の3つの値に加算します。Random Number (0-1)関数は結果を4で割って値（現在の値と前の3つの値の和）の平均を求めます。平均値は波形チャートに表示されます。

Wait Until Next ms Multiple関数（**関数→時間＆ダイアログ**）— この関数は、ループの繰り返しの各回が、入力されたミリ秒単位の時間より短くならないようにします。この作業の場合の入力は500ミリ秒です。アイコンをポップアップして**表示→ラベル**を選択すると、Wait Until Next ms Multipleというラベルが表示されます。

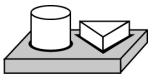
6. Wait Until Next ms Multiple関数の入力をポップアップして**定数を作成**を選択します。数値定数が表示され、この関数に自動的に配線されます。

7. ラベルに[500]と入力します。Wait Until Next ms Multiple関数に配線された数値定数は、500ミリ秒（1/2秒）の待ち時間を指定します。したがって、ループは1/2秒に1回ずつ実行されます。

VIがシフトレジスタを乱数で初期化することを確認してください。シフトレジスタ端子を初期化しない場合、シフトレジスタ端子にはデフォルト値または前回の実行時の最後の値が含まれ、最初のいくつかの平均値は意味を持ちません。

8. VIを実行して動作を確認してください。
9. このVIを、`Random Average.vi`という名前で`LabVIEW¥Activity`ディレクトリに保存します。

☞ **注**　古いデータやデフォルトのデータが現在のデータ測定値に混入しないよう、忘れずにシフトレジスタを初期化してください。

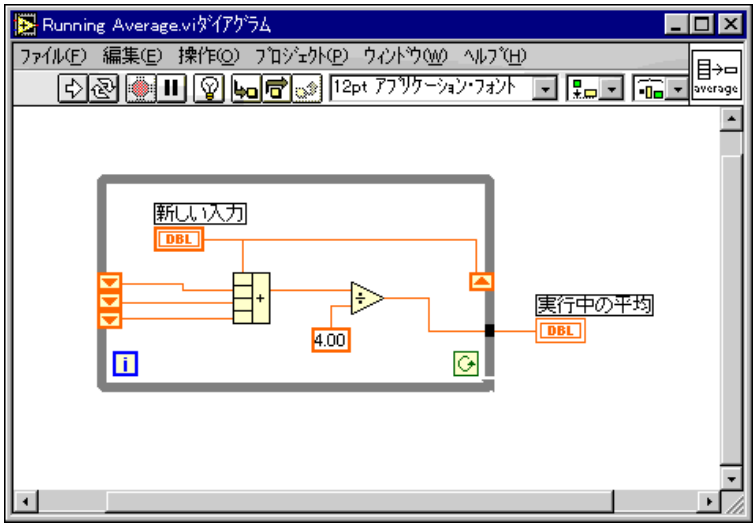


これで作業 3-5 は完了です。

初期化されていないシフトレジスタを使用する

シフトレジスタを初期化するには、WhileループまたはForループの外側からシフトレジスタの左の端子に値を配線します。ただし、場合によっては、VIが実行されるたびにシフトレジスタの初期出力が前回実行時の最後の値になるように、ループとシフトレジスタを使用してVIを繰り返し実行したい場合があります。このためには、ループの外側からシフトレジスタの左の端子に配線しないでおきます。シフトレジスタの左の端子への入力を配線しないでおくと、以後のVI実行の各回の間でステータス情報が保存されます。

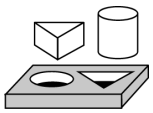
次の図は、4つのデータポイントの移動平均を計算するサブVIの例を示します。VIは初期化されていないシフトレジスタ（および3つの付加要素）を使用して、前のデータポイントを格納します。



VIが呼び出されるたびに、新しい入力と前の3つの値から[実行中の平均]が計算されます。続いて新しい値はシフトレジスタに保存され、前の2つの値がシフトレジスタ内で上に送られます。左のシフトレジスタの入力側に配線された入力値はありませんので、3つの値はすべて、VIが次に実行されるときのために保持されます。

このサブVIは条件端子に何も配線されていませんので、呼び出された場合は必ず1回だけ実行されます。このサブVIの中のWhileループは、複数回ループさせるためではなく、各呼び出しの間にループのシフトレジスタに値を格納するために使用されています。

Running Average VIがメモリにロードされると、初期化されていないシフトレジスタは自動的に0に設定されます。シフトレジスタがブール値に配線されている場合、初期値はFALSEとなります。

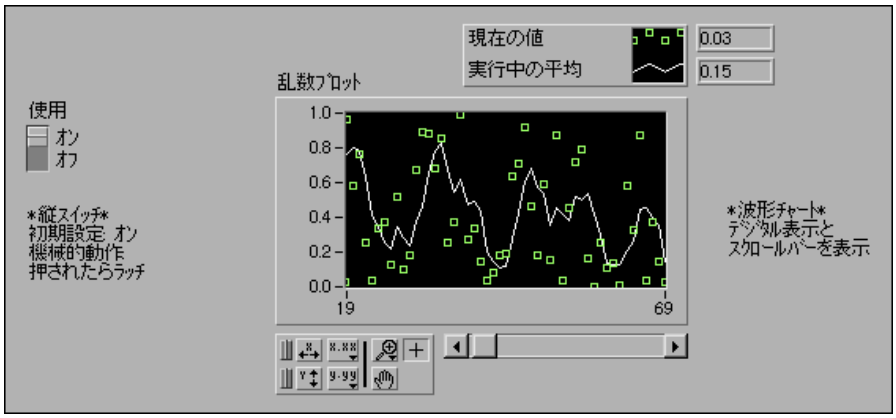


作業 3-6.　マルチプロットチャートを作成する

ここでは、複数のプロットに対応できるチャートを作成することが目的です。

フロントパネル

1. 作業3-5で作成した`Running Average.vi`を開きます。
2. フロントパネルを次の図のように修正します。



a. 位置決めツールを使用して2つのプロットが含まれるように凡例を拡張します。

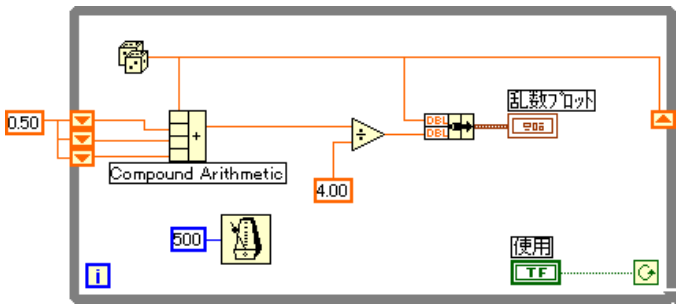
b. チャートをポップアップして**表示→デジタル表示**を選択し、デジタル表示を表示します。必要ならば凡例を移動します。

c. ラベリングツールでラベルをダブルクリックして新しいテキストを入力し、名前をPlot 0から[現在の値]に変更します。ラベル領域のサイズを変更するには、位置決めツールを使用して左側のいずれかの角をドラッグします。同じ方法で名前をPlot 1から実行中の平均に変更します。

d. [現在の値]プロットに対しては、補間を非接続に、ポイントのスタイルを四角形に、色を緑に変更します。凡例をポップアップするとプロットのスタイルと色を変更できます。

ブロックダイアグラム

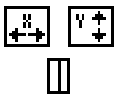
3. 平均値と現在の乱数の両方を同じチャートに表示できるよう、次の図のようにブロックダイアグラムを修正します。



Bundle 関数（**関数→クラスタ**）― この作業で、Bundle 関数はチャートにプロットするために平均値と現在値をまとめます。Bundle 関数をブロックダイアグラム内に配置すると、左図のようなバンドルノードが表示されます。サイズ変更カーソル（関数の角に位置決めツールを移動するとサイズ変更カーソルになります）を使用してノードを拡大すると、別の要素を追加できます。

☞ **注**　**Bundle 関数への入力の順序によって、チャートでのプロットの順序が決まります。たとえば、生データを Bundle 関数の一番上の入力に、平均値を一番下の入力に配線した場合、最初のプロットは生データに対応し、2 番目のプロットは平均値に対応します。**

4. フロントパネルから VI を実行します。VI は、チャートに 2 つのプロットを表示します。2 つのプロットは重ねて表示されますので、同じ縦軸を共有することになります。
5. ブロックダイアグラムから、実行ハイライトを ON にした状態で VI を実行し、シフトレジスタのデータを確認します。
6. 実行ハイライトを OFF にして、フロントパネルから VI を実行します。VI の実行中に、パレットのボタンを使用してチャートを修正します。チャートをリセットする、X 軸や Y 軸の目盛りを設定する、表示形式を変更するなどの操作は、いつでも行えます。また、グラフやチャートをスクロールして他の部分を表示したり拡大することもできます。

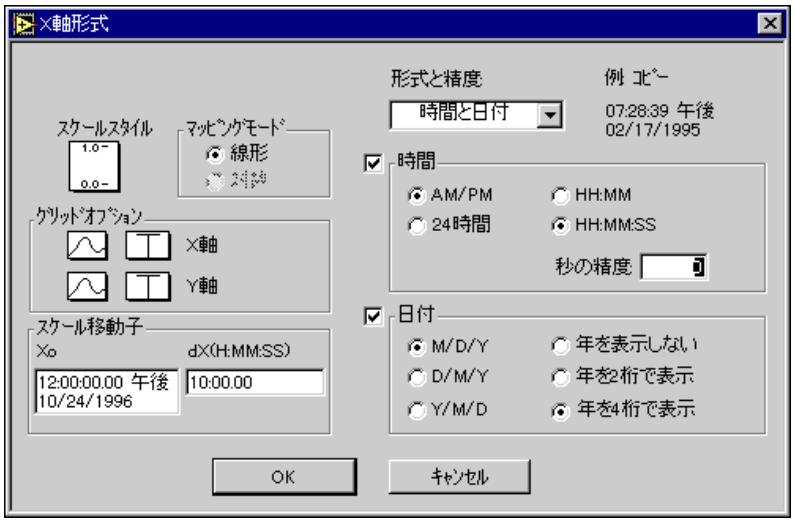


X ボタン Y ボタンを使用すると、X 軸や Y 軸の目盛りをそれぞれ再設定することができます。グラフでいずれかの目盛りが連続的に自動設定されるようにしたい場合は、各ボタンの左のロックスイッチをクリックすると、オートスケーリング機能がロックされます。

その他のボタンを使用すると、軸のテキストの精度を変更したり、チャートの操作モードを制御することができます。これらのボタンを使用してその動作を調べたり、表示する部分をスクロールしたり、チャートの一部を拡大してみてください。

7. 時刻の絶対値または相対値のいずれかを示すように、波形チャートの目盛りをフォーマットします。x 軸の目盛りの時刻のフォーマットを選択するには、x 軸の目盛りをポップアップして**形式...** を選択します。

   a. 絶対値の時刻を選択するには、**形式と精度**メニューリングから**時間と日付**オプションを選択します。こうすると、ダイアログボックスが次の図のように変わります。波形チャートが特定の時刻でスタートし特定の周期で増加していくようにするには、それぞれ *Xo* と *dX* の値を編集します。

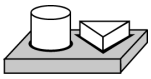


   b. 上図のようにチャートをフォーマットすると、データ表示は 1996 年 10 月 24 日の正午から始まり 10 分ごとに増加していきます。

☞ **注**　**多くの場合、軸のテキストのフォーマットを変更すると、その軸に対して最初に設定されていたよりも多くの物理的容量が必要になります。また、軸を変更すると、テキストが最大サイズより大きくなり、波形が正しく表示されないことがあります。このような問題を解決するには、サイズ変更カーソルを使用してチャートの表示領域を小さくします。**

8. 相対値による時刻フォーマットを選択するには、**形式と精度**メニューリングから**数値**を選択します。さらに、ダイアログボックスの**相対時間（秒）**オプションを選択すれば、時刻を秒で表示できます。次の図のようにダイアログボックスを修正してOKを選択します。

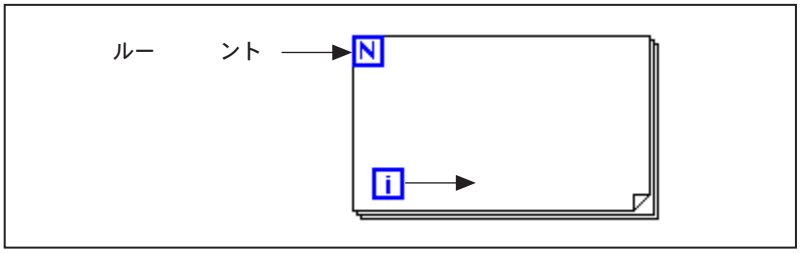
9. VIを実行します。
10. このVIを、`Multiple Random Plot.vi`という名前で`LabVIEW¥Activity`ディレクトリに保存します。



これで作業 3-6 は完了です。

Forループ

Forループは、コードの一部分を指定された回数だけ実行します。Forループはサイズ変更が可能で、Whileループと同様に直接的にブロックダイアグラムにドロップされることはありません。その代わりに、ブロックダイアグラムにForループを示す小さいアイコンが表示されますので、Forループのサイズや位置を変更できます。このためにはまず、すべての端子より上でかつ左にある部分をクリックします。マウスボタンを押したまま、Forループ内に配置したい端子を囲む四角形をドラッグします。マウスボタンを放すと、ユーザが選択したサイズと位置にForループが作成されます。Forループをブロックダイアグラム内に配置するには、**関数→ストラクチャ**からForループを選択します。



For ループは、ダイアグラムの For ループで囲まれた部分をあらかじめ決められた回数だけ実行します。For ループの 2 つの端子について、以下に説明します。

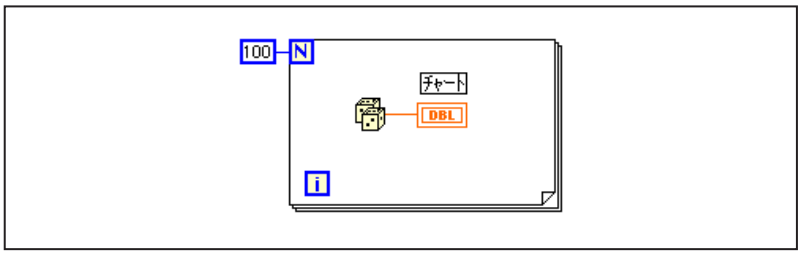
N　カウント端子（入力端子）— カウント端子はループを実行する回数を指定します。

i　繰り返し端子（出力端子）— 繰り返し端子には、ループの実行回数が格納されています。

For ループは、次に示す疑似コードに相当します。

```
For i = 0 to N-1
Execute Diagram Inside The Loop
```

次の図は、100 個の乱数を発生してチャート上に点を表示する For ループを示します。

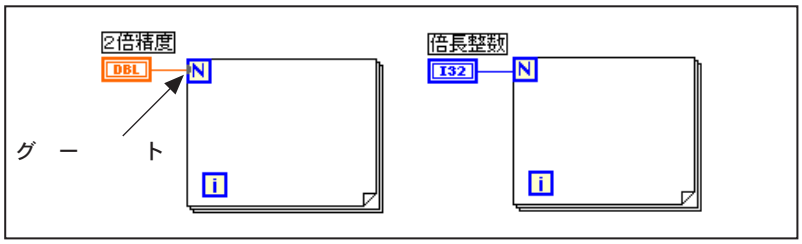


数値変換

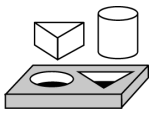
今まで使用してきた数値制御器や表示器はすべて、32 ビットで表現される倍精度浮動小数点数でしたが、G では、数値を整数（バイト、ワード、倍長）や浮動小数点数（単精度、倍精度、拡張精度）として表現することができます。デフォルトの場合、数値は倍精度浮動小数点数として表現されます。

データタイプの異なる 2 つの端子を配線する場合、G は一方の端子の表現方法をもう一方の端子と同じ表現方法に変換します。変換された端子には**強制ドット**というグレーのドットが付きます。

たとえば、For ループのカウント端子の場合を考えてみます。端子は倍長整数で表現されています。倍精度の浮動小数点数をこのカウント端子に配線するとその数値は倍長整数に変換されます。最初の For ループのカウント端子のグレーのドットを確認してください。



注　VI が浮動小数点数を整数に変換する場合、最も近い整数に丸めます。ある数値が 2 つの整数のちょうど中間にある場合は、最も近い偶数の整数に丸められます。たとえば、VI では 6.5 は 6 に、7.5 は 8 に丸められます。これは、IEEE 規格における数値の丸め方です。詳しくは、IEEE 規格 754 を参照してください。

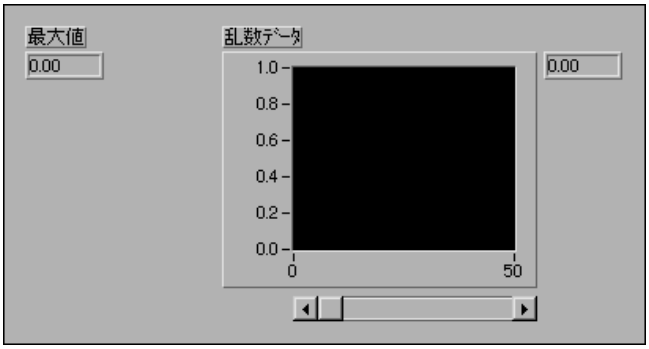


作業 3-7.　Forループを使用する

ここでは、For ループとシフトレジスタを使用して一連の乱数の最大値を求めるのが目的です。

フロントパネル

1. 新しいフロントパネルを開き、次の図のようにオブジェクトを追加します。



a. フロントパネルにデジタル表示器を配置し、[最大値]というラベルを付けます。

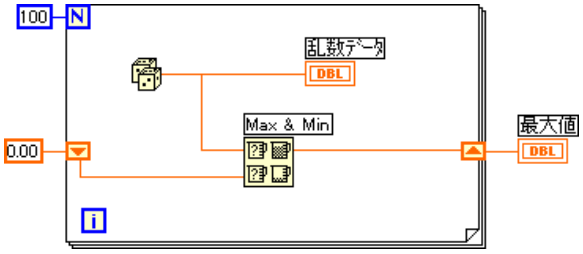
b. フロントパネルに波形チャートを配置し、[乱数データ]というラベルを付けます。チャートの目盛りの範囲を[0.0]～[1.0]に変更します。

c. チャートをポップアップして**表示→スクロールバー**と**表示→デジタル表示**を選択します。パレットと凡例をポップアップして非表示にします。

d. 位置決めツールを使用してスクロールバーのサイズを変更します。

ブロックダイアグラム

2. ブロックダイアグラムを開き、次の図のように修正します。



3. For ループ（**関数→ストラクチャ**）をブロックダイアグラム上に配置します。
4. For ループの左右いずれかの枠をポップアップ（右クリック）し、**シフトレジスタを追加する**を選択してシフトレジスタを追加します。
5. ブロックダイアグラムに以下のオブジェクトを追加します。

Random Number (0-1) 関数（**関数→数値**）— この関数は乱数データを発生します。

数値定数（**関数→数値**）— For ループでは繰り返しの回数を指定する必要があります。ここでは、For ループを 100 回実行します。

数値定数（**関数→数値**）— この練習では、乱数発生器の出力が 0.0 から 1.0 であることがわかっていますので、シフトレジスタの初期値を 0 に設定します。

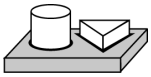
シフトレジスタを初期化するためには収集するデータについてある程度わかっている必要があります。たとえばシフトレジスタを 1.0 に初期化する場合、この値は予想されるすべてのデータ値よりも大きいので、この値が必ず最大値になります。シフトレジスタを初期化しなかった場合、シフトレジスタには VI を前に実行したときの最大値が含まれています。このため、今回収集したデータセットとは無関係な値が最大値として出力される恐れがあります。

Max & Min 関数（**関数→比較**）— 2 つの数値入力を取り込んで、2 つの値のうちの最大値を右上隅に、最小値を右下隅に出力します。この練習問題で取り上げるのは最大値だけですので、最大値出力だけを配線し、最小値出力は無視します。

6. 図のように端子を配線します。最大値端子がForループの内側にある場合は、この端子が連続的に更新されていくのがわかりますが、この場合はループの外側にありますので、この端子には最終的に計算された最大値だけが格納されます。

☞ **注**　**ループの繰り返しのたびに表示器を更新すると時間がかかりますので、実行速度を上げるためには、できるだけこうしたやり方は避けてください。**

7. VIを実行します。
8. このVIを、`Culculate Max.vi`という名前で`LabVIEW¥Activity`ディレクトリに保存します。



これで作業 3-7 は完了です。

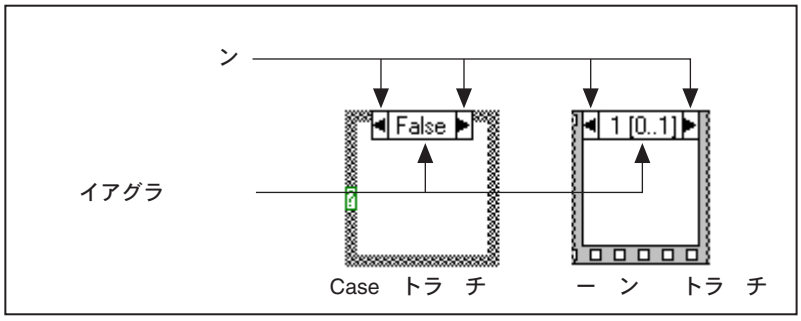
4

Caseストラクチャ、シーケンスストラクチャ、およびフォーミュラノード

この章では、Caseストラクチャ、シーケンスストラクチャ、およびフォーミュラノードの基本概念を紹介し、作業を行いながら以下の内容を説明します。

- Caseストラクチャの使用方法
- シーケンスストラクチャの使用方法
- シーケンスローカルとその使用方法
- フォーミュラノードとその使用方法

Caseストラクチャとシーケンスストラクチャはともに複数のサブダイアグラムを含むことができます。いずれも一度に表示できるのはこのうちの一つだけです。各ストラクチャ枠の上部には**サブダイアグラム表示ウィンドウ**があり、その中央には**ダイアグラム識別子**が、左右には増減用のボタンがあります。ダイアグラム識別子は、現在表示されているダイアグラムを示します。Caseストラクチャの場合、ダイアグラム識別子はサブダイアグラムを選択する値のリストです。シーケンスストラクチャの場合、ダイアグラム識別子は0～*n*–1という順序のフレーム番号です。次の図に、Caseストラクチャとシーケンスストラクチャを示します。

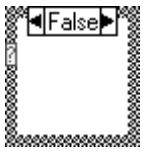


減分ボタン（左）または増分ボタン（右）をクリックすると、それぞれ、前または次のサブダイアグラムが表示されます。最後のサブダイアグラムで増分ボタンを押すと最初のサブダイアグラムが表示され、最初のサブダイアグラムから減分ボタンを押すと最後のサブダイアグラムが表示されます。Caseストラクチャとシーケンスストラクチャに関する詳細は、『G プ

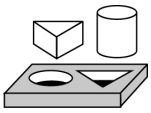
ログラミングリファレンスマニュアル』の「第19章　ストラクチャ」を参照してください。

Caseストラクチャ

Caseストラクチャには複数のサブダイアグラム、つまり**ケース**があり、ストラクチャが実行されるといずれか1つだけが実行されます。どれが実行されるかは、整数、ブール、文字列などの値、または**選択子**と呼ばれる選択端子の外側に配線したenum値によって決まります。Caseストラクチャを次の図に示します。



☞ **注**　他のプログラミング言語のCase文では、通常、ケースが範囲外である場合はどのケースも実行されません。Gでは、範囲外の値に対応できるデフォルトケースを作るか、または考えられる入力値のリストを作成しなければなりません。

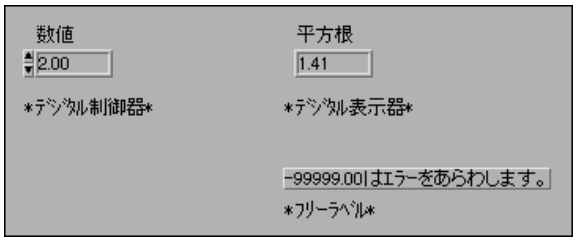


作業 4-1.　Caseストラクチャを使用する

ここでは、数値が正かどうかをチェックするVIを作成することが目的です。数値が正である場合、VIはその数値の平方根を計算し、正でない場合、VIはエラーを返します。

フロントパネル

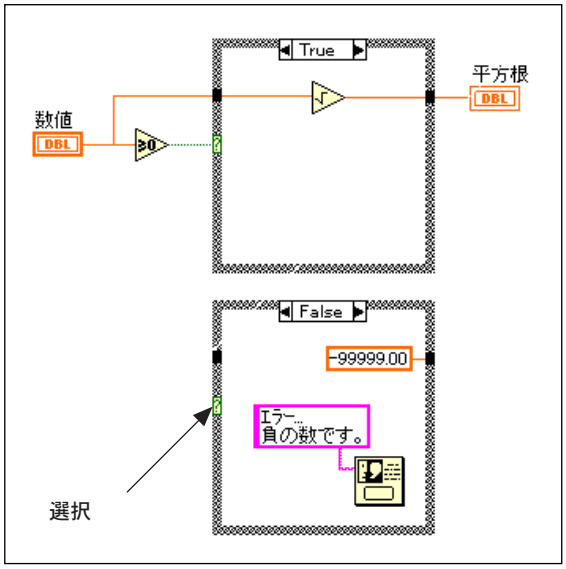
1. 新しいフロントパネルを開き、次の図のようなオブジェクトを作成します。



数値制御器は[数値]を入力します。[平方根]表示器は数値の平方根を表示します。フリーラベルはユーザのメモとして使用します。

ブロックダイアグラム

2. 次の図のようなダイアグラムを作成します。



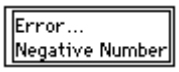
3. **関数→ストラクチャ**からCaseストラクチャを選択してブロックダイアグラム内に配置します。Caseストラクチャはサイズ変更可能なボックスで、直接ダイアグラムにドロップされることはありませんが、位置やサイズを変更することができます。そのためには、Caseストラクチャに入れたいすべての端子より上でかつ左側の部分をクリックします。マウスボタンを押したまま端子を囲む四角形をドラッグします。

Greater Or Equal To 0?関数（**関数→比較**）— 数値入力が0以上である場合にTRUEを返します。

Square Root関数（**関数→数値**）— 入力値の平方根を返します。

数値定数（**関数→数値**）— この作業では、定数はエラーの数値を示します。

One Button Dialog関数（**関数→時間＆ダイアログ**）— この作業で、この関数はエラー... 負の数値というメッセージを含むダイアログボックスを表示します。



文字列定数（**関数→文字列**）— ラベリングツールを使用してボックスにテキストを入力します。

VIは、TRUEのケースまたはFALSEのケースのいずれかを実行します。数値が0以上の場合VIはTRUEケースを実行して数値の平方根を返します。FALSEケースは–99999.00を出力し、[エラー ... 負の数です]というメッセージを含むダイアログボックスを表示します。

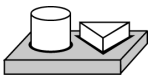
☞ **注**　**各ケースごとに出力トンネルを定義する必要があります。1 つのケース内に出力トンネルを作成すると、他のすべてのケースの同じ位置にトンネルが表示されます。配線されていないトンネルは白い四角形で表示されます。**

4. フロントパネルに戻ってVIを実行します。数値というラベルを付けたデジタル制御器の値を変更して、0以上の数や0より小さい数を試してください。デジタル制御器を負の数に変更すると、ユーザがCaseストラクチャのFALSEケース内に設定したエラーメッセージがLabVIEWで表示されることを確認してください。
5. このVIを、`Square Root.vi`という名前で`LabVIEW¥Activity`ディレクトリに保存します。

VIロジック

この作業のブロックダイアグラムは、テキストベースの言語における以下の疑似コードと同じ機能を果たします。

```
if (Number >= 0) then
Square Root Value = SQRT(Number)
else
Square Root Value = -99999.00
Display Message "Error...Negative Number"
end if
```

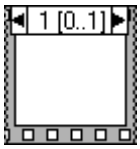


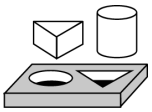
これで作業 4-1 は完了です。

シーケンスストラクチャ

フィルムのフレームのような外観のシーケンスストラクチャは、ブロックダイアグラムを順番に実行します。従来のプログラミング言語では、プログラム文が出てくる順に実行されます。データフロープログラミングでは、すべてのノード入力でデータが使用可能な状態になったときにノードが実行されます。ただし場合によっては、特定のノードを他のノードよりも先に実行しなければならないこともあります。G では、ノードを実行する順序を制御するためにシーケンスストラクチャを使用します。G は、最初にフレーム 0 の枠内のダイアグラムを、2 番目にフレーム 1 の枠内のダイアグラムを、という順序で実行していきます。Case ストラクチャの場合と同様、同時に表示できるフレームは 1 つだけです。

シーケンスストラクチャを、次の図に示します。



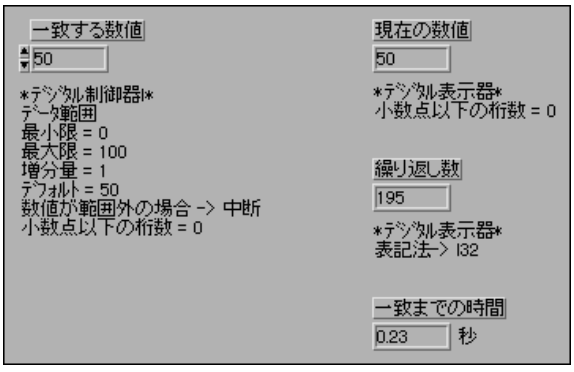


作業 4-2.　シーケンスストラクチャを使用する

ここでは、与えられた数と一致する乱数を発生するまでの時間を求める VI を作成することが目的です。

フロントパネル

1. 新しいフロントパネルを開き、次の図のようなフロントパネルを作成します。図の後の説明に従って、制御器と表示器を修正してください。



一致する（LVB）制御器には、一致させる数値が含まれています。**現在の数値（LVB）**表示器には、現在の乱数が表示されます。**繰り返し数（LVB）**表示器には、一致するまでの繰り返し数が表示されます。**時間（LVB）**表示器には、一致する数が見つかるまでの秒数が表示されます。

数のフォーマットを修正する

デフォルトの場合、LabVIEWは、数値制御器内の値を小数点以下2桁の10進数で表示します（例：3.14）。制御器または表示器のポップアップメニューの**形式と精度...**オプションを使用すると、精度を変更したり、数値制御器と表示器を科学的表記法や工学的表記法で表示することができます。また、**形式と精度...**オプションを使用して、数値や時間と日付のフォーマットで表示することもできます。

2. 一致までの時間（LVB）デジタル表示器をポップアップして**形式と精度...**を選択します。フロントパネルがアクティブウィンドウになっていないと、メニューにアクセスできません。
3. 小数点下の桁数に2を入力してOKをクリックします。

4. 一致する数値（LVB）デジタル制御器をポップアップして**表記法→I32**を選択します。
5. 現在の数値（LVB）デジタル表示器と繰り返し数（LVB）デジタル表示器に対してステップ4を繰り返します。

データ範囲を設定する

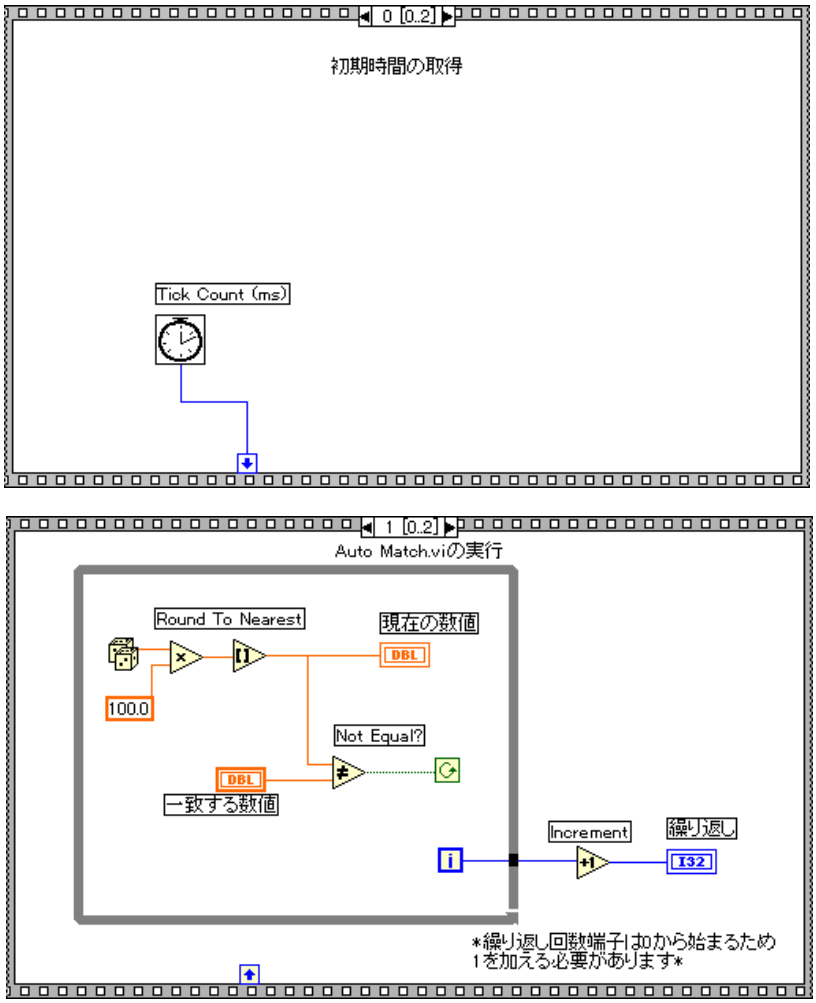
データ範囲...オプションを制御器や表示器に使用すると、あらかじめ決められた範囲や増分を超える値を設定することができます。オプションとしては、値を無視する、値を強制的に範囲内に入れる、実行を中断する、があります。範囲エラーにより実行が中断されたときは、ツールバーの実行ボタンの代わりに範囲エラーの記号が表示されます。また範囲外の制御器は、太い赤の枠で囲まれます。

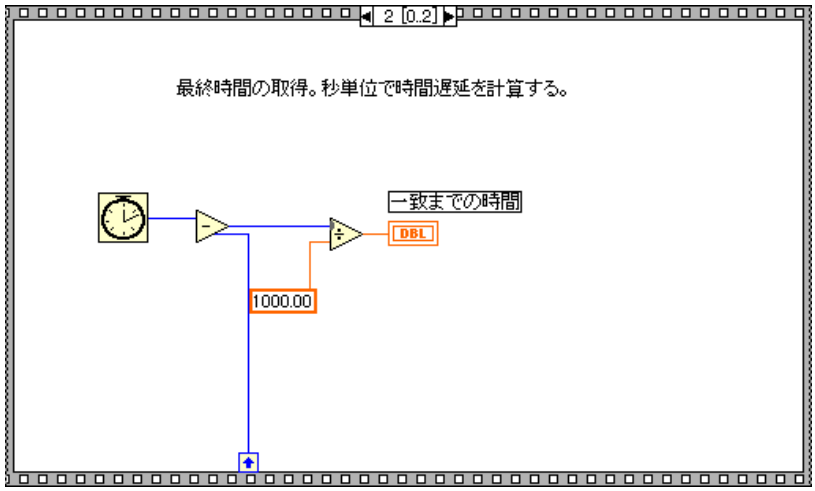
6. 一致する数値（LVB）表示器をポップアップして**データ範囲...**を選択します。
7. 次の図のようにダイアログボックスに入力してOKをクリックします。

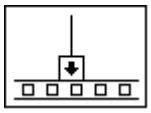
ブロックダイアグラム

8. ブロックダイアグラムを開きます。
9. ブロックダイアグラム内にシーケンスストラクチャ（**関数→ストラクチャ**）を配置します。
10. サイズ変更カーソルでいずれかの角をドラッグして、ストラクチャを拡大します。

11. 枠をポップアップして**後にフレームを追加**を選択し、新しいフレームを作成します。同じ操作を繰り返してフレーム2を作成します。
12. 次の図のようなブロックダイアグラムを作成します。







前の図のフレーム 0 には矢印が表示された小さいボックスがあります。このボックスはシーケンスストラクチャのフレーム間でデータを渡すための**シーケンスローカル**変数です。シーケンスローカルは、フレームの辺の上に作成することができます。こうすると、フレームシーケンスローカルに配線されたデータを、以後のフレームで使用できます。ただし、シーケンスローカルを作成したフレームより前のフレーム内のデータにはアクセスできません。

13. フレーム 0 の下の枠をポップアップして**シーケンスローカルを追加**を選択し、シーケンスローカルを作成します。

シーケンスローカルは空の四角形として表示されます。四角形の内側の矢印は、シーケンスローカルに関数を配線すると自動的に表示されます。

14. この作業の「ブロックダイアグラム」の項の最初の図に示されているブロックダイアグラムを終了します。

Tick Count (ms) 関数（**関数→時間＆ダイアログ**） — 電源を入れてからの経過時間をミリ秒単位で返します。この作業では、Tick Count 関数が 2 つ必要です。

Random Number (0-1)関数（**関数→数値**） — 0～1の範囲の乱数を返します。

Multiply 関数（**関数→数値**） — この作業で、この関数は乱数に 100 をかけます。

数値定数関数（**関数→数値**） — この作業で、数値定数は乗算可能な最大値を示します。

Round to Nearest関数（**関数→数値**）— この作業で、この関数は0～100の間の乱数を最も近い整数に丸めます。

Not Equal?関数（**関数→比較**）— この作業で、この関数は乱数をフロントパネルで指定された数と比較し、両者が等しくない場合はTRUEを返し、そうでない場合はFALSEを返します。

Increment関数（**関数→数値**）— この作業で、この関数はWhileループのカウントを1ずつ増分します。

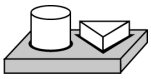
Subtract関数（**関数→数値**）— この作業で、この関数はフレーム2とフレーム0との間の（ミリ秒単位の）経過時間を返します。

Divide関数（**関数→数値**）— この作業で、この関数はミリ秒単位の経過時間数を1,000で割って秒単位の数値に変換します。

数値定数（**関数→数値**）— この作業で、この関数は数値をミリ秒から秒に変換します。

フレーム0で、Tick Count (ms)関数はミリ秒単位の現在時刻を返します。この値はシーケンスローカルに配線され、後に続くフレームでこの値を使用できます。フレーム1では、指定された数がRandom Number (0-1)関数の返す数と一致するまで、VIはWhileループを実行します。フレーム2で、Tick Count (ms)関数は新しい時刻をミリ秒単位で返します。VIは（フレーム0からシーケンスローカル経由で渡された）古い時刻を新しい時刻から引いて、経過時間を求めます。

15. フロントパネルに戻り、[一致する数値（LVB）] 制御器に数値を入力してVIを実行します。
16. このVIを、`Time to Match.vi`という名前で`LabVIEW¥Activity`ディレクトリに保存します。



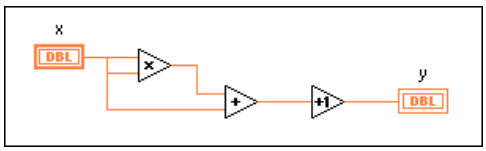
これで作業 4-2 は完了です。

フォーミュラノード

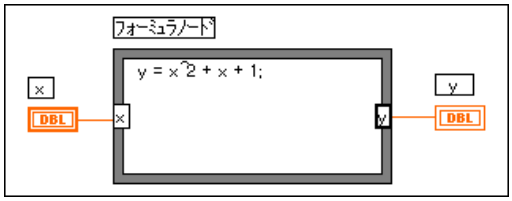
フォーミュラノードはサイズ変更可能なボックスで、これを使用するとブロックダイアグラムに直接フォーミュラを入力できます。フォーミュラノードをブロックダイアグラム上に配置するには、**関数→ストラクチャ**からフォーミュラノードを選択します。式に変数が多くある場合や式が複雑な場合は、この機能が役立ちます。たとえば次の式を考えてみます。

$y = x^2 + x + 1$

G の通常の算術関数を使用してこの式を実行する場合、ブロックダイアグラムは次の図のようになります。

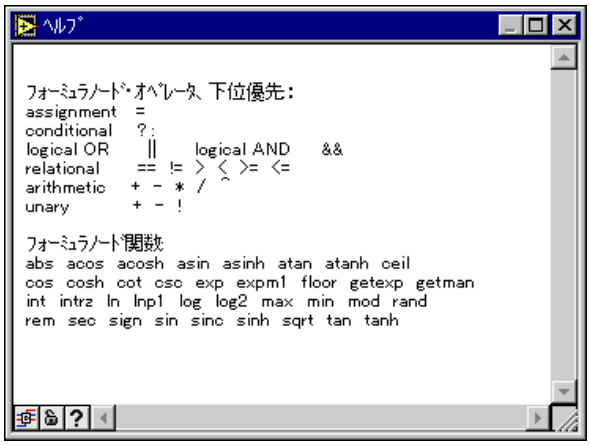


フォーミュラノードを使用すると、同じ式を次の図のように実行することができます。



フォーミュラノードを使用すると、ブロックダイアグラムを細分化せずに、複雑なフォーミュラや複数のフォーミュラを直接入力することができます。フォーミュラの入力にはラベリングツールを使用します。フォーミュラノードの入力端子や出力端子を作成するには、ノードの枠をポップアップして入力端子を追加（出力端子を追加）を選択し、ボックスに変数名を入力します。変数の大文字と小文字は区別されます。枠内に 1 つまたは複数のフォーミュラを入力し、各フォーミュラ文の最後には必ずセミコロン（;）を付けます。

フォーミュラノード内で使用可能な演算子と関数は、次の図のようにフォーミュラノードのヘルプウィンドウに一覧表示されます。各フォーミュラ文の終わりにはセミコロンが付いています。

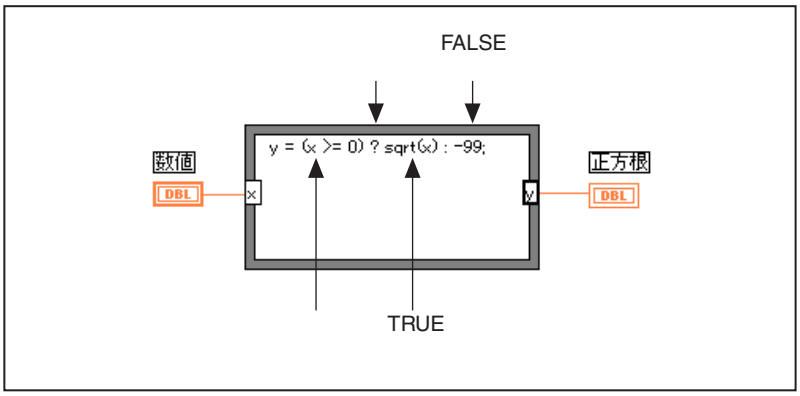


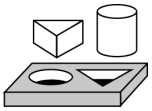
次の例は、フォーミュラノード内で条件付きの代入処理を実行する方法を示します。

次のようにxが正の場合にxの平方根を計算して結果をyに代入するコード部分を考えてみます。xが負の場合このコードはyに–99を代入します。

```
if (x >= 0) then
y = sqrt(x)
else
y = -99
end if
```

次の図のように、フォーミュラノードを使用すると、このコード部分を実行することができます。





作業 4-3.　フォーミュラノードを使用する

ここでは、フォーミュラノードを使用して次の式を計算するVIを作成することが目的です。

$y^1 = x^3 - x^2 + 5$

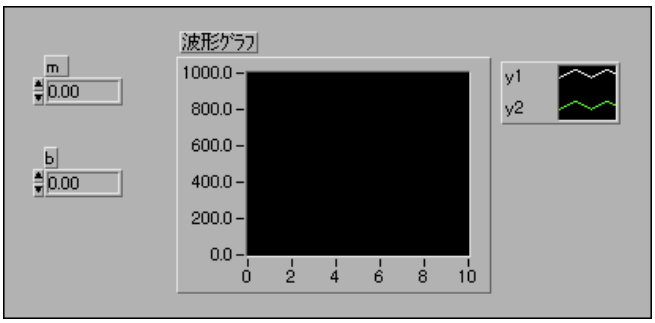
$y^2 = m * x + b$

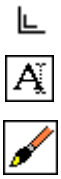
ただし、xは0から10の範囲で変化します。

両方の式に対してフォーミュラノードを1つだけ使用し、結果を同じグラフ上にグラフ表示します。グラフに関する詳細は、「第5章　配列、クラスタ、およびグラフ」を参照してください。

フロントパネル

1. 新しいフロントパネルを開き、次の図のようなフロントパネルを作成します。波形グラフ表示器には、式のプロットが表示されます。VI は2つのデジタル制御器を使用して[m]と[b]の値を入力します。

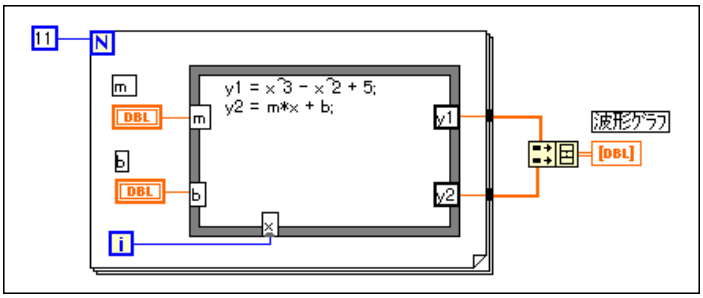


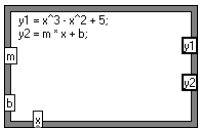


2. **表示→凡例**を選択して、次の図のようなグラフの凡例を作成します。サイズ変更カーソルを使用し、2つのプロットが表示されるように凡例を下にドラッグします。ラベリングツールを使用してプロットの名前を変更します。凡例をポップアップすると、各プロットの線のスタイルを定義することができます。また、プロットの凡例に対してカラーツールを使用すると、各プロットに色を付けることができます。

ブロックダイアグラム

3. 次の図のようなブロックダイアグラムを作成します。





フォーミュラノード（**関数→ストラクチャ**）。このノードを使用すると、フォーミュラを直接入力することができます。3つの入力端子を作成するには、枠をポップアップして**入力端子を追加**を選択します。出力端子を作成するにはポップアップメニューから**出力端子を追加**を選択します。

入力端子や出力端子を作成する場合は、変数名を指定する必要があります。この変数名は、フォーミュラに使用するものと正確に一致しなければなりません。名前の大文字と小文字は区別されます。したがって、端子名に小文字の[a]を使用する場合は、フォーミュラでも小文字の[a]を使用しなければなりません。変数名とフォーミュラの入力には、ラベリングツールを使用します。

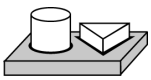
☞ **注**　**変数名の長さに制限はありませんが、名前が長いとダイアグラム内で大きな領域が必要になることに注意してください。フォーミュラ文の最後にはセミコロン（;）を付けます。**

数値定数（**関数→数値**）。カウント端子をポップアップして**定数を作成**を選択し、自動的に数値定数を作成して配線することもできます。数値定数は、Forループの繰り返し回数を指定します。xの範囲が10を含めて0～10である場合は、11をカウント端子に配線する必要があります。

繰り返し端子は0から10までカウントしますので、これを使用してフォーミュラノードでのxの値を制御します。

Build Array（**関数→配列**）は、2つの配列入力をマルチプロットグラフのフォームに入れます。サイズ変更カーソルでいずれかの角をドラッグして2つの入力端子を作成します。配列に関する詳細は、「第5章　配列、クラスタ、およびグラフ」を参照してください。

4. フロントパネルに戻り、[m]と[b]に異なる値を設定してVIを実行します。
5. このVIを、`Equations.vi`という名前で`LabVIEW¥Activity`ディレクトリに保存します。



これで作業 4-3 は完了です。

人工データ依存

ワイヤで接続されていないノードは、任意の順序で実行できます。ノードは、必ずしも左から右に、上から下に実行する必要はありません。シーケンスストラクチャは、自然なデータ依存が存在しない場合に実行順序を制御するひとつの方法です。

実行順序を制御するもう一つの方法としては、値ではなくデータの到着によりオブジェクトの実行をトリガする条件である人工的なデータ依存を作成する方法があります。受ける側は実際に内部でデータを使用しなくても構いません。場合によってはノード間の人工的なリンクのために混乱が生ずるという欠点もありますが、ノードすべてを同一のレベルで表示できることが人工的依存のメリットです。

Examples¥General¥structs.llbのTiming Template (datadep).viを開くと、シーケンスストラクチャでなく人工的データ依存を使用するために、Timing Templateがどのように変更されているかを確認することができます。

5

配列、クラスタ、およびグラフ

この章では、多形性、配列、クラスタ、およびグラフの基本概念を紹介し、作業を行いながら自動指標付け、グラフVI、解析VIについて説明します。

配列

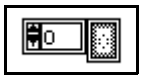
配列とは、すべてが同じタイプのデータ要素の集合体です。配列には1次元のものや2次元以上のものがあり、メモリが許せば、1つの次元ごとの最大要素数は2^{31}–1個です。配列の各要素にアクセスするには指標を使用します。指標の範囲は0から*n*–1で、*n*は配列に含まれる要素数です。次に示す1D数値配列はこのストラクチャを示します。最初の要素の指標は0、2番目の要素の指標は1、のようになっていることに注意してください。

10 配

0	1	2	3	4	5	6	7	8	9
1.2	3.2	8.2	8.0	4.8	5.1	6.0	1.0	2.5	1.7

配列の作成方法と初期化方法

ブロックダイアグラム内のデータソースとして配列が必要な場合は、**関数→配列**を選択し、配列シェルを選択してブロックダイアグラム上に配置することができます。操作ツールを使用すると、数値定数、ブール定数、または文字列定数を選択して空の配列内に配置できます。次の図は、数値定数が挿入された配列シェルの例を示します。

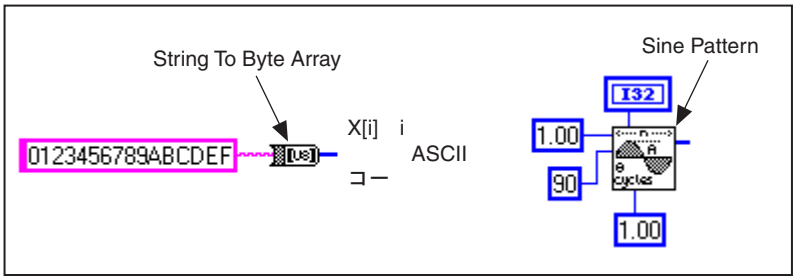


フロントパネル上で配列を作成するには、**制御器**パレットから**配列&クラスタ**を選択してフロントパネル上に配列シェルを配置します。さらにオブジェクト（数値など）を選択して配列シェル内に配置します。これにより数値配列が生成されます。

☞ **注**　**また、フロントパネル上で配列と配列に対応する制御器を作成し、その配列をブロックダイアグラムにコピーまたはドラッグして対応する定数を作成することもできます。**

フロントパネルで配列の制御器と表示器を作成する方法に関する詳細は、『G プログラミングリファレンスマニュアル』の「第 14 章 配列とクラスタの制御器および表示器」を参照してください。

ブロックダイアグラム上で配列を作成したり初期化する方法はいくつかあります。次の図のように、ブロックダイアグラム関数の中にも配列を生成するものがあります。



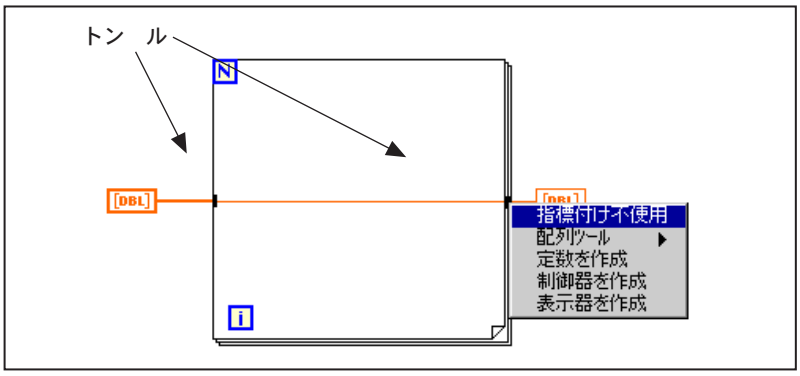
配列制御器、定数、および表示器

フロントパネルやブロックダイアグラム上で配列制御器、定数、表示器を作成するには、配列シェルを、数値、ブール値、文字列、またはクラスタと組み合わせます。別の配列、チャート、またはグラフを配列要素とすることはできません。配列の例は、`Examples¥General¥arrays.llb` を参照してください。

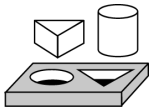
自動指標付け

For ループおよび While ループストラクチャは、境界部で自動的に配列の要素を指標付けして配置することができます。これらの機能をまとめて**自動指標付け**と呼びます。自動指標付けを ON にして外部のノードからループ境界部の入力トンネルに任意の次数の配列を配線すると、その配列の構成要素は最初の要素から一度に 1 つずつループに入ります。ループは 1D 配列のスカラ要素、2D 配列の 1D 配列などに指標を付けます。出力トンネルでは逆の操作が行われ、要素は順番に 1D 配列内に、また 1D 配列は 2D 配列内にという順に配置されます。

☞ **注**　**For ループに配線された配列はどれも、デフォルトで自動指標付けが行われます。自動指標付けを無効にするには、トンネル（入力配列の入力ポイント）をポップアップして指標付け不使用を選択します。**



While ループに配線された配列は、デフォルトでは自動指標付けが行われません。自動指標付けを ON にするには、While ループの配列トンネルをポップアップします。



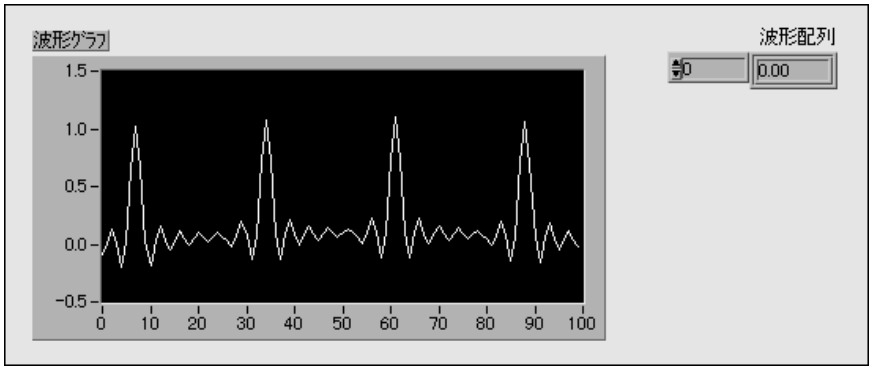
作業 5-1.　自動指標付けにより配列を作成する

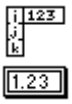
ここでは、For ループの自動指標付け機能を使用して配列を作成し、波形グラフに配列をプロットすることが目的です。

Generate Waveform VI を使用して配列を生成する VI を作成し、波形グラフに配列をプロットします。また、VI を修正して複数のプロットをグラフ化します。

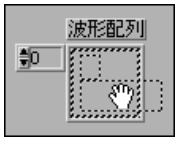
フロントパネル

1. 新しいフロントパネルを開きます。





2. フロントパネルの**制御器→配列＆クラスタ**から配列シェルを配置し、この配列シェルに[波形配列]というラベルを付けます。
3. 次の図のように、**制御器→数値**からのデジタル表示器を配列シェルの要素表示の中に配置します。この表示器には配列の内容が表示されます。



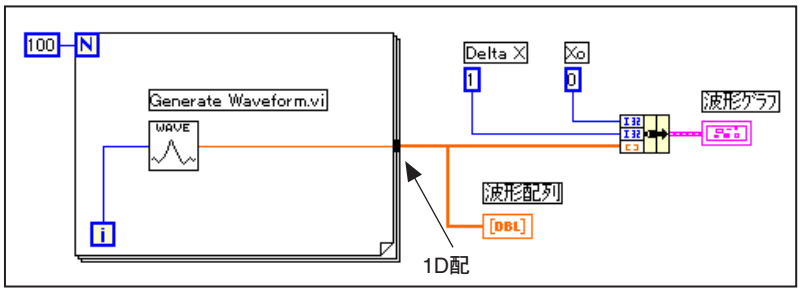
4. **制御器→グラフ**からの波形グラフをフロントパネル内に配置し、このグラフに[波形グラフ]というラベルを付けます。

5. サイズ変更カーソルで角をドラッグしてグラフを拡大します。
6. 凡例とパレットを非表示にします。
7. グラフをポップアップして**Ｙ軸→Ｙ軸オートスケール**を選択解除することにより、オートスケーリングをOFFにします。

8. テキストツールを使用してY軸の目盛りの範囲を[`-0.5`]～[`1.5`]に変更します。

ブロックダイアグラム

9. 次の図のようなブロックダイアグラムを作成します。



Generate Waveform VI（**関数→VIを選択**の後`LabVIEW¥Activity`ディレクトリから選択）－波形の１つの点を返します。このVIにはスカラインデックスの入力が必要なため、ループ繰り返し端子をこの入力に配線します。

Generate Waveform VIはループの境界部では配列に変わるため、ワイヤが太くなることに注意してください。

For ループは、境界部で自動的に配列内に要素を配置します。これが、自動指標付けと呼ばれるものです。この場合、ループカウント数値入力に配線された数値定数にはFor ループがあり、100 個の要素（指標は 0 ～ 99）を含む配列を生成します。

Bindle関数（**関数→クラスタ**）— プロット要素を集めてクラスタにします。正しく配線するには、Bundle 関数アイコンのサイズを変更する必要があります。アイコンの左下の角に位置決めツールを移動すると、ツールが左の図のようなサイズ変更カーソルに変わります。ツールが変わったらクリックして、3 番目の入力端子が表示されるまで下にドラッグします。今度は、前の図に示す通り、続けてブロックダイアグラムを配線することができます。

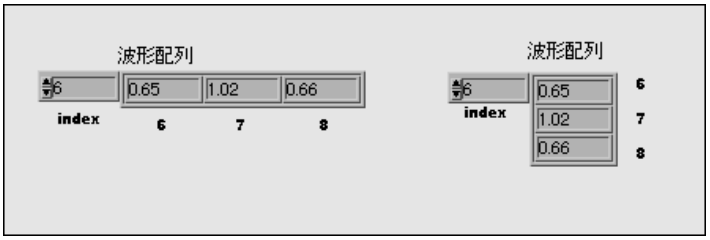
数値定数（**関数→数値**）— 3 つの数値定数は、For ループの繰り返し回数、X の初期値、X の増分値を設定します。左の図のような For ループのカウント端子をポップアップして定数を作成を選択すると、その端子に自動的に数値定数を追加して配線できることに注意してください。

10. フロントパネルから VI を実行します。VI は、自動指標付けされた波形配列を波形グラフにプロットします。*X* の初期値は 0、*X* の増分値は 1 です。
11. *X* の増分値を 0.5 に、*X* の初期値を 20 に変更してもう一度 VI を実行します。

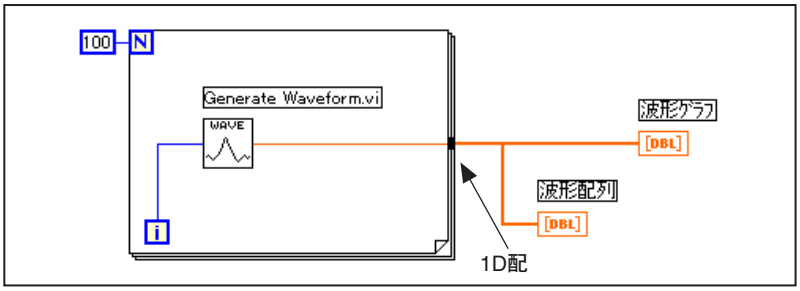
同じ 100 個のデータ点のぞれぞれが、今度は、初期値 20、増分値 0.5 でグラフに表示されることを確認してください（X 軸に注目）。時間計測テストでは、このグラフは 20 秒から始まる 50 秒分のデータに対応します。

12. 指標表示に要素の指標を入力することで、波形配列の任意の要素を表示することができます。配列サイズより大きい数を入力した場合は淡色表示になり、その指標に対して定義された要素がないことを示します。

同時に複数の要素を表示する場合は、配列表示器のサイズを変更することができます。配列の右下の角に位置決めツールを移動すると、ツールが左図のような配列サイズ変更カーソルに変わります。ツールが変わったら、右または真下にドラッグします。今度は、次の図のように、指定した指標に対応する要素から始まる複数の要素が、指標が昇順になるように配列内に表示されます。



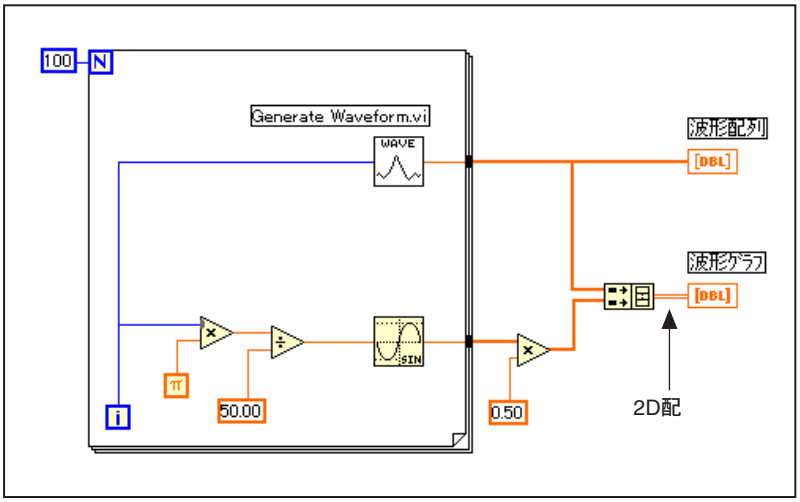
前のブロックダイアグラムでは、波形の*X*の初期値と*X*の増分値を指定しました。デフォルトの場合、*X*の初期値は0、*X*の増分値は1です。したがって、次の図のように、*X*の初期値と*X*の増分値が指定されていなくても、波形配列を波形グラフ端子に直接配線することができます。



13. ブロックダイアグラムに戻ります。Bundle 関数およびこれに配線された数値定数を削除します。関数および定数を削除するには、位置決めツールで関数と定数を選択して<Delete>を押します。さらに**編集→不良ワイヤの削除**を選択します。前の図のようにブロックダイアグラムの配線を完成します。
14. VIを実行します。*X*の初期値0、*X*の増分値1で波形がプロットされることを確認してください。

マルチプロットグラフ

通常はシングルプロットグラフに渡されるデータタイプを配列にすると、マルチプロット波形グラフを作成することができます。



15. 作業を進め、上の図のようなブロックダイアグラムを作成します。

Sine 関数（**関数→数値→三角関数**）— この作業では、この関数を For ループ内で使用してサイン波の 1 サイクルを示す点の配列を作成します。

Build Array関数（**関数→配列**）— この練習問題では、この関数を使用して、1 つの波形グラフ上に 2 つの配列をプロットするのに適したデータストラクチャを作成します。この場合は2D配列となります。2 つの入力を作成するには、位置決めツールで Build Array 関数の角をドラッグして拡大します。

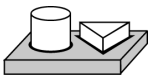
Pi 定数（**関数→数値→数値定数を追加**）— Multiply 関数と Divide 関数は**関数→数値**の中にあることを思い出してください。

16. フロントパネルに切り替えて VI を実行します。

2 つの波形が同じ波形グラフ上にプロットされることを確認してください。どちらのデータセットに対しても、デフォルトの *X* の初期値は 0、*X* の増分値は 1 となっています。

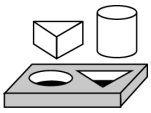
注　グラフ上のプロットの外観を変更するには、特定のプロットの凡例をポップアップします。たとえば、折れ線グラフから棒グラフに変えるには**一般プロット→棒グラフ**を選択します。

17. このVIを、`Graph Waveform Arrays.vi`という名前で`LabVIEW¥Activity`ディレクトリに保存します。



これで作業 5-1. は完了です。

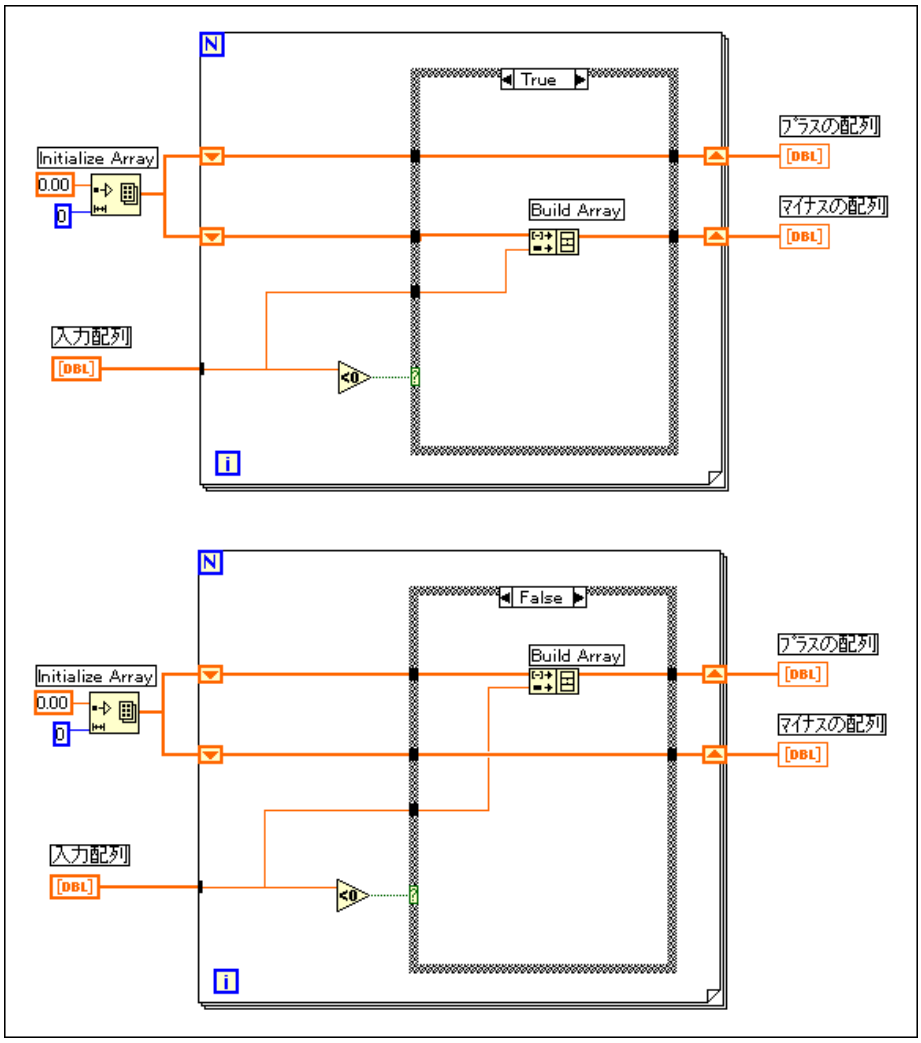
前の例でForループは100回実行されましたが、これは、定数[`100`]がカウント端子に配線されているからです。次の作業では、ループの実行回数を指定する別の方法を示します。



作業 5-2.　入力配列に対して自動指標付けを使用する

ここでは、Forループで自動指標付けを使用して配列を処理するVIを開いて実行することが目的です。

1. **ファイル→開く ...**を選択してSeparate Array Values VIを開きます。このVIは、`Examples¥General¥arrays.llb`に入っています。
2. ブロックダイアグラムを開きます。次の図は、TRUEケースとFALSEケースの両方が表示されるブロックダイアグラムを示します。

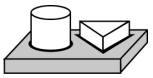


[入力配列]からのワイヤが、配列であることを示すForループの外側の太いワイヤから、単独の要素であることを示すループ内の細いワイヤに変わることを確認してください。配列の*i*番目の要素は、繰り返しの各回の間に配列から自動的に指標付けされます。

自動指標付けを使用してForループの回数を設定する

カウント端子は配線されないままになっていることに注目してください。For ループに入力される配列に対して自動指標付けを使用すると、ループは配列のサイズに従って実行されますので、値をカウント端子に配線する必要がありません。複数の配列に対して自動指標付けを使用した場合や配列を自動指標付けした上にカウントも設定した場合の実際の繰り返し回数は、可能な範囲で最も小さい値となります。

3. VIを実行します。8個の入力値のうちの4個はプラスの配列内に、4個はマイナスの配列内に表示されます。
4. ブロックダイアグラムから、定数5をForループのカウント端子に配線し、VIを実行します。入力配列にはまだ8個の要素がありますが、プラスの配列には3つ、マイナスの配列には2つの値が表示されます。これは、Nが設定され自動指標付けが行われた場合、小さい方の数が、実際のループの繰り返し回数として使用されることを示します。
5. 変更内容を保存せずにVIを閉じます。

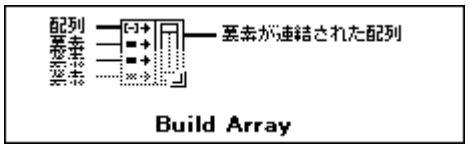


これで作業 5-2. は完了です。

配列関数を使用する

Gには配列を処理するための多くの関数が用意されており、**関数→配列**の中に入っています。このような関数としては、Replace Array Element、Search 1D Array、Sort 1D Array、Reverse 1D Array、Multiply Array Elementsがあります。使用可能な関数や配列に関する詳細は、『Gプログラミングリファレンスマニュアル』の「第14章　配列とクラスタの制御器および表示器」や、**オンラインリファレンス→関数およびVIのリファレンス**を参照してください。

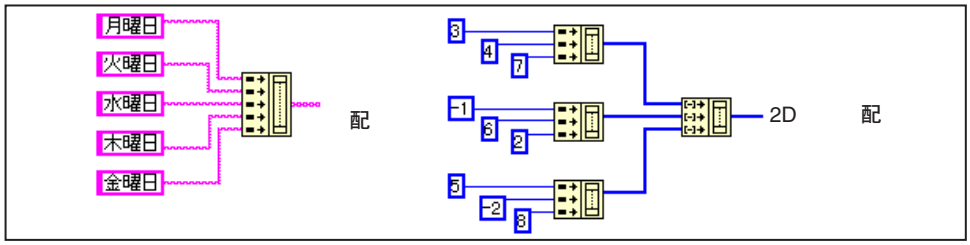
Build Array



Build Array関数（**関数→配列**）— この関数を使用すると、スカラ値や他の配列から配列を作成することができます。最初に、Build Array関数は1つのスカラ入力とともに表示されます。

Build Array関数には必要な数だけ関数を追加することができ、各入力はスカラまたは配列のいずれでも構いません。さらに多くの入力を追加するには、関数の左側をポップアップして**要素入力端子を追加**または**配列入力端子を追加**を選択します。また、サイズ変更カーソルを使用してBuild Arrayノードを拡大することもできます（位置決めツールをオブジェクトの角に移動するとサイズ変更カーソルに変わります）。入力を削除するには、サイズ変更カーソルでノードを縮小するか、**入力端子を削除**を選択します。

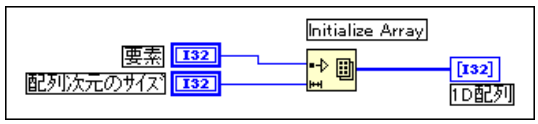
配列を作成してブロックダイアグラムの定数からの値で初期化する2つの方法を、次の図に示します。左側では、5つの文字列定数から文字列の1D配列が作成されます。右側では、数値定数の3つのグループから3つの1D数値配列が作成されます。続いて3つの配列から2Dの数値配列が作成されます。結果として、3, 4, 7; –1 ,6, 2; 5, –2, 8という行を含む3 × 3の配列ができます。



また、他の配列をスカラ要素と組み合わせて配列を作成することもできます。たとえば2つの配列と3つのスカラ要素があり、これらを組み合わせて配列1、スカラ1、スカラ2、配列2、スカラ3という順序の新しい配列を作成する場合を考えてみてください。

Initialize Array

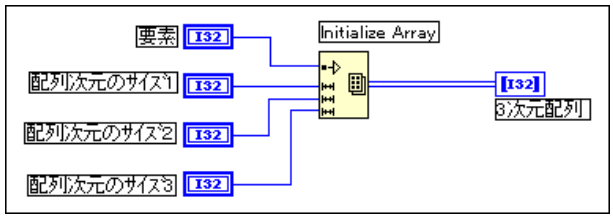
この関数を使用すると、すべての要素の値が同じであるような配列を作成することができます。次の図では、この関数により1D配列が作成されます。



要素入力によって各要素のデータタイプと値が決まり、次元サイズ入力によって配列の長さが決まります。たとえば、倍長整数 [要素] の値が 5 で、[配列次元サイズ] の値が 100 である場合、結果は、値がすべて 5 に設定された 100 個の倍長整数からなる 1D 配列となります。前の図のように、フロントパネルの制御器の端子から、ブロックダイアグラムの定数から、またはユーザのダイアグラムの他の部分の計算値から、入力を配線することができます。

次数が 2 以上の配列を作成して初期化するには、関数の左下をポップアップして**次元を追加**を選択します。また、サイズ変更カーソルを使用して Initialize Array ノードを拡大し、追加した各次元ごとに 1 つずつ、さらに次元サイズ入力を追加することもできます。次元を削除するには、関数のポップアップメニューから**次元を削除**を選択するかサイズ変更カーソルを使用してノードを縮小します。

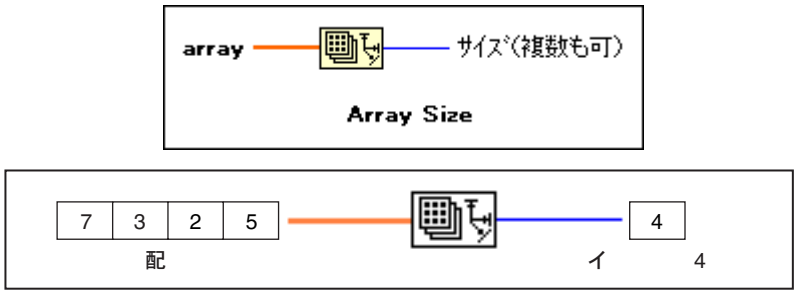
3D 配列の初期化方法を、次のブロックダイアグラムに示します。

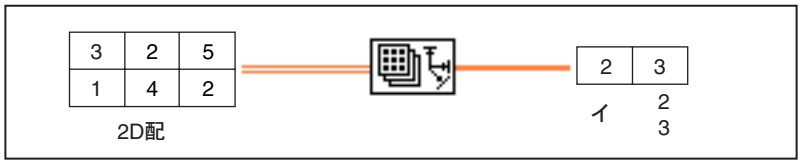


すべての次元サイズ入力が 0 である場合、関数は、指定されたタイプと次数の空の配列を作成します。

Array Size

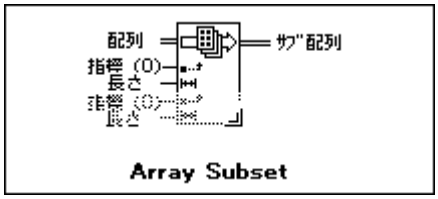
Array Size は、入力配列に含まれる要素数を返します。



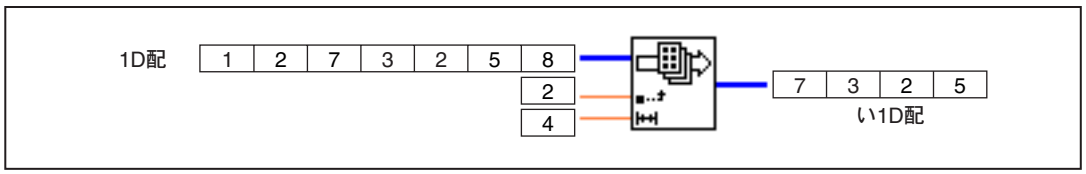


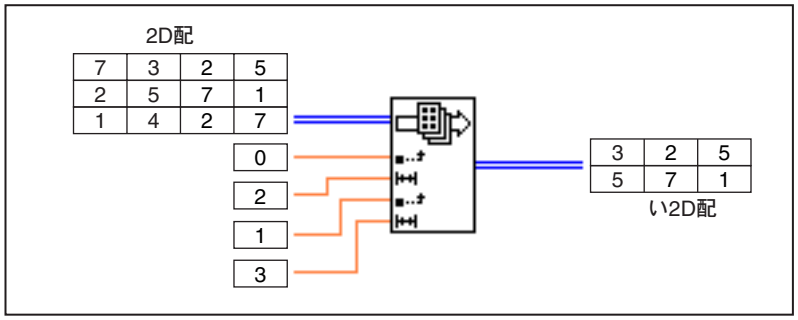
Array Subset

この関数を使用すると、配列または行列の一部分を抽出することができます。



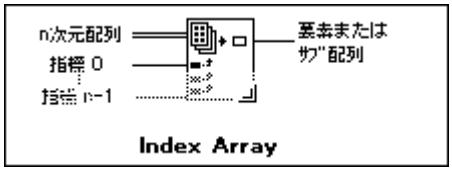
Array Subset は、指標で始まり長さの要素を含む部分配列を返します。次の図に、Array Subset の例を示します。配列の指標が 0 から始まることに注意してください。



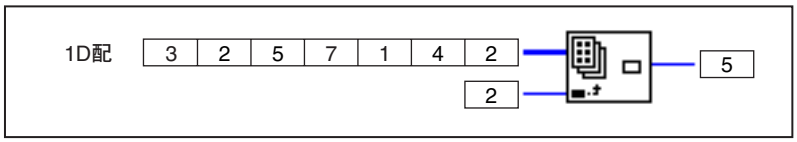


Index Array

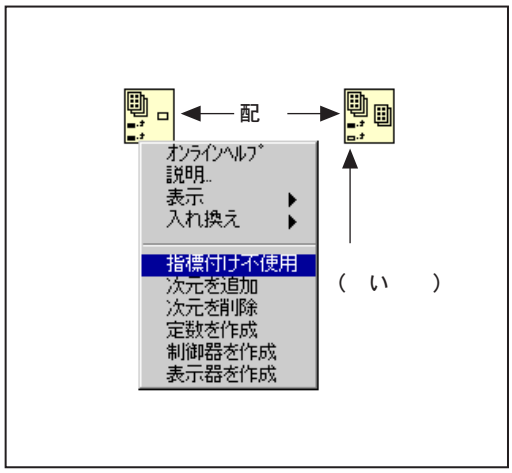
Index Array 関数は配列の要素にアクセスします。



次の図は、Index Array 関数が配列の 3 番目の要素にアクセスしている例を示します。最初の要素の指標が 0 であるため 3 番目の要素の指標は 2 になることに注意してください。

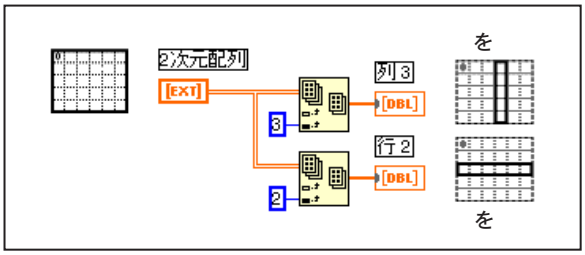


また、この関数を使用して、多次元配列の 1 つまたは複数の次元を抽出して元の配列のサブ配列を作成することもできます。このためには、次の図のように、2 つの指標入力が含まれるように Index Array 関数を拡張し、第 2 の指標端子のポップアップメニューから**指標付け不使用**コマンドを選択します。これで、配列の特定の列へのアクセスが無効になります。行の指標に対してこれを行うと、2D 配列の中の指定された行の要素を要素とする配列が得られます。また、行端子に対する指標付けを不使用にすることもできます。

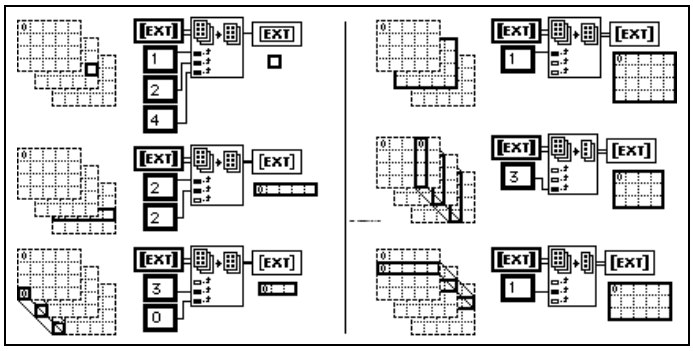


指標付けを不使用にすると、指標端子のボックスが白に変わることに注意してください。不使用にした指標を元に戻すには、同じメニューから**指標付け使用**コマンドを選択します。

サブ配列を抽出する場合には、どの次元でも組み合わせることができます。次の図は、2D配列から1Dの行または列の配列を抽出する方法を示します。



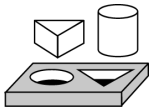
3D配列から、2つの指標端子を不使用にすると2D配列を抽出し、1つの指標端子を不使用にすると1D配列を抽出することができます。次の図は3D配列から抽出するいくつかの方法を示します。



Index Array 関数を使用して配列を抽出するときの規則を、以下に示します。

- 出力オブジェクトの次数は、不使用にした指標端子の数と等しくなければならない。たとえば、
  - 全く不使用にしない場合 ＝ スカラ要素
  - 1つを不使用にした場合 ＝ 1Dコンポーネント
  - 2つを不使用にした場合 ＝ 2Dコンポーネント
- 指標付けを使用にする端子に配線された値により出力要素を指定する必要がある。

したがって、前の左下の例は、第0列第3行のすべての要素からなる1D配列を生成するためのコマンドと解釈することができます。右上の例は、ページ1の2D配列を生成するコマンドと解釈できます。前の図に示す通り、新しい0番目の要素が、元のものに最も近くなります。

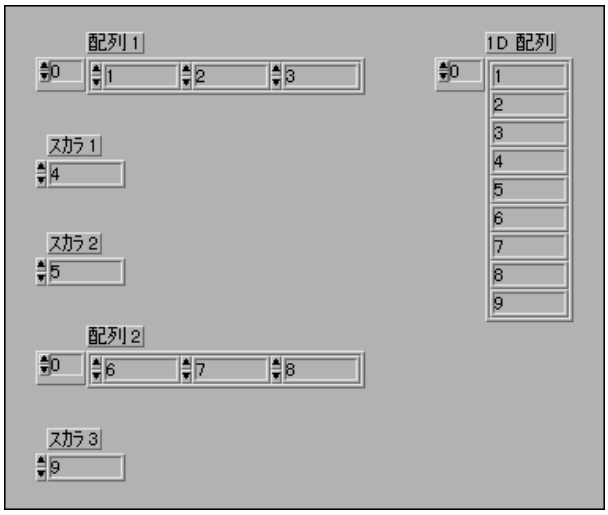


作業 5-3.　Build Array 関数を使用する

ここでは、Build Array 関数を使用して要素と配列を組み合わせて配列をさらに大きくすることが目的です。

フロントパネル

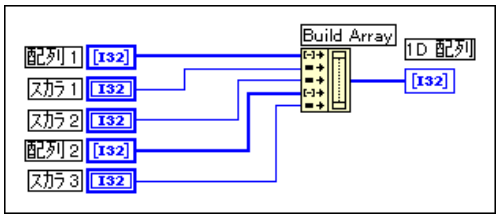
1. 次の図のように、新しいフロントパネルを作成します。



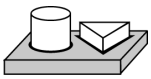
2. **制御器→数値**パレットからのデジタル制御器を配置して[スカラ1]というラベルを付けます。表示をI32に変更します。
3. これをコピーしてペーストし、さらに2つのデジタル制御器を作成し、[スカラ2]、[スカラ3]というラベルを付けます。
4. デジタル制御器の配列を作成し、[配列1]というラベルを付けます。これをコピーしてペーストし、[配列2]というラベルを付けます。
5. 上図のように配列を拡大し、[配列1]、[スカラ1]、[スカラ2]、[配列2]、[スカラ3]に1～9の値を入力します。
6. 配列をコピーしてペーストし、表示器に変更して[1D配列]というラベルを付けます。この配列を拡大して9つの値を表示します。

ブロックダイアグラム

7. Build Array 関数（**関数→配列**）をブロックダイアグラム上に配置します。5 つの入力が含まれるように、位置決めツールで Build Array 関数を拡大します。
8. Build Array ノードの最初の入力をポップアップして**配列に変更**を選択します。4 番目の入力に対しても同じ操作をします。
9. 配列とスカラをノードに配線します。次の図のように、出力配列は [配列 1] の要素、およびその後に続く [スカラ 1]、[スカラ 2]、[配列 2] の要素、そのあとに続く [スカラ 3] から構成されます。



10. VI を実行します。1 つの 1D 配列内に [スカラ 1]、[スカラ 2]、[スカラ 3]、[配列 1]、[配列 2] の値が表示されるのがわかります。
11. この VI を、`Build Array.vi` という名前で `LabVIEW¥Activity` ディレクトリに保存します。



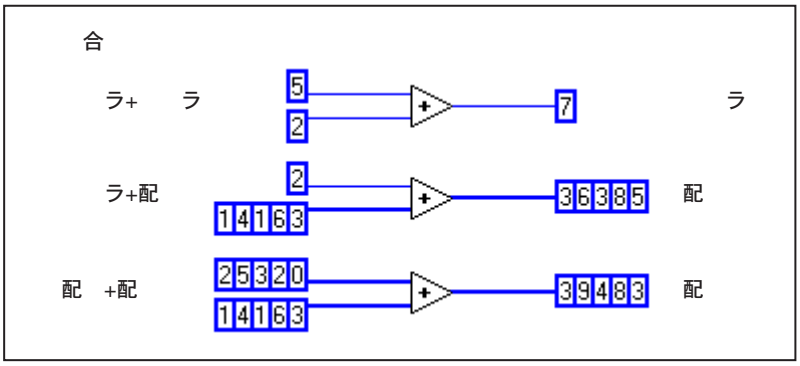
これで作業 5-3. は完了です。

メモリの効率的な使用：データのコピーを最小限に抑える

メモリを節約するため、倍精度配列の代わりに単精度配列を使用することができます。メモリの割り当て方法に関しては、『G プログラミングリファレンスマニュアル』の「第 28 章　性能に関する問題」の「メモリの使用状況をモニタする」の項を参照してください。

多形性とは？

多形性とは、さまざまなタイプ、次元、表現方法の入力データに対する関数の調整機能を意味します。ほとんどのG関数は多形的です。たとえば、Add関数の多形的組合せのいくつかを次の図に示します。



最初の組合せでは、2つのスカラが加算されて結果がスカラになります。第2の組合せではスカラが配列の各要素に加算されて結果が配列になります。配列はデータの集合体です。第3の組合せではある配列の各要素が他の配列の対応する要素に加算されます。また、数値のクラスタやクラスタの配列など、他の組合せを使用することもできます。

これらの原則は、他のG関数やデータタイプにも適用できます。G関数の多形性にはさまざまな程度があります。関数の中には、数値入力とブール入力を受け取るものもありますし、その他のデータタイプの組合せを受け取るものもあります。多形性に関する詳細は、**オンラインリファレンス→関数およびVIリファレンス**を参照してください。

クラスタ

クラスタは、タイプの異なるデータ要素を含むことが可能なデータタイプです。作業5-4で作成するブロックダイアグラムの中のクラスタは、ブロックダイアグラム上の複数の場所の関連データ要素をグループ化しますので、ワイヤの過密な配線が少なくなります。クラスタを使用する場合、サブVIの接続端子が少なくて済みます。クラスタは、Pascalの場合のレコードやCの場合の構造体に相当します。クラスタは、ワイヤを束にした電話ケーブルのようなものと考えることができます。ケーブルの中の各ワイヤは、クラスタの中の異なる要素を表します。コンポーネントとしては、*X*の初期値（0）、*X*の増分値（1）、Y配列（ブロックダイアグラム上の数値定数により与えられる波形データ）があります。Gでは、Bundle関数を使用してクラスタを作成します。クラスタに関する詳細は、『Gプログラミングリファレンスマニュアル』の「第14章　配列とクラスタの制御器および表示器」を参照してください。

グラフ

グラフは、1つまたは複数のプロットと呼ばれるデータ配列を2次元表示にしたものです。**制御器→グラフ**パレットには、3つのタイプのグラフがあります。

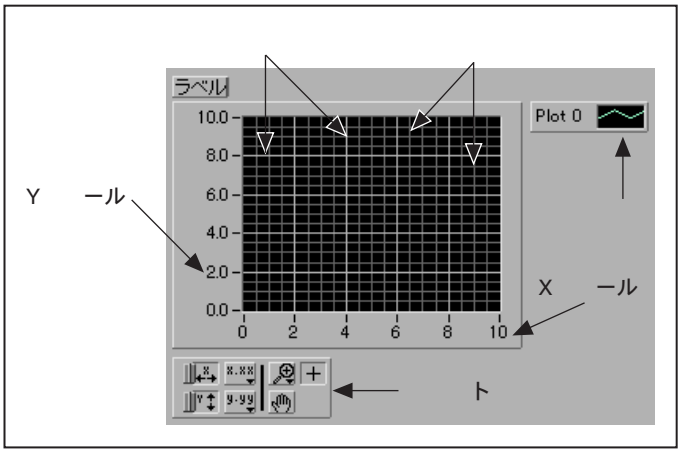
- XYグラフ
- 波形グラフ
- 強度グラフ

グラフとチャートの違う点は、グラフではブロックでデータ点がプロットされるのに対し、チャートでは点ごとまたは配列ごとにデータがプロットされることです。

グラフVIの例については、`Examples¥General¥Graphs`を参照してください。

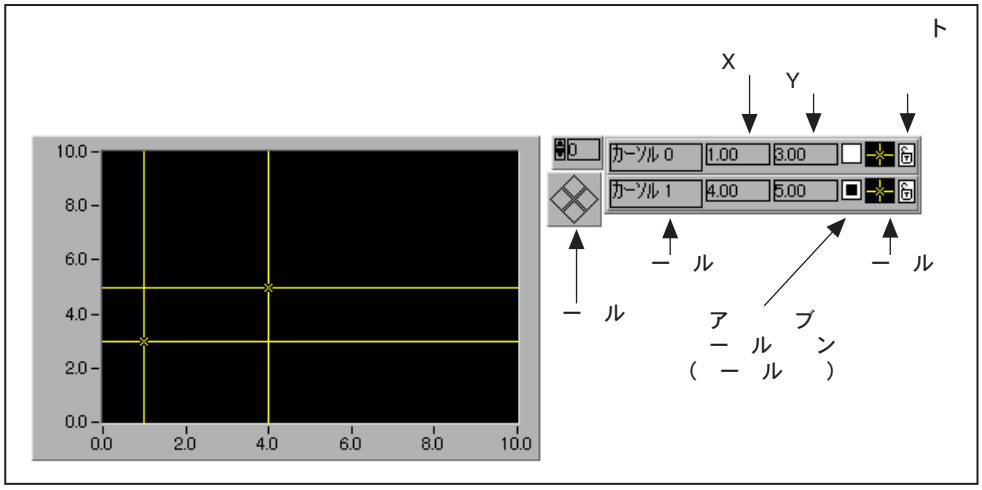
グラフをカスタマイズする

波形グラフとXYグラフは、どちらにも数多くのオプションパーツがあり、グラフに対応するポップアップメニューの**表示**サブメニューを使用することで、これらを表示したり非表示にすることができます。このようなオプションとしては、与えられたプロットの色やスタイルを定義できる凡例や、VIを実行するときの目盛りやフォーマットオプションを変更できるパレット、カーソルディスプレイなどがあります。次の図のグラフに、カーソルディスプレイを除くすべてのオプション要素を示します。



グラフカーソル

G のどのグラフでも、カーソルやカーソル表示を配置したり、プロット上のカーソルにラベルを付けることができます。プロット上にロックするようにカーソルを設定したり、同時に複数のカーソルを移動することができます。1つのグラフで使用できるカーソルの数に関して制限はありません。カーソルを含む波形グラフを、次の図に示します。



グラフのカスタマイズに関する詳細は、『G プログラミングリファレンスマニュアル』の「第15章　グラフとチャートの制御器および表示器」を参照してください。

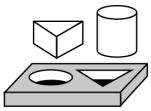
カーソル値を読み込み、カーソルを使用してプログラム的にグラフの拡大や縮小を行う例については、Examples¥General¥Graphs¥zoom.llb の ZoomGraph VI を参照してください。

グラフの軸

絶対時刻や相対時刻を示すようにグラフの目盛りをフォーマットすることができます。時刻、日付、または両方を表示する場合は絶対時刻フォーマットを使用してください。G で日付を考慮しなくてもよい場合は、相対時刻フォーマットを使用します。絶対時刻または相対時刻フォーマットを選択するには、チャートをポップアップして修正したい目盛りを選択します。**形式**... を選択すると**形式**ダイアログボックスが有効になり、チャートのさまざまな属性を指定することができます。

データ集録配列

Data Acquisition VI を使用してプラグイン式のデータ集録ボードから返されるデータには、単独の値、1D 配列、または 2D 配列の形式があります。Examples¥General¥Graphs には、配列やグラフに対してさまざまな関数を実行する VI が含まれた多くのグラフの例が用意されています。

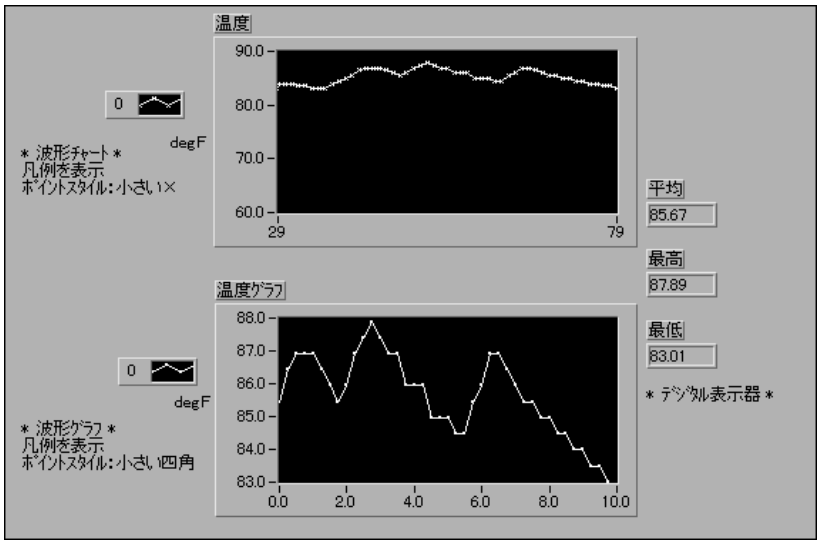


作業 5-4.　グラフ VI と解析 VI を使用する

ここでは、リアルタイムで温度を測定して表示する VI を作成することが目的です。また、この VI は温度の平均値、最大値、最小値も表示します。

フロントパネル

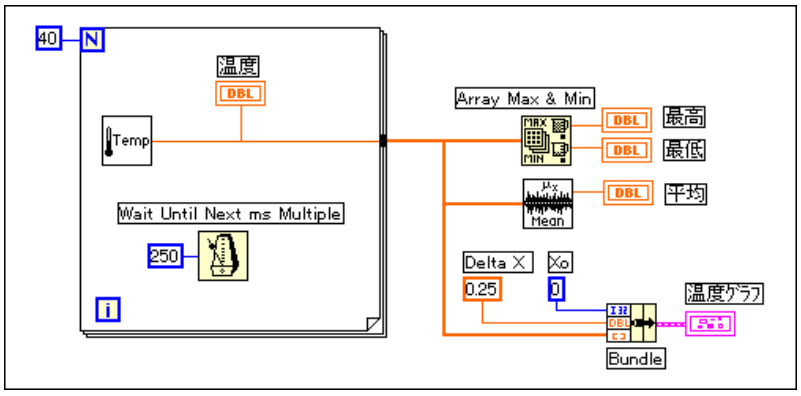
1. 次の図のように新しいフロントパネルを作成します。波形チャートや波形グラフの点のスタイルを修正するには、対応する凡例をポップアップします。チャートの目盛りを次の図のように設定します。



温度の波形チャートには、集録された温度がそのまま表示されます。集録の後、VIは[温度グラフ]にデータをプロットします。デジタル表示器[平均]、[最高]、[最低]には、温度の平均値、最大値、最小値が表示されます。

ブロックダイアグラム

2. 次の図のようにブロックダイアグラムを作成します。



Digital Thermometer VI（**関数→VI を選択する**の後 `LabVIEW¥Activity` ディレクトリから選択）— 温度の測定値を1つ返します。

Wait Until Next ms Multiple関数（**関数→時間＆ダイアログ**）— この練習問題で、この関数はFor ループが必ず0.25 秒（250 ミリ秒）ごとに実行されるようにします。

数値定数（**関数→数値**）— Wait Until Next ms Multiple関数をポップアップして**定数を作成**を選択し、自動的に数値定数を作成して配線することもできます。

Array Max & Min関数（**関数→配列**）— この作業で、この関数は集録中に測定された温度の最大値と最小値を返します。

Mean VI（**関数→解析→確率と統計**。LabVIEW Base Package ユーザの場合は**関数→基礎解析**。）— 温度の測定値の平均を返します。

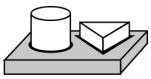
Bundle 関数（**関数→クラスタ**）— プロット要素を集めてクラスタにします。この要素には、*X* の初期値（0）、*X* の増分値（0.25）、およびY 配列（温度データ）が含まれます。関数のサイズを変更するには、位置決めツールを使用していずれかの角をドラッグします。

For ループは40 回実行されます。Wait Until Next ms Multiple 関数により250 ミリ秒ごとに実行が繰り返されます。VI は、For ループの境界部で作成された配列内に温度の測定値を格納します（自動指標付け）。For ループの実行が完了した後、配列はサブVI と[温度グラフ]に渡されます。

Array Max & Min関数は、温度の最大値と最小値を返します。Mean VI は、温度の測定値の平均を返します。

完成したVI は、X の初期値を0、X の増分値を0.25 としてデータ配列をまとめます。温度配列の点を0.25 秒ごとに波形グラフにプロットするには、X の増分値を0.25 にする必要があります。

3. フロントパネルに戻り、VI を実行します。
4. このVI を、`Temperature Analysis.vi` という名前で `LabVIEW¥Activity` ディレクトリに保存します。



これで作業 5-4. は完了です。

強度プロット

LabVIEWには、3次元データを表示するのに、強度チャートおよび強度グラフという2つの方法があります。どちらの強度プロットも、各値が色に対応付けられた、数値の2次元配列を受け取ります。色のマッピングは、オプションの色彩勾配スケールを使用して対話的操作で定義したり、チャートの属性ノードを使用してプログラム的に定義することができます。強度チャートと強度グラフの使用例については、Examples¥General¥Graphs¥llbディレクトリのintgraph.llbを参照してください。

6

文字列とファイル I/O

この章では、文字列制御器と表示器、ファイルの入出力動作を紹介し、作業を行いながら以下の操作方法を説明します。

- 文字列制御器と表示器を作成する
- 文字列関数を使用する
- ファイルの入出力動作を行う
- データをスプレッドシート形式でファイルに保存する
- テキストファイルに対してデータの書き込みと読み込みを行う

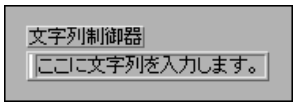
文字列

文字列とは、ASCII 文字の集合体です。計測器制御では、数値データを文字列として渡してから、その文字列を数値に変換することができます。数値データをディスクに格納するときにも、文字列としてデータを書き込むこともできます。数値を ASCII ファイルに格納するには、数値をディスクファイルに書き込む前にまず数値を文字列に変換する必要があります。

文字列の例については、`Examples¥General¥strings.llb` を参照してください。

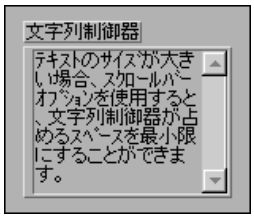
文字列制御器と表示器を作成する

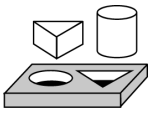
制御器→文字列 & 表では、左の図のような文字列制御器と表示器が表示されます。テキストを入力したり変更するには、操作ツールやラベリングツールを使用します。文字列制御器や表示器を拡大するには、位置決めツールで角をドラッグします。



文字列とファイルI/O

フロントパネルの文字列制御器や表示器が占める面積を最小限におさえたい場合は、**表示→スクロールバー**を選択します。このオプションを選択できない場合は、制御器のサイズを縦に大きくして使用できる状態にする必要があります。



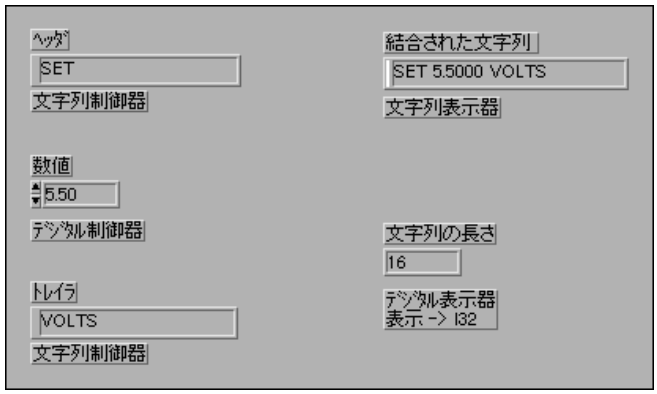


作業 6-1.　文字列を連結する

LabVIEWには、文字列を操作する多くの関数があります。この作業では、いくつかの文字列関数を使用して数値を文字列に変換し、その文字列を他の文字列と連結して1つの出力文字列にします。

1. 新しいフロントパネルを開き、次の図のようにオブジェクトを追加します。制御器と表示器は、必ず図と同じになるように修正してください。

フロントパネル

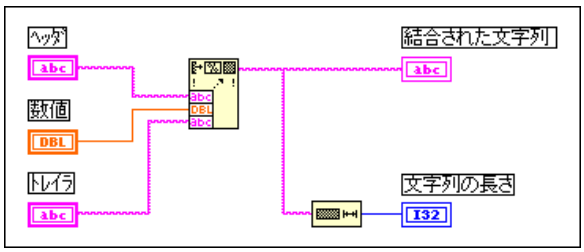


2つの文字列制御器とデジタル制御器を組み合わせて1つの出力文字列にし、文字列表示器に表示することができます。デジタル表示器には文字列の長さが表示されます。

この作業の統合された文字列出力の形式は、GPIB（IEEE488）計測器やシリアル（RS-232 または RS-422）計測器との通信に使用されるコマンド文字列の形式と似ています。計測器コマンドに使用される文字列についての詳細は、本書の「第II部 I/Oインタフェース」を参照してください。

ブロックダイアグラム

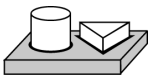
2. 次の図のようにブロックダイアグラムを作成します。



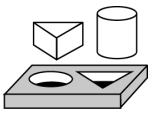
Format Into String関数（**関数→文字列**）は、数値と文字列を連結してフォーマットし、1つの出力文字列にします。位置決めツールを使用し、3つのパラメータ入力を追加します。

String Length関数（**関数→文字列**）は、連結された文字列の文字数を返します。

3. VIを実行します。Format Into String関数が2つの文字列制御器とデジタル制御器を連結して1つの出力文字列を生成することを確認してください。
4. このVIを、`Build String.vi`という名前で保存します。このVIは、次の練習問題で使用します。



これで作業 6-1 は完了です。



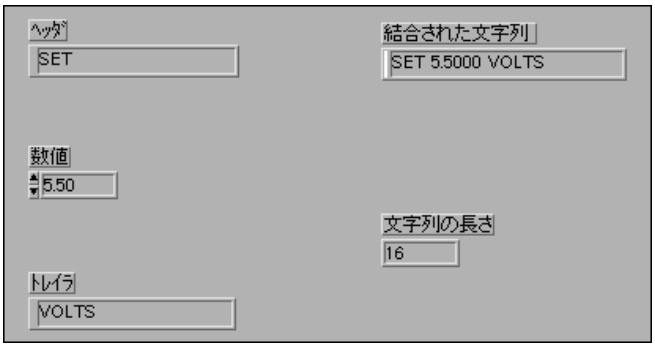
作業 6-2.　形式文字列を使用する

ここでは、指定した形式に従って文字列を作成することが目的です。

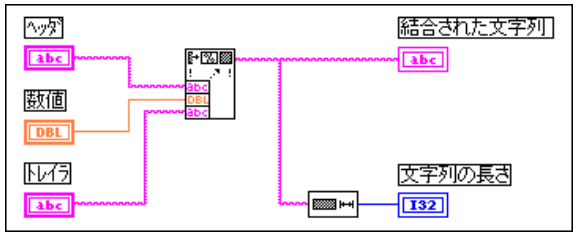
作業6-1で作成したBuild String VIを使用して形式文字列を作成します。形式文字列を使用すると、フィールド幅、基数（8進数や16進数など）、パラメータを分離するテキストなど、パラメータの形式を指定することができます。

フロントパネル

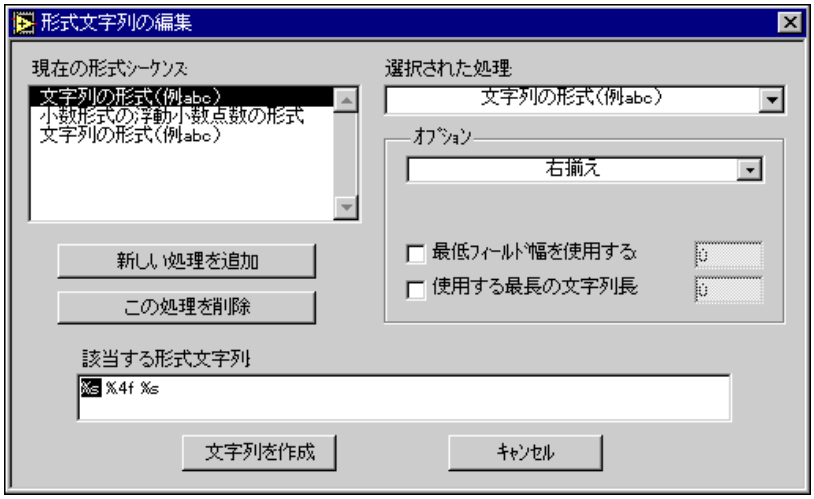
1. 作業6-1で作成したBuild String VIを開きます。



ブロックダイアグラム



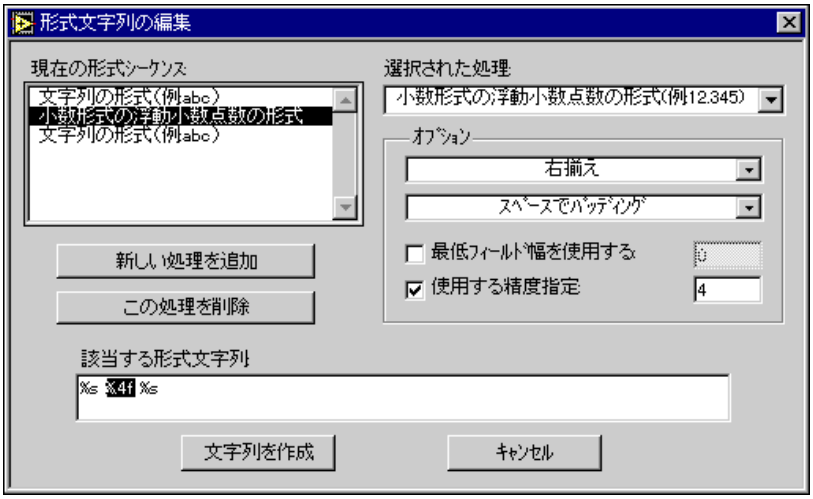
2. Format Into String VIでポップアップして**形式文字列の編集**を選択します。次のようなダイアログボックスが表示されます。



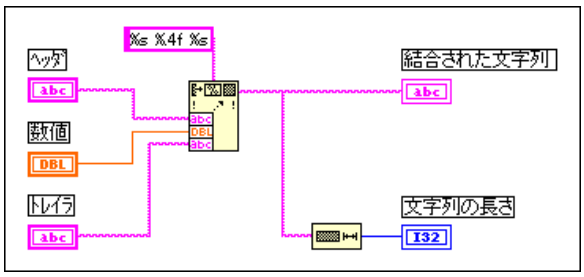
☞ **注**　**ノードをダブルクリックして、形式文字列の編集ダイアログボックスにアクセスすることもできます。**

現在の形式シーケンスには、配線した順序でパラメータタイプが含まれていることを確認してください。

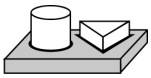
3. 数値の精度を4に設定します。
   a. 現在の形式シーケンスリストボックスの[小数形式の浮動小数点数の形式]をハイライトにします。
   b. 使用する精度指定チェックボックスの中をクリックします。
   c. 使用する精度指定チェックボックスの横の数字をハイライト表示して4を入力し、`<Enter>`**（Windows）**、`<return>`**（Macintosh）**、`<Return>`**（Sun）**、`<Enter>`**（HP-UX）**を押します。次の図は、数値の精度を設定するためにオプションが選択された状態を示します。



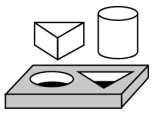
4. **文字列を作成**ボタンを押します。このボタンを押すと、次の図のように、自動的に正しい形式文字列情報が挿入され、形式文字列が関数に配線されます。



5. フロントパネルに戻り、2つの文字列制御器にテキストを入力し、デジタル制御器に数値を入力します。VIを実行します。
6. VIを保存して閉じます。このVIには、`Fomat String.vi`という名前を付けます。



これで作業 6-2 は完了です。

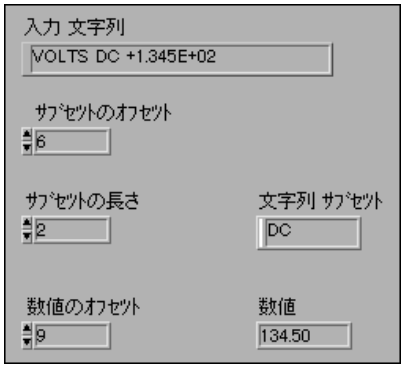


作業 6-3.　文字列サブセットと数値の抽出

ここでは、数値を表現した文字列を含む文字列サブセットを抽出して数値に変換します。

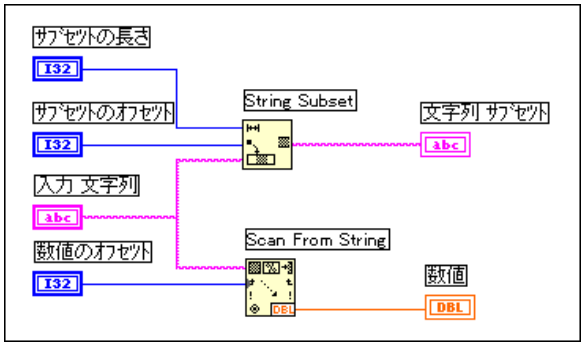
フロントパネル

1. Examples¥General¥strings.llbからParse string.viを開きます。デフォルト入力を使用してVIを実行します。**DC**の文字列サブセットが入力文字列として選択されることを確認してください。また、文字列の中の数値の部分が数値に変換されたことも確認してください。別の制御値で試したり（文字列の指標は配列と同様0から始まることを忘れないでください）、ブロックダイアグラムを表示して入力文字列から要素が解析される状態を確認することもできます。



ブロックダイアグラム

2. 次の図のように、Parse String VIのブロックダイアグラムを開きます。



LabVIEWは、String Subset関数とScan From String関数を使用して入力文字列を解析します。

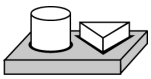
String Subset関数（**関数→文字列**）は、**オフセット**から始まり**文字数**分の部分文字列を返します。最初の文字のオフセットは0になります。

計測器から受け取ったデータ文字列をデータ値に変換するなど、多くの場合に、文字列を数値に変換する必要があります。

Scan From String関数（**関数→文字列**）は、文字列をスキャンして、有効な数字（0、9、+、–、e、E、ピリオド）を数値に変換します。形式文字列を配線してある場合、Scan From Stringは形式に従って変換を行います。また形式文字列を配線していない場合、関数内の各デフォルト入力端子ごとにデフォルトの変換を行います。この関数は、**オフセット**から**入力文字列**のスキャンを始めます。最初の文字のオフセットは0になります。

ヘッダの長さ（この例の場合の[VOLTS DC]）がわかっている場合や、文字列の中に有効な数字しか含まれていない場合には、Scan From String関数が役に立ちます。

3. **ファイル→閉じる**を選択してVIを閉じます。VIの保存は行いません。



これで作業 6-3 は完了です。

ファイルI/O

Gのファイル I/O 関数（**関数→ファイル I/O**）は、ファイルに関する操作を行うための強力で柔軟性の高いツールです。LabVIEW のファイル I/O 関数は、データの読み込みと書き込みだけでなく、ファイルやディレクトリの移動と名称変更、読み込み可能な ASCII テキストのスプレッドシートタイプのファイル作成、サイズの小さいバイナリ形式での高速データ書き込みなどを行います。

ファイルに対しては、次の 3 つの異なる形式でデータを格納したり取り出すことができます。

- ASCIIバイトストリーム ― ワープロやスプレッドシートプログラムなど他のソフトウェアパッケージからデータにアクセスしたい場合は、データを ASCII 形式で格納する必要があります。この方法でデータを格納するには、すべてのデータを ASCII 文字列に変換しなければなりません。
- データログファイル ― このファイルはGでのみアクセスできるバイナリ形式です。データログファイルはデータベースファイルに似ており、タイプの異なる複数のデータを 1 つのファイルの 1 つの（ログ）レコードに格納することができます。
- バイナリバイトストリーム ― このファイルは、データを格納する方法としては最もサイズが小さく、高速です。データをバイナリ文字列形式に変換する必要があり、ファイルに対してデータの保存／検索を行うときに使用するデータタイプが正確にわかっていることが必要です。

ここでは、最も一般的なデータファイル形式である ASCII バイトストリームファイルについて説明します。ファイル I/O の例は、`Examples¥File` を参照してください。

ファイルI/O 関数

ほとんどのファイルI/O動作には、既存のファイルを開くまたは新しいファイルを作成する、ファイルに対して書き込みや読み込みを行う、ファイルを閉じる、という 3 つの基本的なステップがあります。このため LabVIEW では、多くのユーティリティ VI が**関数→ファイル I/O** の中に用意されています。ここでは、9 つの高水準ユーティリティについて説明します。これらのユーティリティ関数は、ファイル I/O 関数を利用したエラーチェックとエラー処理の機能が組み込まれた中級 VI を使用して作成されています。

Write Characters To File VIは、新しいバイトストリームファイルに文字列を書き込んだり、既存のファイルに文字列を追加します。このVIは、ファイルを開くかまたは作成してデータを書き込み、ファイルを閉じます。

Read Characters From File VIは、指定された文字オフセットから始まる指定された数の文字を、バイトストリームファイルから読み込みます。このVIは、処理の前にファイルを開き、処理の後にファイルを閉じます。

Read Lines From File VIは、指定された文字オフセットから始まる指定された数の行をバイトストリームファイルから読み込みます。このVIは、処理の前にファイルを開き、処理の後にファイルを閉じます。

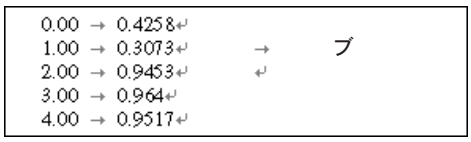
Write To Spreadsheet File VIは、単精度の数値からなる1次元または2次元配列をテキスト文字列に変換し、その文字列を新しいバイトストリームファイルに書き込むか、既存のファイルに追加します。選択的にデータを置き換えることができます。このVIは、処理の前にファイルを開き、処理の後にファイルを閉じます。このVIを使用すると、ほとんどのスプレッドシートプログラムで読み込み可能なテキストファイルを作成することができます。

Read From Spreadsheet File VIは、指定された文字オフセットから始まる指定された数の行または列を数値テキストファイルから読み込み、そのデータを2次元の単精度数値配列に変換します。選択的に配列を置き換えることができます。このVIは、処理の前にファイルを開き、処理の後にファイルを閉じます。このVIを使用すると、テキスト形式で保存されたスプレッドシートファイルを読み込むことができます。

その他のファイルI/O関数を参照するには、**関数→ファイルI/O→バイナリファイル**VIまたは**関数→ファイルI/O→上級ファイル関数**を選択します。

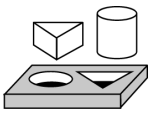
スプレッドシートファイルに書き込む

データをファイルに保存するアプリケーションでごく一般的なもののひとつは、スプレッドシートで開くことができるようにテキストファイルをフォーマットする処理です。ほとんどのスプレッドシートでは、次の図のように列の区切りにはタブが、行の区切りにはEOL（End of Line：行末）文字が使用されています。



スプレッドシートプログラムを使用してファイルを開くと、次のような表が作成されます。

	A	B	C
1	0	0.4258	
2	1	0.3073	
3	2	0.9453	
4	3	0.964	
5	4	0.9517	
6			

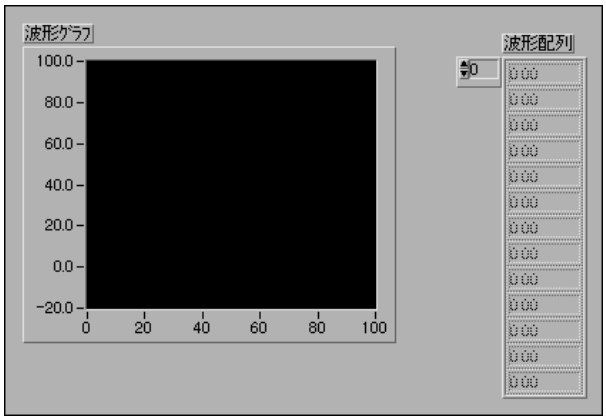


作業 6-4.　スプレッドシートファイルに書き込む

ここでは、ファイルI/O関数を使用して新しいファイルにASCII形式でデータを保存できるように既存の VI を修正することが目的です。後で、スプレッドシートアプリケーションからこのファイルにアクセスできます。

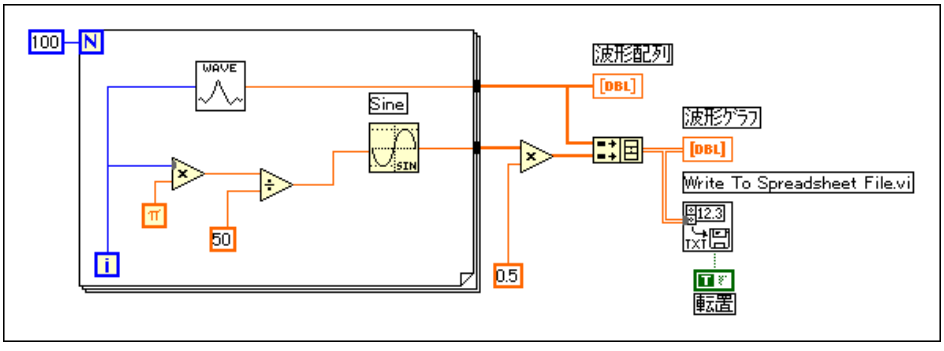
フロントパネル

1. 作業5-1で作成したGraph Waveform Arrays.viを開きます。この VI は 2 つのデータ配列を生成して、グラフ上にプロットします。この VI を修正して 2 つの配列をファイルに書き込みます。各列にはデータ配列が含まれます。



ブロックダイアグラム

2. Graph Waveform Arrays.viのブロックダイアグラムを開き、次の図の右下に追加されたブロックダイアグラムの関数を追加して、VI を修正します。



Write To Spreadsheet File VI（**関数→ファイルI/O**）は、2次元配列をスプレッドシート文字列に変換してファイルに書き込みます。パス名を指定してない場合は、ファイルダイアログボックスが現れ、ファイル名を聞いてきます。Write To Spreadsheet Fileは1次元または2次元のいずれかの配列をファイルに書き込みます。この例のデータ配列は2次元ですので、1次元入力には配線する必要がありません。このVIでは、データ内でタブやカンマなどのようなスプレッドシートの区切りまたは区切り文字列が使用できます。

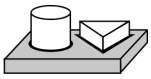
ブール定数（**関数→ブール**）は、Gが2次元配列を転置してからファイルに書き込むかどうかを制御します。値をTRUEに変更するには、操作ツールで定数をクリックします。この場合はデータ配列が行に固有のものであるため（2次元配列の各行はデータ配列）、データを転置する必要があります。スプレッドシートファイルの各列にはデータ配列が含まれているため、最初に2次元配列を転置する必要があります。

3. フロントパネルに戻ってVIを実行します。データ配列が生成されるとファイルダイアログボックスが現れ、作成する新しいファイルの名前を聞いてきます。ファイル名を入力して**OK**をクリックします。

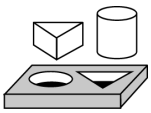
注意　`mywork.llb`などのVIライブラリにはデータを書き込まないでください。このような操作を行うと、ライブラリに上書きされて前の作業結果が失われる恐れがあります。

4. VIを保存して`Waveform Arrays to File.vi`という名前を付け、VIを閉じます。
5. こうしておくと、先ほど作成したファイルを、スプレッドシートソフトウェアやテキストエディタを使用して開き表示することができます。100個の要素からなる2つの列が表示されるはずです。

この例では、データ配列の収集が完了するまで、データの変換やファイルへの書き込みは行われません。サイズの大きいデータバッファを集録する場合や、データを生成すると同時にデータ値をディスクに書き込んでいきたい場合は、別のファイルI/O VIを使用する必要があります。



これで作業 6-4 は完了です。



作業 6-5.　ファイルにデータを追加する

ここでは、温度データをASCII形式でファイルに追加するVIを作成するのが目的です。このVIは、Forループを使用して温度値を生成し、ファイルに格納します。毎回の反復で、データを文字列に変換し、区切り文字のカンマを追加し、文字列をファイルに追加します。

フロントパネル

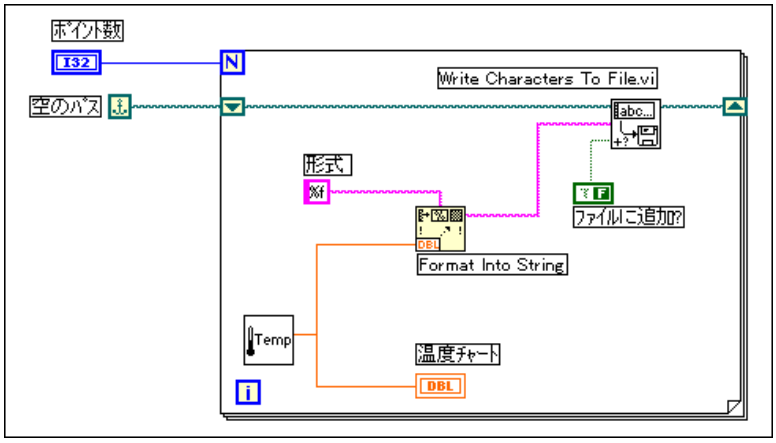
1. 次の図のように、新しいフロントパネルを開いてオブジェクトを配置します。

フロントパネルには、デジタル制御器と波形チャートが含まれています。**表示→デジタル表示**を選択します。[ポイント数]制御器は、集録してファイルに書き込む温度値の数を指定します。チャートには、温度曲線が表示されます。チャートのY軸の目盛りの範囲を70.0～90.0に、X軸の目盛りの範囲を0～20に設定し直します。

2. デジタル制御器のポイント数をポップアップして**表記法→I32**を選択します。

ブロックダイアグラム

3. ブロックダイアグラムを開きます。



4. For ループを追加してサイズを拡大します。この VI は、[ポイント数] 制御器で指定された数の温度値を発生します。
5. ループの枠をポップアップしてループにシフトレジスタを追加します。このシフトレジスタには、ファイルのパス名が含まれています。
6. オブジェクトの配線を終了します。

Empty Path 定数（**関数→ファイル I/O →ファイル定数**）。Empty Path 関数は、最初にファイルに値を書き込むときにパスが空になるようにシフトレジスタを初期化します。ファイルダイアログボックスが、ファイル名を入力するように要求してきます。

Digital Thermometer VI は、温度センサからの疑似的な温度測定値を返します。

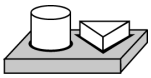
Format Into String 関数（**関数→文字列**）は、温度測定値（数値）を文字列に変換して後ろにカンマを付けます。

文字列定数（**関数→文字列**）。この形式文字列は、数値を分数形式の文字列に変換して後ろにカンマを付けるように指定します。

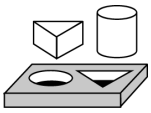
Write Characters To File VI（**関数→ファイル I/O**）は、文字列をファイルに書き込みます。

ブール定数（**関数→ブール**）は、ループが繰り返すときに選択したファイルに新しい温度値が追加されるように、Write Characters To File VI の [ファイルに追加] を TRUE に設定します。値を True に設定するには、その定数を操作ツールでクリックします。

7. フロントパネルに戻り、[ポイント数] を 20 に設定した状態で VI を実行します。ファイルダイアログボックスがファイル名を聞いてきます。ファイル名を入力すると、VI は、各点が生成されるたびにそのファイルに温度値を書き込む動作を開始します。
8. この VI を、`Write Temperature to File.vi` という名前で `LabVIEW¥Activity` ディレクトリに保存します。
9. Write for Windows や Teach Text for Macintosh、および UNIX のテキストエディタなど、任意のワープロソフトウェアを使用してそのデータファイルを開き、内容を表示します。カンマで区切られた（小数点以下 3 桁の精度の）20 個のデータ値を含むファイルが表示されるはずです。



これで作業 6-5 は完了です。

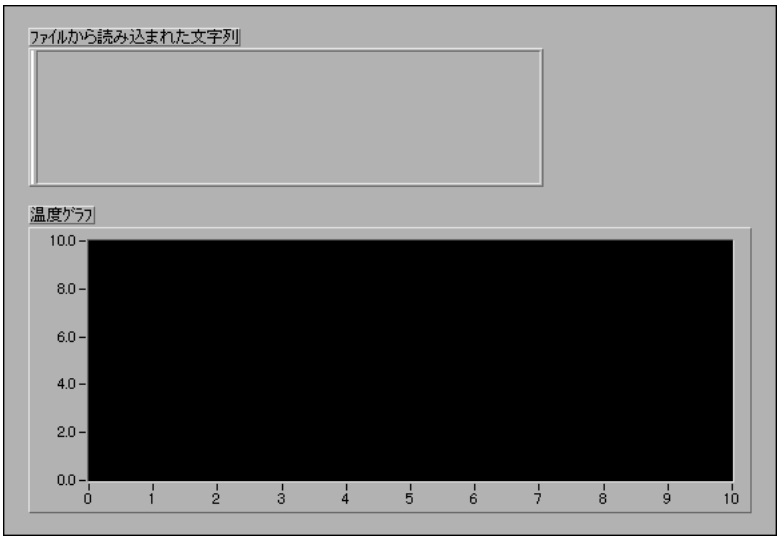


作業 6-6.　ファイルからデータを読み込む

ここでは、前の例で書き込んだデータファイルを読み込み、データを波形グラフに表示する VI を作成することが目的です。データは、保存したときと同じデータ形式で読み込む必要があります。したがって、前に文字列データタイプを使用して ASCII 形式でデータを保存してありますので、いずれかのファイル I/O VI を使用して文字列データとしてデータを読み込む必要があります。

フロントパネル

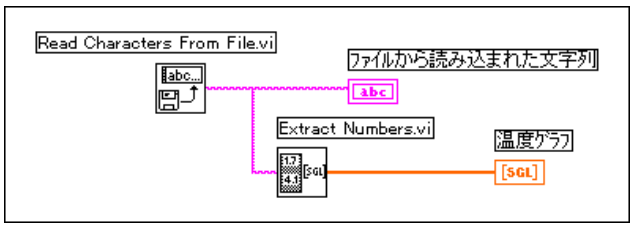
1. 新しいフロントパネルを開き、次の図のようなフロントパネルを作成します。



フロントパネルには、文字列表示器と波形グラフが含まれています。ファイルから読み込んだ文字列表示器には、前の作業で書き込んだファイルのカンマで区切られた温度データが表示されます。グラフには温度曲線が表示されます。

ブロックダイアグラム

2. 次の図のようにブロックダイアグラムを作成します。



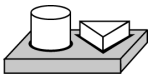
Read Characters From File VI（**関数→ファイルI/O**）は、ファイルからデータを読み込んで情報を文字列で出力します。パス名が指定されていない場合は、ファイルダイアログボックスがファイル名を入力するよう要求してきます。この例ではデフォルトである 512 文字より少ない数の文字しかファイル内にありませんので、読み込む文字数を決める必要はありません。

データがファイルに格納されたときの格納方法がわかっていないと、データを読み出すことができません。ファイルの長さがわかっている場合は、

Read Characters From File VIを使用して、既知の読み込み文字数を判断することができます。

Extract Numbers VI（Examples¥General¥strings.llb）は、カンマ、改行などの数字でない文字で区切られた数値を含むASCII文字列を取り込んで、数値の配列に変換します。

3. フロントパネルに戻ってVIを実行します。ファイルダイアログボックスが要求してきたら、先程ディスクに書き込んだデータファイルを選択します。Write Temperature to File VIの例で表示されたのと同じデータ値がグラフに表示されるはずです。
4. このVIを保存してTemperature from File.viという名前を付け、VIを閉じます。



これで作業 6-6 は完了です。

ファイルI/O関数を使用する

場合によっては、データをディスクに保存する際にユーザが必要とする機能が、簡易ファイルI/O関数では得られないことがあります。このような場合は、**関数→ファイルI/O→上級ファイル関数**の関数を使用する必要があります。

ファイルを指定する

ファイルを指定する方法としては、プログラムによる方法と、ダイアログボックスを使用する方法の2つがあります。プログラムによる方法では、VIのファイル名とパス名を指定します。

(Windows) パス名は、ドライブ名（たとえばC）、および後に続くコロン、円コード（¥）で区切られたディレクトリ名、ファイル名から構成されます。たとえばC:¥DATADIR¥TEST1は、Cドライブの DATADIR ディレクトリ内のTEST1という名前のファイルを示します。

(Macintosh) パス名は、ドライブ名、および後に続くコロン、コロンで区切られたフォルダ名、ファイル名から構成されます。たとえばHardDrive:DataFolder:Test1は、HardDriveというボリュームのフォルダDataFolderの中のTest1という名前のファイルを示します。

(UNIX) パス名は、スラッシュで区切られたディレクトリ名と後に続くファイル名から構成されます。たとえば/usr/datadirectory/test1

は、ディレクトリ /usr/datadirectory の中の testl というファイルを示します。

（すべてのプラットフォーム）ダイアログボックスを使用する方法の場合、File Dialog 関数がダイアログボックスを表示しますので、これを使用して対話形式によりディレクトリを検索してからファイル名を入力することができます。

パスと refnum

パスとは、ファイルを識別するためのGのデータタイプです。ファイルパスを入力したり表示するには、パス制御器とパス表示器に各プラットフォームの標準構文を使用します。パス制御器とパス表示器の動作は多くの点で文字列制御器や文字列表示器と似ていますが、G でサポートされているプラットフォームに合わせてGがパスを形式する点が異なります。

refnum は、開いているファイルを識別するGのデータタイプから構成されます。ユーザがファイルを開くと、G はそのファイルに対応付けられた refnum を返します。開いているファイルに対して実行される動作はすべて、このファイルの refnum を使用して各ファイルを識別します。refnum は、ファイルが開いている間のみ有効です。ファイルを閉じると、G は refnum とそのファイルとの間の関係を解除します。その後ファイルを開いたときに付けられる新しい refnum は、G が以前に使用した refnum とは異なることがあります。

G は、操作とファイルを対応付けるだけでなく、ファイルからの現在の読み込み位置や、他のユーザのそのファイルへのアクセス許容度など、各 refnum についての情報を記憶しています。そのため、1 つのファイルに対して、同時に独立した操作を行えます。1 つのファイルを何回も開いた場合、開いたファイルに対する各操作ごとに、異なる refnum が返されます。

ファイルI/Oの例

次の例を使用すると、適切なエラーチェックとエラー処理機能を持つ完全なファイルI/O関数の使用方法を確認できます。

Write to Text File VI（Examples¥File¥smplfile.llbに入っています）は、測定時刻の付いたデータ値を含むASCIIテキストファイルを書き込みます。

Read from Text File VI（Examples¥File¥smplfile.llb に入っています）は、測定時刻の付いたデータ値を含むASCIIテキストファイルを読み込みます。

データログファイル

この章に出てくる例では、一連のASCII文字として格納されたデータを含むファイルを処理する簡単な方法が示されています。スプレッドシートプログラムのような他のソフトウェアパッケージで読み込むファイルを作成する場合は、この方法がよく使用されます。Gには、この他に**データログファイル**というファイル形式があります。データログファイルは、ユーザがファイルを作成したときに決定した任意の1つのデータタイプのレコードのシーケンスとして、データを格納します。Gは、これらのレコードに従ってデータログファイル内のデータに指標を付けます。データログファイル内のすべてのレコードは1つのタイプでなければなりませんが、そのタイプは複合的なものにすることができます。たとえば、文字列、数値、配列を含むクラスタがレコードに含まれるように各レコードを設定することができます。

VIを使用してデータを取り出す場合は、データと文字列との間の変換に余分な時間がかかりますので、データをASCIIファイルに書き込まない方がよいでしょう。たとえば2次元配列を、ヘッダとタイムスタンプを含むスプレッドシート形式の文字列に変換する処理は複雑です。他のアプリケーションで読み込み可能な形式でデータを格納する必要がない場合は、データをデータログファイルとして書き出すのがよいでしょう。この形式では、データをファイルに書き込む場合にわずかの操作で済み、書き込みと読み込みがはるかに高速になります。また、データの元のブロックをログまたはレコードとして読み出すことができるため、レコードに含まれるデータのバイト数がわかっている必要がなくなり、データの読み込みが簡単になります。Gは、データログファイルの各レコードのデータ数を記録しています。

Write Datalog File Example（`Examples¥File¥datalog.llb`）は、新しいデータログファイルを作成し、指定された数のレコードをファイルに書き込みます。各レコードは、文字列と単精度数の配列を含むクラスタです。

データログファイルを読み込むには、ファイルに書き込みを行ったときに使用されたのと同じデータタイプを使用する必要があります。Read Datalog File Example（`Examples¥File¥datalog.llb`）は、Write Datalog File Exampleで作成されたデータログファイルを一度に1レコードずつ読み込みます。読み込まれるレコードは、文字列と単精度数の配列を含むクラスタで構成されています。

第II部

I/Oインタフェース

第II部では、データを入出力するための、データ集録、GPIB、シリアル、VXIなどのインタフェースに関する基本的な事項を説明します。リアルタイムデータ集録についての基本事項は、『データ集録ベーシックマニュアル』を参照してください。VISA（Virtual Instrument Software Architecture：仮想計測器ソフトウェアアーキテクチャ）は、GPIB、シリアル、VXI計測器とインタフェイスするための単独のソフトウェアAPIです。特定の計測器用に特別に開発されたLabVIEWアプリケーションは、計測器ドライバと呼ばれます。ナショナルインスツルメンツではVISAライブラリを使用したいくつかの計測器ドライバを提供していますが、ユーザ独自の計測器ドライバを作成することもできます。

「第II部 I/Oインタフェース」は、以下の章から構成されています。

- 「第7章 LabVIEW計測器ドライバ入門」では、ナショナルインスツルメンツの計測器ドライバの作成方法と使用方法を説明します。
- 「第8章 LabVIEW VISAチュートリアル」では、イベント、ロック、メッセージベースの通信、レジスタベースの通信などを使用して共通のVISAアプリケーションを実装する方法を示します。
- 「第9章 LabVIEWのGPIB関数の概要」「第9章 LabVIEW GPIB関数の概要」では、GPIBの動作、およびIEEE 488とIEEE 488.2インタフェースの違いについて説明します。
- 「第10章 シルアルポートVI」では、シリアルポート通信用のVIを紹介し、シリアル通信に影響を及ぼす重要な要素について説明します。

7

LabVIEW 計測器ドライバ入門

この章では、計測器ドライバライブラリから計測器ドライバをインストールして使用する方法をまず始めに説明し、最後にユーザ独自の計測器ドライバを作成する方法を紹介します。この章では、計測器との間の通信状態のチェック、計測器ドライバを使用したアプリケーションの開発、計測器ドライバの作成でよく使用される技法を段階を追って説明します。

LabVIEW の計測器ドライバとは？

計測器ドライバとは、LabVIEW の標準 VISA I/O 関数を使用して計測器と通信を行う一連の LabVIEW の VI です。各 VI はプログラムによって、計測器に対する構成、読み出し、書き込み、トリガなどの操作を行います。LabVIEW の計測器ドライバを使用すると、各計測器ごとに複雑な上級プログラミングコマンドを覚える必要がありません。

LabVIEW の計測器ドライバライブラリには、GPIB、VXI、シリアルインタフェースを使用するさまざまなプログラマブル計測器用の計測器ドライバが含まれています。ライブラリドライバは、そのままで計測器に使用することができますが、計測器ドライバにはブロックダイアグラムのソースコードが添付されているので、必要に応じ、個々のアプリケーションに合わせてこれらをカスタマイズすることもできます。

計測器ドライバの入手方法

計測器ドライバは、計測器ドライバCDからインストールできるほか、ナショナルインスツルメンツのホームページからダウンロードすることもできます。最新の計測器ドライバ注文用紙を入手するには、プッシュボタン式の電話で米国ナショナルインスツルメンツの自動ファックスシステムであるFax Backサポートにアクセスしてください。番号は(512) 418-1111（米国）または(800) 329-7177（米国）です。また、インターネットの計測器ドライバネットワークを使用してドライバをダウンロードすることもできます。このネットワークにアクセスするには、`http://www.natinst.com/idnet`に接続してください。

特定の計測器用の計測器ドライバが存在しない場合は、

1. 類似の計測器用ドライバを試してみてください。同じメーカー製の類似の計測器では、コマンドセットが全く同じでなくても似ている場合が多くあります。
2. この章の「短時間で簡単にLabVIEW計測器ドライバを開発する」の項のガイドラインに沿って、計測器ドライバを作成します。
3. 高機能の完全な計測器ドライバを開発します。ナショナルインスツルメンツのものと同等のドライバを開発するためには、当社のホームページから『Application Note 006、Developing a LabVIEW Instrument Driver』をダウンロードしていただけます。完全な計測器ドライバを開発するには、このアプリケーションノートが役に立ちます。

LabVIEW計測器ドライバのインストール先

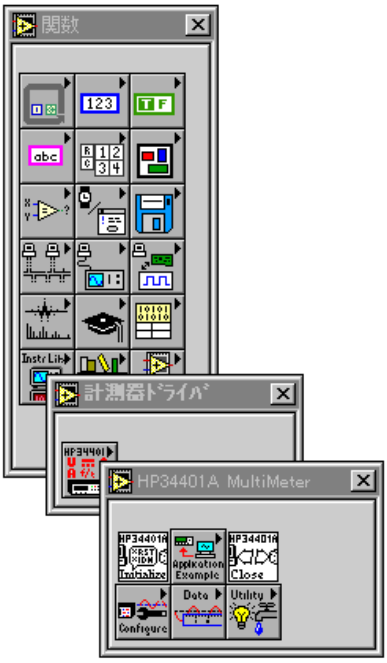
計測器ドライバは、`LabVIEW¥instr.lib`のサブディレクトリとしてインストールする必要があります。たとえば、LabVIEWに含まれているHP34401A計測器ドライバは、次のディレクトリにインストールされます。

```
LabVIEW¥instr.lib¥hp34401a
```

このディレクトリには、メニューファイル、および計測器ドライバを構成するためのVIライブラリがあります。このメニューファイルを使用すると、**関数**パレットから計測器ドライバVIを表示できます。VIライブラリには、計測器ドライバVIが含まれています。

計測器ドライバVIへのアクセス方法

計測器ドライバVIは、**関数**パレットの一番下近くの**計測器ドライバ**サブパレットに表示されます。



多くの計測器ドライバにはメニューパレットがあり、次のような要素が含まれています。

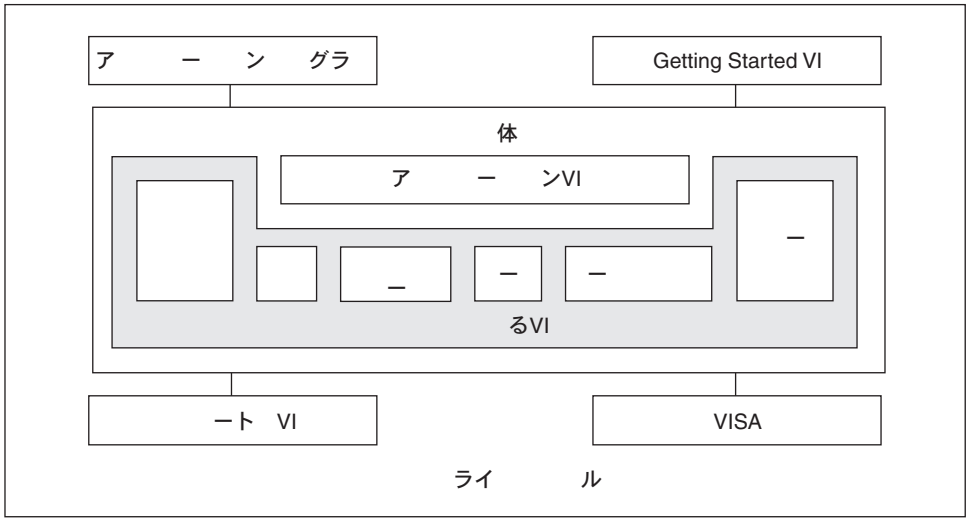
- 初期化VI
- クローズVI
- 使用例サブパレット
- 構成サブパレット
- アクション／ステータスサブパレット
- データサブパレット
- ユーティリティサブパレット

計測器ドライバVIには、**関数**パレットから**VIを選択**オプションを使用してアクセスすることもできます。計測器ドライバの階層全体を表示するに

は、VI Tree VIを開きます。これは実行不可能なVIであり、計測器ドライバの機能ストラクチャを示すためのものです。

計測器ドライバストラクチャ

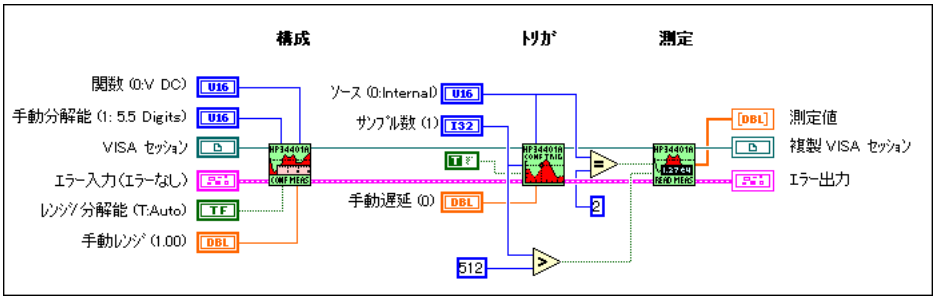
次の図に、標準的な計測器ドライバの構成を示します。このモデルを理解すると、これが多くの計測器ドライバに当てはまることがわかります。



サンプルアプリケーションは、修正せずに使用できる簡単なアプリケーションVIです。計測器との間の通信を確認するにはこのVIを実行します。通常、フロントパネルからVIを実行する前に必要となるのは、計測器アドレスの変更のみですが、まれに、(GPIB::2のような）VISAリソース名を指定することが必要になる場合があります。VISAリソース名に関する詳細は、「第8章 LabVIEW VISAチュートリアル」を参照してください。サンプルアプリケーションは通常、初期化VI、アプリケーションVI、閉じるVIという3つのサブVIから構成されます。

アプリケーションVIは、プログラムによる一般的な計測器操作を実行するのに下位要素関数をまとめた高レベルのサンプルです。アプリケーションVIの例としては、最もよく使用される計測器の構成や測定を制御するためのVIなどがあります。これらのVIは、計測器の構成、トリガ、測定値の読み込みなどの一般的な操作を実行するコードの例として役に立ちます。

アプリケーションVIはアイコンやコネクタペーンを含む標準VIですので、ドライバに対する単独の測定指向インタフェースが必要な場合に任意の上位アプリケーションからこれらを呼び出せます。多くのユーザにとって、計測器制御に必要な計測器ドライバ VI はアプリケーション VI のみです。次の図のHP34401 Example VI で、アプリケーション VI のフロントパネルとブロックダイアグラムを示します。



最初に呼び出される計測器ドライバVIである**初期化VI**は、計測器との間の通信を確立します。また、初期化VIは計測器をデフォルトの電源ON状態または他の特定の状態にするのに必要な任意の処理を実行できます。通常は、アプリケーションプログラムの最初に**初期化 VI** を呼び出すだけでOKです。

構成VIは、希望する動作を実行するように計測器を構成するソフトウェアルーチンを集めたものです。計測器によって、数多くの**構成 VI** があります。これらのVIが呼び出されると、計測器は測定を行ったりシステムに指示を送ることができる状態になります。

アクション／ステータスに分類されるVIには、2つのタイプがあります。**アクションVI**は、テスト動作や測定動作を起動したり終了します。このような動作には、トリガシステムを準備状態にする、トリガ信号を発生する、などの動作も含まれます。アクションVIは**構成VI**と異なり計測器の設定値を変更することはなく、現在の構成に応じて動作を実行するように計測器に指示するのみです。**ステータスVI**は、計測器の現在のステータスや、保留状態の動作のステータスを取得します。

データVIは、計測器との間のデータ転送を行います。このような例としては、測定用の計測器から測定値や波形を読み込むVI、ソース計測器に波形やデジタルパターンをダウンロードするVIなどがあります。

ユーティリティVIは、最も頻繁に使用される計測器ドライバを補助するさまざまな動作を行います。リセット、自己テスト、レビジョン、エラークエリ、エラーメッセージなど、大部分の計測器ドライバテンプレートVIはユーティリティVIです。また、キャリブレーションや記憶、設定値の再読み込みなどの動作を行う他のカスタム計測器ドライバVIも、ユーティリティVIに含まれす。

クローズVIは、計測器へのソフトウェア接続を終了し、システムリソースを解放します。一般に、**クローズVI**は、アプリケーションプログラムの終了時または計測器との通信の終了時に一度呼び出すだけです。それぞれの初期化VI呼び出しが正常に行われ、それに対応する**クローズVI**が用意されていることを確認してください。そうでないと、不必要なメモリリソースが残ったままになってしまいます。

☞ **注**　**アプリケーション関数は初期化VIとクローズVIを呼び出しません。アプリケーション関数を実行するには、最初に初期化VIを実行する必要があります。サンプルアプリケーションは初期化VIとクローズVIを呼び出します。**

計測器ドライバVIのヘルプを表示する

LabVIEW計測器ドライバに関する説明文書を表示するには、LabVIEWの**ヘルプ**ウィンドウを使用します。各計測器ドライバVIや各フロントパネルの制御器には**ヘルプ**の説明が用意されています。**ヘルプ**ウィンドウを表示するには、**ヘルプ**メニューから**ヘルプを表示**を選択します。VIのヘルプを表示するには、カーソルをVIアイコンの上に移動します。フロントパネル制御器のヘルプを表示するには、ヘルプと表示したい制御器の上にカーソルを移動します。**ヘルプ**ウィンドウに全体的な説明が表示されない場合は、制御器または表示器のポップアップメニューから**データ処理→説明...**を選択すると、制御器または表示器のヘルプを表示できます。

サンプルアプリケーションを対話形式で実行する（GPIBアドレス、シリアルポート、および論理アドレスを選択する）

計測器との間の通信状態をチェックしたり、プログラムによる通常の計測器動作をテストするには、まずサンプルアプリケーションを開く必要があります。各制御器をよく確認して正しく設定します。一般に最初の実行時には、アドレスフィールドを除くほとんどの制御器はデフォルトで使用してください。アドレスは正しく設定する必要があります。計測器のアドレスがわからない場合は、計測器ウィザードで調べます。VI を実行したら、妥当なデータが返されているか、エラーに関する報告がエラークラスタにないかを調べます。サンプルアプリケーションが異常終了する場合の最も多い原因を、以下に示します。

1. NI-VISAがインストールされていない。LabVIEW のインストール時にオプションとして NI-VISA を選択しなかった場合は、サンプルアプリケーションを実行する前に NI-VISA をインストールする必要があります。
2. 計測器のアドレスが間違っていた。サンプルアプリケーションでは、ユーザが正しい計測器アドレスを指定する必要があります。計測器アドレスがよくわからない場合は、計測器ウィザードか Find Resource 関数を実行します。アドレス文字列の構文がよくわからない場合は、「第 8 章　LabVIEW VISA チュートリアル」を参照してください。計測器ウィザードにアクセスするには、LabVIEW ダイアログボックスの**ソリューションウィザード**を選択します。プロンプトが表示されたら、**計測器ウィザード**を選択します。
3. ユーザが使用しているモデルが、計測器ドライバがサポートするものと正確に一致しない。使用のモデルが、計測器ドライバでサポートされているか、再チェックする必要があります。

サンプルアプリケーションを使用して通信に関する基本的な事項が確認できたら、必要に応じて計測器制御をカスタマイズすることができます。ユーザのアプリケーションに必要な内容がサンプルアプリケーションと似ている場合、カスタマイズされた VI を作成するには、**ファイル**メニューから**別名で保存...**を選択してサンプルアプリケーションのコピーを保存する方法が最も簡単です。フロントパネルのデフォルト値を変更するには、**操作**メニューから**現在のすべての設定をデフォルト設定にする**を選択します。ブロックダイアグラムの変更に、アプリケーション VI や他のサブ VI に配線された定数の変更が含まれることもあります。前述のように、サンプルアプリケーションのブロックダイアグラムは通常、初期化 VI、アプリケーション関数 VI、クローズ VI という 3 つの VI から構成されます。

対話形式でコンポーネント VI をテストする

多くのユーザは、コンポーネント VI をアプリケーションの中に入れる前に、対話形式でコンポーネント VI をテストしたいと考えます。それによって、計測器の適切な構成を選択しやすくなります。対応するフロントパネルからコンポーネント VI を実行するには、まず初期化 VI を実行する必要があります。後に続く VI では、下の図のように最初に VISA セッション制御器をポップアップして**オープンセッション**サブメニューからリソース名を選択する必要があります。

一度リソースを選択すると、リソースの選択を再設定しなくても、フロントパネルからコンポーネントVIを対話式に何回も実行することができます。

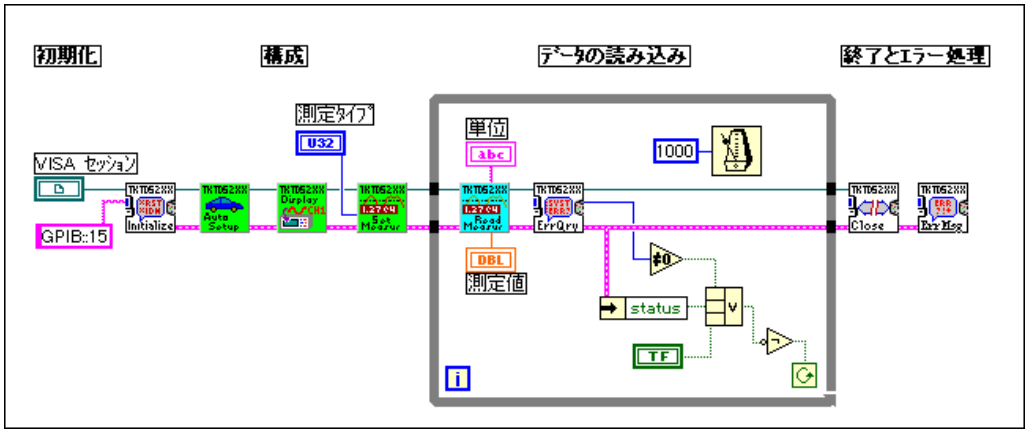
通常、アプリケーション内でコンポーネントVIを呼び出すのと同じ順序で、コンポーネントVIを実行する必要があります。最初にコンポーネントVIを実行し、その後で1つまたは複数の構成VIを実行します。トリガにより測定を行う場合は、アクションVIを呼び出してトリガの準備をする必要があります。そしてデータVIを呼び出すと測定値が収集されます。計測器ドライバのコンポーネントVIのテストが終了したら、クローズVIを実行してリソースの割り当てを解除する必要があります。

ユーザアプリケーションを作成する

必要なコンポーネントVIとそれらの実行順序の選択が終わると、アプリケーションを作成することができます。実行順序がアプリケーションVIと同様である場合は、アプリケーションVIのブロックダイアグラムを修正してもかまいません。ユーザアプリケーションがアプリケーションVIと大幅に異なる場合は、ユーザ自身のVIを作成する必要があります。

ブロックダイアグラム上にVIを順番に配置し、VISAセッションとエラークラスタパラメータを使用してVIを配線します。すべての要素のすべての入力を配線する必要はありません。ユーザアプリケーションに対してデフォルト値で十分である場合は、入力端子を配線する必要はありません。キー入力の場合は、VIを文書化する方法として一応デフォルトを配線しておく方がよいでしょう。くり返し測定を行う場合は、データ測定VIをループ内に配置する必要があります。ただし、コンポーネントVIをループ内に配置する場合は、ループに、またはループから渡されるVISAセッションワイヤとエラーワイヤに対する指標付けを不使用にする必要があることを覚えておいてください。

計測器エラーをチェックするには、エラークエリ VI を定期的に呼び出す必要があります。次の図に示すように、ユーザは、オシロスコープを使用して 1 秒に 1 回ずつ周波数測定を行い、オペレータに対して表示することができます。考えられる次の 3 つの条件によりループが終了することを確認してください。すなわち、フロントパネルの制御器を使用してオペレータが VI を停止した場合、エラークエリ VI でエラーが検出された場合、または VISA I/O インタフェースでエラーが発生した場合です。ループ内でエラーが発生した場合、Error Message VI はオペレータに対してポップアップメッセージを表示します。Error Message VI は、LabVIEW の General Error Handler VI と似ていますが、計測器に固有なより多くのエラーを報告することができます。いくつかの計測器ドライバ VI を実行した後は、エラーが発生している可能性もありますので、このようなエラーを見つけて表示できるよう Error Message VI を使用するのがよいでしょう。



関連項目

Open VISA Session Monitor VI

計測器ドライバアプリケーションを対話式に、またはプログラム的にデバッグする場合には、Open VISA Session Monitor VI が便利です。デバッグ時には、閉じる必要がある多くの VISA セッションが開いている場合があります。あまり多くの VISA セッションを開いて閉じないでおくと、使用可能なメモリリソースが少くなってしまいます。開いているすべてのセッションをすばやく閉じるには、`labview¥vi.lib¥utility¥visa.llb` ライブラリに含まれている Open VISA Session Monitor VI を実行することができます。または、作業内容を保存して終了してから、もう一度 LabVIEW を立ち上げる方法もあります。LabVIEW を終了すると開いているすべての VISA セッションが閉じられます。

エラー処理

計測器制御アプリケーションではいくつかの原因によりエラーとなることが考えられますので、エラー処理を実行することが重要です。

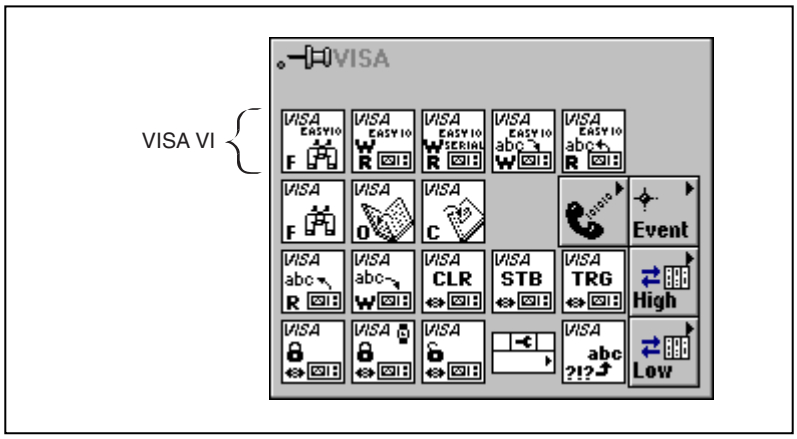
- VISAまたはその下位ソフトウェアやハードウェアが正常にインストールされていないためにVISA関数からエラーが返される場合があります。たとえばGPIB計測器と通信を行う場合は、NI-488.2が正しくインストールされていないとナショナルインスツルメンツのGPIBコントローラカードを使用できません。同様に、ボードがインストールされていない場合や正しく構成されていない場合は、計測器ドライバVIからエラーが返されます。このタイプのエラーは、Error Message VIやLabVIEWのGeneral Error Handler VIで検出することができます。
- ユーザがアクセスしているデバイスが、ユーザの送ったコマンドに応答していない場合に、VISA関数からエラーが返されることがあります。計測器のアドレスが間違っている、動作に異常がある、送られたコマンドを計測器が解釈できない、などの原因が考えられます。このタイプのエラーは、Error Message VI やLabVIEW のGeneral Error Handler VIで検出することができます。
- 計測器からエラーの報告が送られます。通常、計測器は、コマンドが無効、設定値が範囲外、ハードウェアオプションがない、などさまざまな理由のエラーを認識します。このような計測器エラーを検出するには、計測器ドライバのエラークエリVIを呼び出し、続いてError Message VIを呼び出します。

エラー処理に関する詳細は、「第8章 LabVIEW VISAチュートリアル」の「VISAによるエラー処理」の項を参照してください。

計測器との通信状態をテストする

計測器ドライバVIで計測器とうまく通信できない場合は、VISA Open VIやVISA Close VIを動作させずに、簡易VISA IO VIを使用して対話形式で簡単な読み込みと書き込みをテストしてください。たとえばEasy VISA Write VIの場合は、リソース名と計測器に送るメッセージを提供するだけで十分です。簡易VISA IO VIには次のようなものがあり、これらを次の図に示します。

- Easy VISA Find Resources
- Easy VISA Write
- Easy VISA Read
- Easy VISA Write & Read
- Easy VISA Serial Write & Read
- Easy Register Write
- Easy Register Read



これらの簡易VISA IO VIはテストのためには好都合ですが、これらをアプリケーション開発に使用する場合は注意が必要です。これらのVIは、計測器に対して読み込みや書き込みを行うたびに必ずVISAセッションを開いて閉じる動作を行いますので、このようなVIを繰り返し呼び出すとアプリケーションのスピードが低下する恐れがあります。したがって、アプリケーション開発には標準のVISA関数を使用するのが最良の方法です。

短時間で簡単にLabVIEW計測器ドライバを開発する

ナショナルインスツルメンツは新しい計測器ドライバの開発を続けていますが、必要なすべての計測器用のドライバを開発するためのリソースが当社にあるわけではありません。計測器にインタフェースする必要がありながら使用可能なドライバがない場合もあるでしょう。ここでは、ユーザのアプリケーション用の簡単な計測器ドライバを開発する方法を説明します。

既存のドライバを修正する

すべて最初から始める前に、自分の計測器に合うドライバがないかチェックします。この場合は、メーカーのホームページやナショナルインスツルメンツのホームページもチェックしてください。ホームページをチェックするときには、類似の計測器をサポートする計測器ドライバもよく調べてください。同じモデルシリーズの計測器のコマンドセットは、よく似ている場合が多くあります。同様に、機能が似ているSCPI計測器の場合も、コマンドセットが似ています。このようなドライバを入手して、自分の計測器とコマンドセットが似ているかチェックします。同じモデルシリーズの計測器の場合は、メーカーに問い合わせてコマンドセットの相違点の詳細を調べる必要があるかもしれません。類似するSCPI計測器を比較する場

合は、計測器ドライバのコマンドを、計測器のプログラミングマニュアルに記載されているコマンドと比較する必要にがあります。

既存のドライバを修正してコードを最適化したい場合があります。さまざまなユーザに使用されるドライバの場合は、コンポーネントVIにより、自分のアプリケーションには不必要な設定が変更されてしまう可能性もあります。通常、最適化する必要があるのは、ループ内で繰り返し呼び出されるVIだけです。構成VIは通常一度しか呼び出されませんので、アプリケーションの速度にはほとんど影響を与えません。

計測器ドライバVIを修正する際に最も簡単なのは、**ファイル**メニューから**別名で保存** ... を選択して名前を変更する方法です。新しいVIを識別できるようにするには、接頭辞か記述部を修正して名前を変更します。たとえば、Tektronix TDSオシロスコープの計測器ドライバを別の計測器で動作するように修正する場合は、VIの接頭辞をTKTDS7XXから自分の計測器に合った名前に変更します。名前を修正すれば、ブロックダイアグラムとフロントパネル制御器を修正しても構いません。ブロックダイアグラムに対するほとんどの変更は、文字列関数に関係があります。Pick Line and AppendやSelect and Appendなどの文字列関数についてよく知らない場合は、『LabVIEWオンラインリファレンス』で詳細を調べてください。

各初期化VIには、計測器のモデルまたはモデルシリーズに固有な識別クエリを呼び出すオプションがあります。このオプションをOFFにするか、あるいは識別クエリコマンドに対する応答を変更する必要があります。SCPI計測器の場合、このコマンドは*IDN?になります。

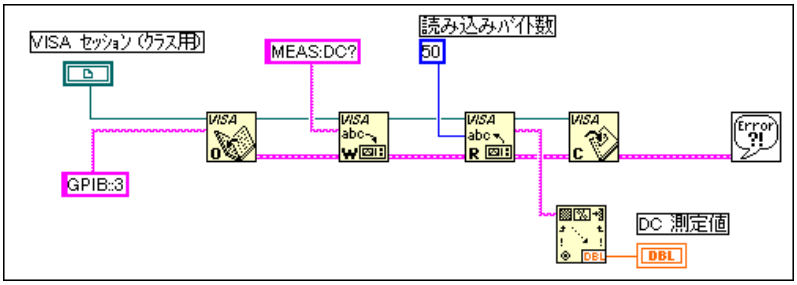
計測器ドライバをどの程度変更する必要があるかは、計測器とそれらのコマンドセットがどの程度似ているかによって決まります。コマンドセットが大幅に異なる場合は、一から始める方がよいでしょう。

簡易ドライバを開発する

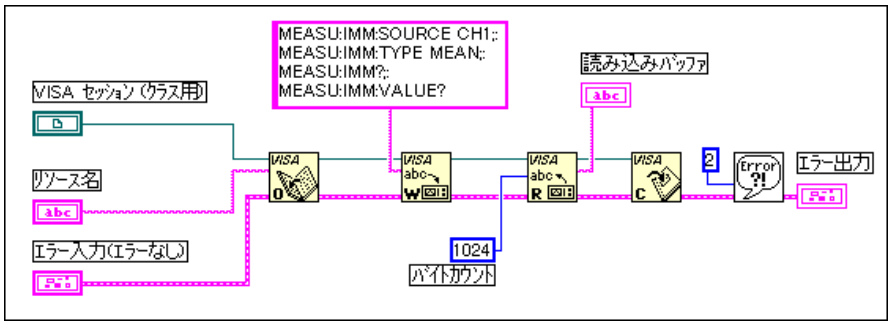
メッセージベースの計測器はほとんど、計測器に対する一連の書き込みと読み込みによってプログラム的に制御されます。簡易ドライバのほとんどで必要となるVISA関数は、VISA Open、VISA Write、VISA Read、VISA Closeの4つだけです。

次の図に示す簡易計測器ドライバVIは、計測器に対して書き込みと読み込みを1回しか行いません。このVIは、最初にVISA Open VIを使用して計測器用のリソースを開きます。次にこのVIは、（計測器のユーザマニュアルに説明されている）MEAS:DC?コマンドを送って計測器からDC測定値を返すようにします。VISA Read関数は、測定値を文字列の形で返します。この測定値を他の数値関数で使用するため、Scan from String関数を使用して文字列が数値に変換されます。計測器に対する最後の読み込みまたは書き込みが完了すると、VISA Close関数が呼び出されます。この後、計測器

IO 関数で発生する可能性のあるエラーに対処するための Simple Error Handler VI を呼び出します。

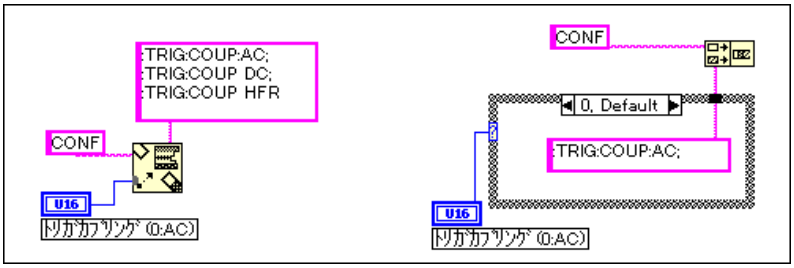


さらにモジュール式の計測器ドライバにするため、実現したい動作のタイプに従って、計測器に対する読み込みと書き込みをさらに分割してみることもできます。また、構成を設定するのに必要な読み込みと書き込みを組み合わせて 1 つの VI にし、測定値の読み込みは別の VI にすることもできます。繰り返し測定を行うため測定 VI をループ内に入れることもできます。自分の設定の構成が正確にわかっている場合は、次の図のように、すべての構成コマンドを 1 つの文字列定数に含めることもできます。



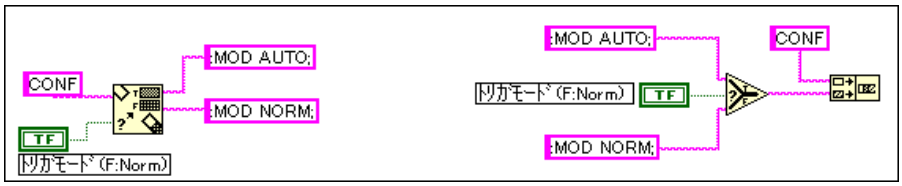
また、ユーザが別の構成を選択できるようにしたい場合は、コマンド文字列をプログラム的に作成する必要があります。Pick Line and Append 関数を使用すると、1 回の操作で選択肢から文字列を選択して他の文字列に連結することができます。この手順は、Case ストラクチャと Concatenate Strings 関数を使用するよりも簡単です。

次の図の左側のブロックダイアグラムは、右側のブロックダイアグラムよりも簡単です。

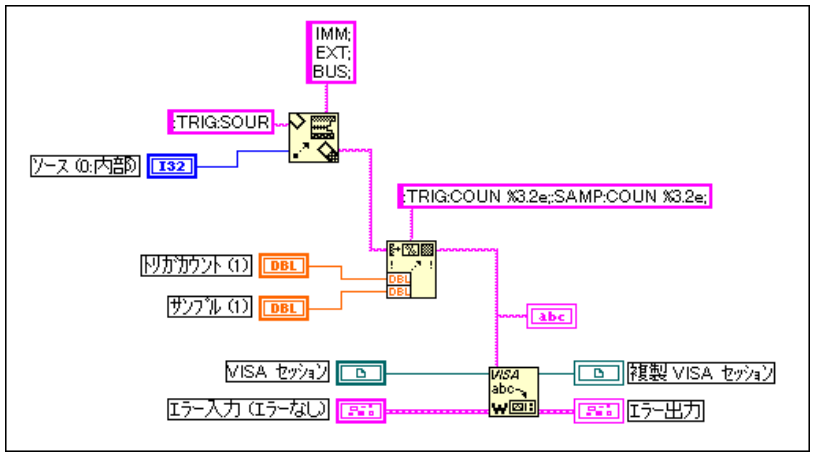


Select and Append 関数を使用すると、1 回の操作で文字列定数を選択して他の文字列に連結することができます。この手順は、Select 関数の後に Concatenate 関数を使用する方法よりも簡単です。

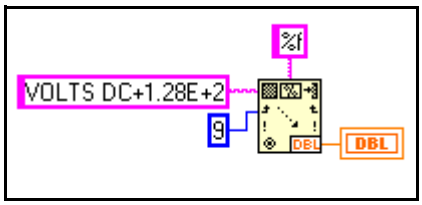
次の図の左側のブロックダイアグラムは、右側のブロックダイアグラムよりも簡単です。



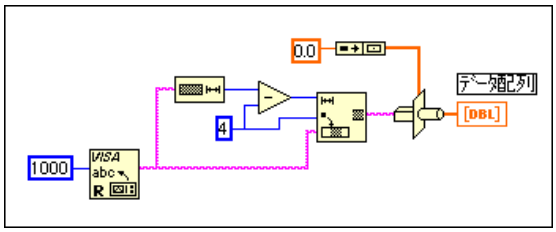
コマンド文字列の作成に役立つその他の文字列関数としては、Format into String 関数と Format and Append 関数があります。これらの関数は、さまざまな形式処理オプションを使用して 1 つまたは複数の数字を文字列に変換します。次の図のブロックダイアグラムに、Pick Line & Append 関数と Format into String 関数の両方を示します。



計測器からの応答を読み込んだ後、測定値を解析して数値にする必要があります。Scan from String 関数は、ASCII 数字を数値に変換するのに役立ちます。次のコードは、文字列入力の "VOLTS DC" という部分を取り除いて "+1.28E+2" を倍精度の数値に変換します。マルチメータからの応答は通常、文字列入力となります。



計測器から返ってくるのがバイナリデータである場合は、Type Cast関数を使用します。この関数は、ワイヤのデータタイプを変更しますが、メモリデータの格納方法は変更しません。VISA Readの場合、返されるデータは、ASCIIまたはバイナリのいずれにコード化されているにしても、文字列です。したがって、バイナリ文字列を数値または数値配列に変換するために、文字列を別のタイプにキャストする必要があります。次の例は、1,000バイトの応答文字列から4バイトのヘッダを取り除き、残った値を単精度配列に変換します。



多機能ドライバを開発する

他の人が使用するためのドライバを開発している場合、多機能ドライバを開発したいと考えることもあるでしょう。このようなドライバは、よりモジュール化されていて、ナショナルインスツルメンツの計測器ドライバライブラリと似たアーキテクチャを持ち、エラー報告関数やユーティリティ関数も装備しています。より多くの詳細機能を持つドライバの作成については、ナショナルインスツルメンツのホームページの、『Application Note 006、Developing a LabVIEW Instrument Driver』を参照してください。

IVI計測器ドライバとともにLabVIEWを使用する

600個を超えるLabVIEWソースコードドライバの他に、IVI（Intelligent Virtual Instruments：インテリジェントバーチャルインスツルメンツ）ドライバを使用して計測器を制御することもできます。IVI計測器ドライバはLabWindows/CVIで開発されたDLLベースのドライバであり、製造テストのユーザは、さらに次のようなメリットを得ることができます。

- 計測器ステータスのキャッシングを行うことにより性能が改善される
- シミュレーション
- マルチスレッドによる安全性
- 計測器属性へのアクセス

IVI計測器ドライバの使用方法に関する詳細は、『LabVIEW オンラインリファレンス』を参照してください。

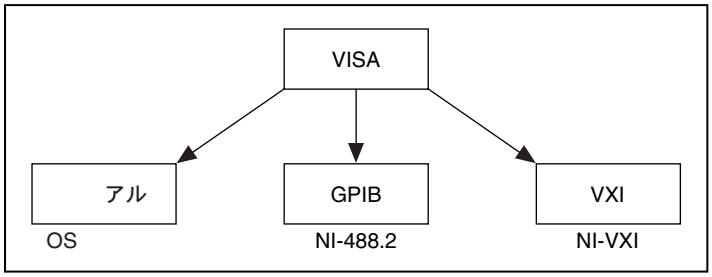
8

LabVIEW VISA チュートリアル

この章では、LabVIEW における VISA の仕様について概要を示します。また、VISA による計測器のプログラミングに関連する基本概念を説明し、例を用いて VISA の概念を簡単に説明します。

VISA とは？

VISA とは、計測用プログラムを作成するための標準的な I/O アプリケーションプログラムインタフェース（API）です。VISA 自身には計測用プログラムを作成する機能はありません。VISA は、下位ドライバを呼び出すための上級 API です。NI-VISA の階層構造を、次の図に示します。



VISA は、VXI、GPIB、またはシリアル計測器を制御できますので、使用する計測器のタイプに合わせて適切なドライバを呼び出せます。VISA に関する問題をデバッグする際には、この階層が存在することを念頭に置くことが重要です。VISA の問題のように見えても、実際には、VISA が呼び出しているドライバに異常がある場合もあります。

サポートされるプラットフォームと環境

VISA は、計測器ドライバを開発するための業界標準となっており、現在ナショナルインスツルメンツが作成している計測器ドライバのほとんどが VISA を使用しています。このためプラットフォームでシステムレベルのドライバを使用できる場合は、Macintosh、Windows *3.x*、Windows 95、Windows NT、Solaris 1、Solaris 2、および HP-UX をサポートします。

VISAを使用する理由

VISAは標準となっている

VISAは、計測業界における計測器ドライバ用の標準APIとなっています。また、1つのAPIを使用して、VXI、GPIB、シリアルなど、異なるタイプの計測器を制御することができます。

インタフェースの独立性

VISAは、インタフェースのタイプに関係なく同じ操作で計測器との通信を行います。たとえば、メッセージベースの計測器にASCII文字列を書き込むVISAコマンドは、計測器がシリアル、GPIB、VXIのいずれの場合でも同じです。このように、VISAではインタフェースの独立性が得られますのでバスインタフェースの切り替えが容易になり、したがってさまざまな異なるインタフェースに合わせて計測器をプログラムしなければならないユーザでも、1つのAPIを学習するだけで済みます。

プラットフォームの独立性

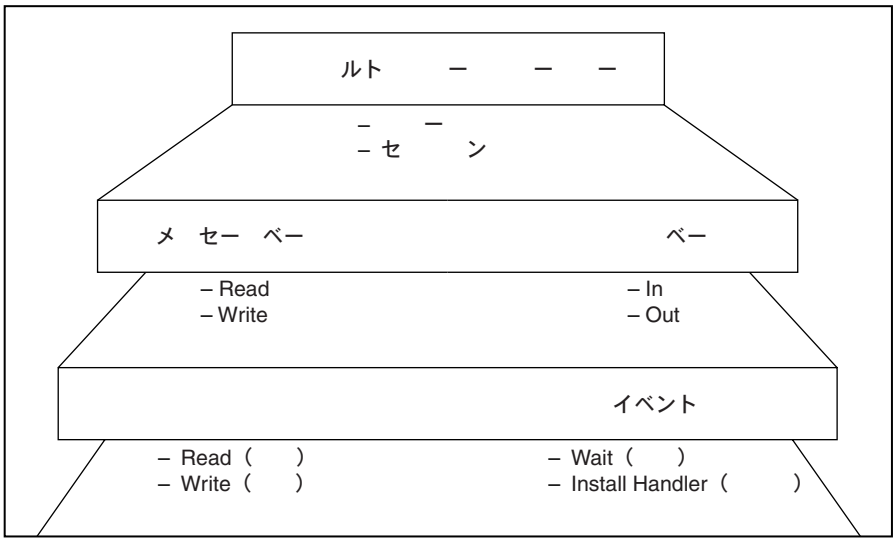
VISAは、VISA関数呼び出しを使用して記述されたプログラムを、あるプラットフォームから別のプラットフォームに容易に移植できるように作られています。プラットフォームの独立を確保するため、VISAでは独自のデータタイプが厳密に定義されています。したがって、プラットフォーム間での整数変数のバイト単位のサイズなどの問題が、VISAプログラムに影響を与えることはありません。VISA関数呼び出しと関連パラメータは、すべてのプラットフォームで同じです。ソフトウェアを、別のプラットフォームに移植してからコンパイルし直すことができます。LabVIEWプログラムは、LabVIEWをサポートするどのプラットフォームにも移植することができます。

将来への対応が容易

VISAのもう1つの利点としては、VISAがオブジェクト指向APIであり、将来新しい計測用インタフェースを開発したときそれに容易に対応できるため、新しいインタフェースに対するアプリケーションの移植性が高いことにあります。

VISAの基本概念

VISA APIの内部構造の簡単な概要を、次の図に示します。



デフォルトリソースマネージャ、セッション、および計測器デスクリプタについて

デフォルトリソースマネージャは、VISA 動作の最上位に位置します。LabVIEWは、最初のVISA VIの呼び出しで、デフォルトのリソースマネージャとの通信を自動的に確立します。ここでは、リソースとセッションという2つの用語を定義する必要があります。

リソース ― 通信を可能にする識別子です。たとえば、計測器（INSTR）、メモリアクセス（MEMACC）などのリソースがあります。

セッション ― 既存のいずれかのリソースに対する接続（リンク）。デフォルトリソースマネージャもこれに含まれます。

他のリソースに対するセッションを開くには、VISA のデフォルトリソースマネージャを使用します。計測器に対するセッションを開いてからでないと、VIはその計測器と通信できません。

また VISA のデフォルトリソースマネージャは、システム内で使用可能なリソースを検索することもできます。検索後、これらのリソースのいずれかに対するセッションを開くことができます。

リソースの検索方法

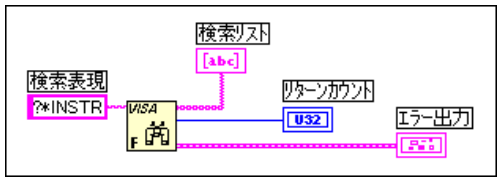
次の図に示すVISAのFind Resource関数は、システム内の使用可能なリソースを検索します。一般的にVISAプログラムでは、この関数が最初に使用されます。この関数を使用すると、アプリケーションを実行するのに必要なリソースがすべて使用可能かどうかを判断できます。

VISA Find Resource関数に必要な唯一の入力は、**表現**と呼ばれる文字列です。Find Resource VIが返すリソースのタイプは、この文字列によって決まります。**表現**に使用できる文字列を次の表に示します。これらは、『LabVIEWオンラインリファレンス』にも掲載されています。

計測器リソース	表現
GPIB	GPIB[0 - 9]*::? *INSTR
GPIB-VXI	GPIB-VXI?*INSTR
GPIBまたはGPIB-VXI	GPIB?*INSTR
VXI	VXI?*INSTR
すべてのVXI	?*VXI[0 - 9]*::?*INSTR
serial	ASRL[0 - 9]*::?*INSTR
すべて	?*INSTR

メモリアクセスリソース	表現
VXI	VXI?*MEMACC
GPIB-VXI	GPIB-VXI?*MEMACC
すべてのVXI	?*VXI[0-9]*::?*MEMACC
すべて	?*MEMACC

関数の戻り値としては、**リターンカウント**（見つかったリソースの数を報告する）と**検索リスト**があります。**検索リスト**は文字列の配列です。各文字列には、見つかったリソースについての記述が含まれています。これらの文字列は計測器デスクリプタと呼ばれます。システム内で使用可能なすべてのリソースを見つけるVIを、次の図に示します。



計測器デスクリプタ — VISAリソースの正確な名称と位置。この文字列の形式は次のようになります。

インタフェースタイプボードの指標::アドレス::VISAクラス

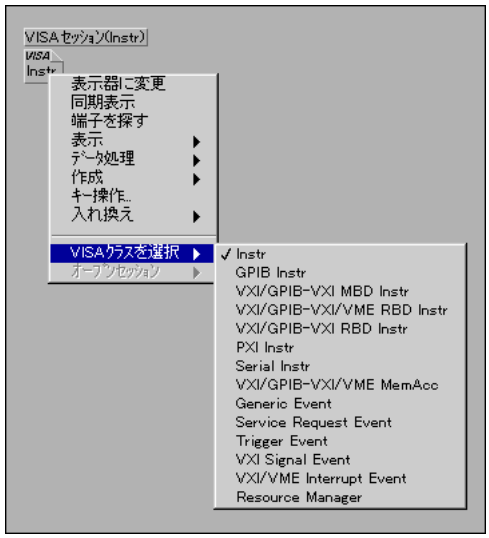
計測器デスクリプタは、検索クエリで見つかった特定の計測器のことです。ボードの指標を使用する必要があるのは、システム内に複数のインタフェースタイプが存在する場合のみです。たとえば、システムに2つのGPIBプラグインボードがある場合、一方をGPIB0、もう一方をGPIB1という名前で参照することができます。この場合は、計測器デスクリプタの中でボードの指標を使用する必要があります。アドレスパラメータは、VXI計測器の場合は計測器の論理アドレスになり、GPIB計測器の場合はGPIBの基本アドレスになります。シリアル計測器では、アドレスパラメータを使用しません。たとえばASRL1::INSTRは、パーソナルコンピュータのCOM 1シリアルポートのディスクリプタです。

VISAクラスとは？

VISAクラスは、VISA動作の一部またはすべてを含む分類です。INSTRは、計測器に対するすべてのVISA動作を含む最も一般的なクラスです。将来別のクラスをVISA仕様に追加することもできます。現時点では、VISAクラスを計測器デスクリプタに含める必要はありませんが、将来的な互換性を確保するためには含めておくべきです。現在VISAクラスが空白のままになっている場合は、デフォルトのINSTRクラスになります。

VISA制御器をポップアップする

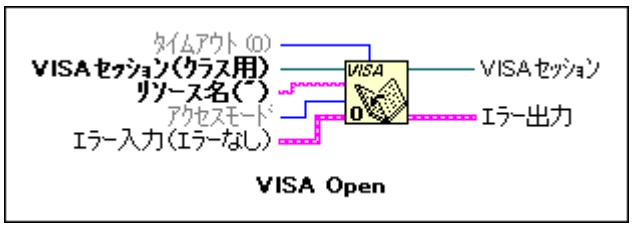
LabVIEWには、現在使用可能なVISAセッションに対してクラスを設定する方法が他にも用意されています。次の図のように、フロントパネルのVISAセッション制御器をポップアップしてVISAクラスを選択することができます。



デフォルトのInstrクラス以外のクラスを選択した場合、このセッションに正常に配線できるのは、そのデバイスクラスに対応する動作を行うための関数だけです。たとえば、VISAクラスとしてGPIB Instrが選択されている場合に、VXIレジスタアクセス用の関数をセッションに配線することはできません。

セッションを開く

システム内のリソースに対してセッションを開くには、計測器デスクリプタが使用されます。VISA Open関数を、次の図に示します。

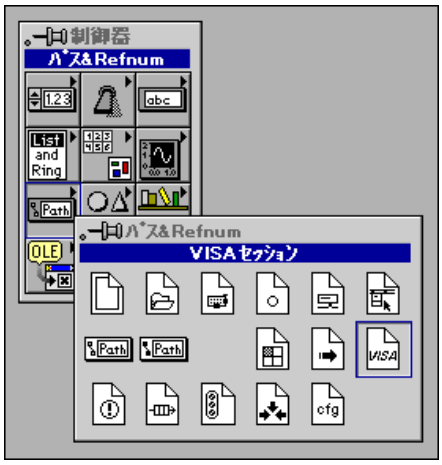


リソース名入力は、セッションが開かれるリソースの VISA 計測器デスクリプタです。

☞ Find Resource VI を使用しなくても、リソースの計測器デスクリプタを取得できます。ユーザやプログラマが指定した計測器デスクリプタとともに VISA Open VI を使用することができますが、デスクリプタの構文を確実なものにするには、Find Resource を最初に実行するのが最も良い方法です。

☞ **注　ほとんどのアプリケーションでは、アプリケーションが通信を行う相手となる各計測器に対して一度セッションを開くだけで十分です。アプリケーション全体にわたってこのセッションを使用し、アプリケーションの終了時に閉じることができます。**

VISA Open VIに対するVISAセッション入力もあることに注意してください。LabVIEW 内のリソースに対してセッションを開くには、フロントパネルの VISA セッション制御器が必要です。フロントパネルの VISA セッション制御器は、**制御器**パレットの**パス &Refnum** サブパレットの中にあります。



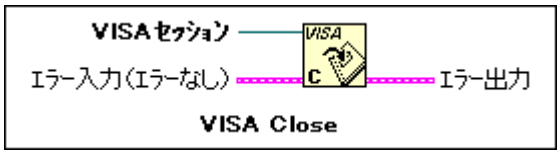
デフォルトリソースマネージャ、計測器デスクリプタ、およびセッションの間の関連性

ここで、デフォルトリソースマネージャ、計測器デスクリプタ、およびセッションについてはっきり理解しておくことが重要です。VISA のデフォルトリソースマネージャは電話のオペレータによく似ています。デフォルトリソースマネージャへのセッションを開くこと（この処理は LabVIEW 内で自動的に行われることを思い出してください）は、電話器を取り上げてオペレータを呼び出し、プログラムと VISA ドライバとの間の通信回線を確立するのと似ています。

次に電話のオペレータは、電話番号をダイヤルしてシステム内のリソースとの間の通信回線を確立します。リソースマネージャが使用する電話番号が、計測器デスクリプタに当たります。通信回線は、VISA リソースに対して開かれるセッションです。またリソースマネージャは、使用可能なすべての電話番号を探すことができます。これが、VISA Find Resource の操作です。

セッションを閉じる

VISA リソースに対して開かれたセッションもまた、コンピュータ内のシステムリソースを使用します。VISA プログラムを正常に終了するには、開いているすべての VISA セッションを閉じる必要があります。このために、次の図のような VISA Close VI が用意されています。



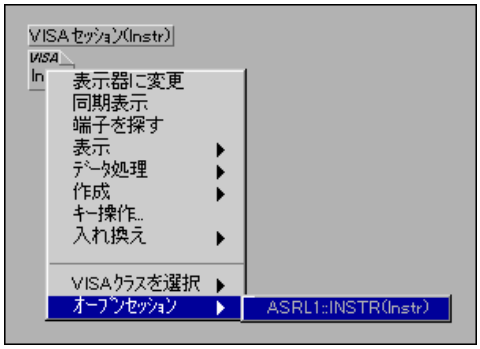
VISA Close VI に対する VISA セッション入力が閉じられるセッションです。このセッションは、もともとは VISA Open VI の出力セッション端子から来ています。

VI を実行したときにセッションを閉じないと、そのセッションは開かれたままになります。開かれたセッションの数が増えてシステムリソース上の問題が発生しないように、アプリケーション内で開いているセッションを閉じるのが良いでしょう。ただし、セッションを開いたままにしておく方が便利な場合もあります。

☞ **注**　**VI を中止した場合（たとえば VI のデバッグ中など）、VISA セッションは自動的には閉じられません。このようなセッションを閉じるためには、Open VISA Session Monitor VI（`vi.lib¥Utility` の中にあります）が役に立ちます。**

セッションを開いたままにしておく方が良い場合

VI を実行し、セッションを閉じずに開いたままにしておくと、そのセッションを以後の VI で使用できます。開いているセッションにアクセスするには、フロントパネルの VISA セッション制御器をポップアップして**オープンセッション**を選択します。そうすると、フロントパネルの VISA セッション制御器の出力が、選択された開かれたセッションとなります。このようにすると、前の VI の実行時に開かれたまま残されたセッションを閉じることができます。また、この方法を使用すると、アプリケーションの一部分を対話式にテストすることもできます。開いているセッションを選択する例を、次の図に示します。



VISA セッション制御器を使用すると開いているセッションを確かめることができます。フロントパネルの VISA セッション制御器をポップアップして開いているセッションにアクセスする方法も、計測器ドライバの一部を対話式に実行するのに便利です。

VISA によるエラー処理

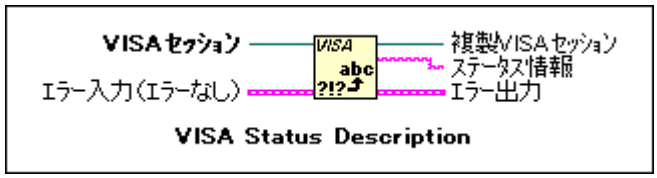
VISA VI によるエラー処理は、LabVIEW における他の I/O VI によるエラー処理に似ています。それぞれの VISA VI にはエラー入力端子とエラー出力端子があり、エラークラスタをダイアグラム内のある VI から別の VI に渡すのに使用されます。エラークラスタには、エラーが発生したかどうかを示すブールフラグ、数値による VISA エラーコード、およびエラーが発生した VI の位置を格納する文字列が含まれています。エラーが発生すると、以降の VI は実行せずそのままエラークラスタを渡します。フロントパネルのエラークラスタ表示器に VISA VI のエラー出力端子からの出力が表示された様子を、次の図に示します。

この場合はエラーが発生していることに注意してください。また、コード表示器では VISA エラーコードが途切れていることにも注意してください。VISA エラーコードは通常は 16 進数形式の 32 ビット整数です。LabVIEW のエラークラスタには、コードが 10 進数で表示されます。『LabVIEW オンラインリファレンス』の中の「VISA エラーコード」の項にも、10 進数のエラーコード一覧が掲載されています。ただし、上図に

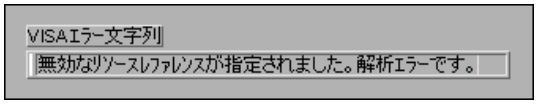
示す通り、これらのエラーコードはエラークラスタ内では途切れています。コード表示器のサイズを変更しないと、エラーコード全体が表示されません。

LabVIEW の Simple Error Handler VI と General Error Handler VI は、**関数**パレットの**時間＆ダイアログ**サブパレットの中にあります。エラーが発生した場合、これらの VI は、考えられるエラー原因を示すポップアップダイアログボックスを表示します。Simple Error Handler VI は、前の例で使用されたエラークラスタと同じエラーを返しますが、次の図のように、エラーに関するより詳細な情報が得られます。

考えられる原因の下にコードが表示されることに注意してください。LabVIEW のエラー処理 VI でポップアップダイアログボックスによりエラーを処理する方法は、常に便利なわけではありません。VISA は、VISA エラーコードを取り込んでそのコードに対応するエラーメッセージ文字列を出力として生成する動作も行えます。このような VI を次の図に示します。



この VI に対する入力は VISA セッションと VISA エラークラスタです。VI は入力のエラークラスタ内の VISA コードをチェックして、コードをテキストにより記述したものを**ステータス情報**に出力します。VISA Status Description VI から返されたエラー文字列が表示された LabVIEW の文字列表示器を、次の図に示します。



エラー処理を実行するためにどの方法を使用するか、プログラムの性質によって異なりますが、ある種のエラー処理メカニズムは、VISA を含むどのプログラムにも組み込む必要があります。

簡易 VISA VI

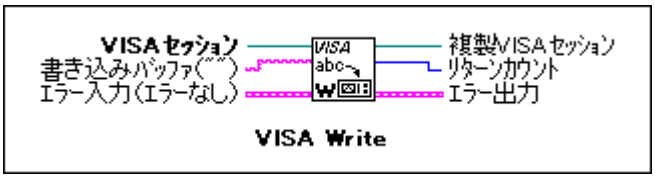
簡易 VISA VI を使用すると、計測器との間の通信が確立されたことを確認することができます。アプリケーションを開発する場合は、パレット内の別の VISA VI を使用する必要があります。その方が計測器をより高度に制御できます。簡易 VISA VI に関する詳細は、「第 7 章　LabVIEW 計測器ドライバ入門」の「計測器との通信状態をテストする」の項を参照してください。以下の項の例では、簡易 VISA VI を使用しておりません。

メッセージベースの通信

シリアル、GPIB、および多くの VXI デバイスは、さまざまなメッセージベースのコマンド文字列を認識します。VISA レベルでは、計測器に対してコマンド文字列を送るのに使用される実際のプロトコルを知らなくても操作することができます。ユーザ側では、メッセージベースのデバイスとの間でメッセージの書き込みや読み込みを行いたいということさえわかっていれば十分です。このような動作を行うために使用される VI が、VISA Write と VISA Read です。

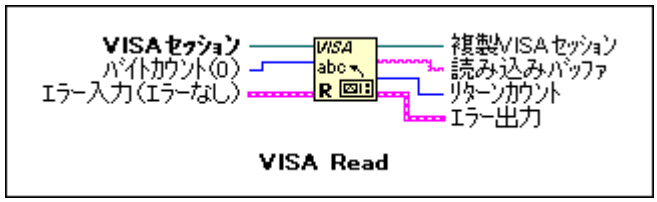
☞ **注**　**GPIB、シリアル、およびメッセージベースの VXI 計測器のどれに対してもメッセージベースのコマンドを書き込む場合の同じ VI が使用されます。VISA は、使用されるリソースのタイプから、呼び出すべき関数を自動的に判断します。**

VISA Write VI を、次の図に示します。



セッション以外の唯一の入力は、計測器に送られる文字列です。

VISA Read VI も同様に容易に使用できます。VISA Read VI を次の図に示します。

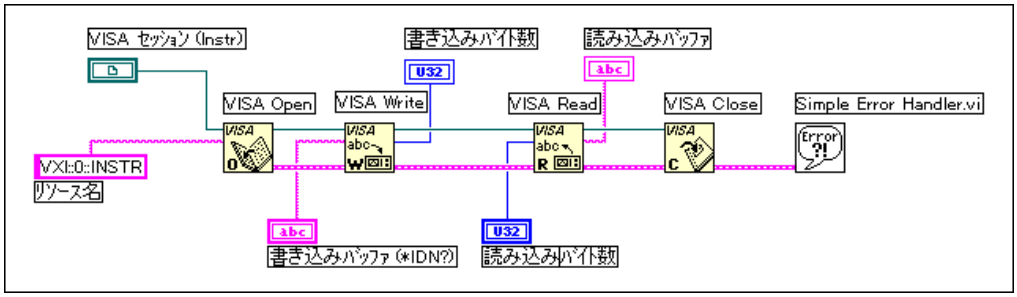


VISA Read VIには、計測器から読み込むべき最大バイト数に等しいバイトカウント入力を与える必要があります。このバイト数を読み込んだ場合、または転送の終了が指示された場合に、VIは読み込みを停止します。

実際に計測器が認識するメッセージベースのコマンドは、メーカーによって異なります。メッセージベースの計測器に対するコマンドは、IEEE 488.2とSCPIで標準化されており、多くの計測器はこれらの規格に従っていますが、個々の計測器に適合するコマンドを確実に調べるには、メーカーが提供している資料を参照するしか方法がありません。ただし、多くのメッセージベースのデバイス用の計測器ドライバが存在し、これらの計測器ドライバには、適切なASCIIコマンド文字列をまとめて一緒に計測器に送る関数が含まれています。最新のドライバを入手するには、ナショナルインスツルメンツのホームページやftpサイトをご覧ください。

メッセージベースのデバイスに対する書き込みと読み込みの方法

メッセージベースの計測器に*IDN?（識別）文字列を書き込んで応答を読み込む簡単な例を、次の図に示します。

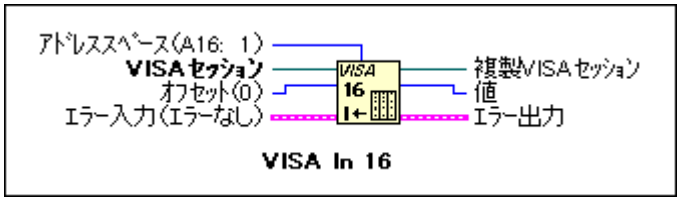


このプログラムは*IDN?コマンドを認識するすべてのデバイスに対して正常に使用できます。デバイスは、シリアル、GPIB、またはメッセージベースのVXIが可能で、唯一異なる点は計測器デスクリプタだけです。

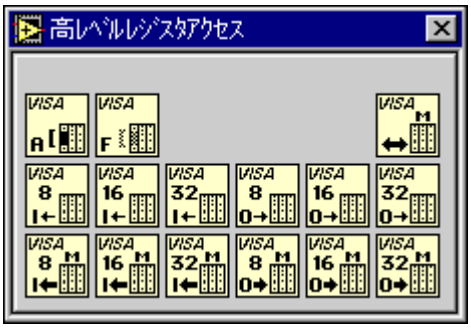
レジスタベースの通信（VXIのみ）

VISAには、VXI計測器に使用するための一連のレジスタアクセスVIが含まれています。GPIBデバイスまたはシリアルデバイスしか使用しない場合、この項の内容は適用されません。

VXI計測器によっては、メッセージベースのコマンドをサポートしないものがあります。このような計測器と通信を行う唯一の方法が、レジスタアクセスによる方法です。すべてのVXI計測器は、A16メモリ領域の上位16キロバイトに構成レジスタを持っています。したがって、レジスタアクセス関数を使用して、メッセージベースのデバイス用の構成レジスタに対して書き込みや読み込みを行うこともできます。レジスタからの値の読み込みに使用される基本的なVISA動作が、VISA Inです。この動作には、8、16、32ビットの値を読み込むための3つの異なるバージョンがあります。計測器がサポートしているアクセス幅に合ったVIを使用する必要があります。たとえば、ある計測器が16ビットアクセス用に設計されている場合に32ビットアクセスを行えば、バスエラーが返ってくることが考えられます。VISA In 16 VIを次の図に示します。



このVIを始めとする基本レジスタアクセスVIは、VISA関数パレットの**高レベルレジスタアクセス**サブパレットの中にあります。

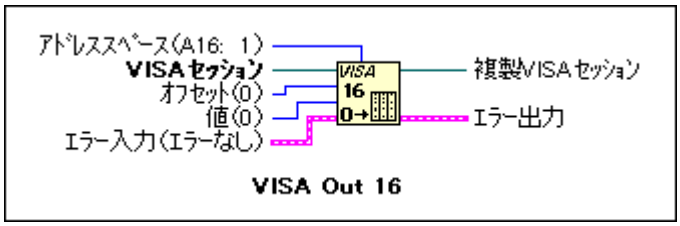


アドレススペースの入力は、使用するVXIアドレスの領域を示します。オフセット入力は、混乱の原因となることがあります。VISAは、各アドレス領域内でデバイスが要求するベースメモリアドレスを記憶していること

を覚えておいてください。オフセット入力は、このベースアドレスに対する相対値です。

次の例を考えてみてください。論理アドレス1にデバイスがあり、VISA In 16 VIを使用してこのデバイスのID／論理アドレス構成レジスタを読み込みたいと仮定します。このレジスタがA16領域の絶対アドレス0xC040にあり、論理アドレス1のデバイス用の構成レジスタが0xC040～0xC07Fの間にあることがわかっています。一方、VISAもこれを認識していますので、ユーザが指定する必要があるのは、この領域内でユーザがアクセスしたい部分のオフセットだけです。この場合、そのオフセットは0です。

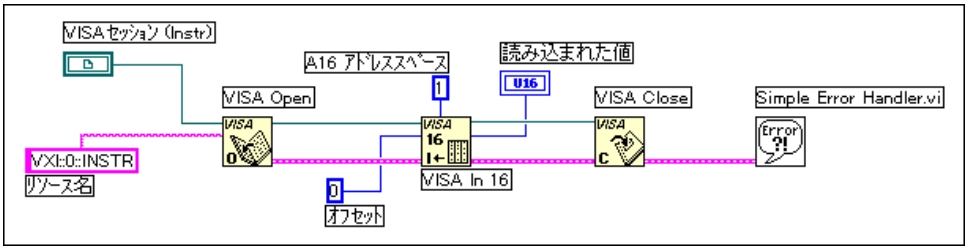
VISA In操作と類似した上級レジスタアクセス操作があります。これらの操作はVISA Out操作といい、レジスタの書き込みを行います。VISA Out 16 VIを次の図に示します。



このVIはVISA In 16 VIと似ていますが、書き込む値を端子に与えなければならない点が違います。ただし、VISA Out VIを使用する場合、レジスタによっては書き込みサイクルに応答しないものやバスエラーの原因となるものがあることに注意してください。

基本レジスタアクセス

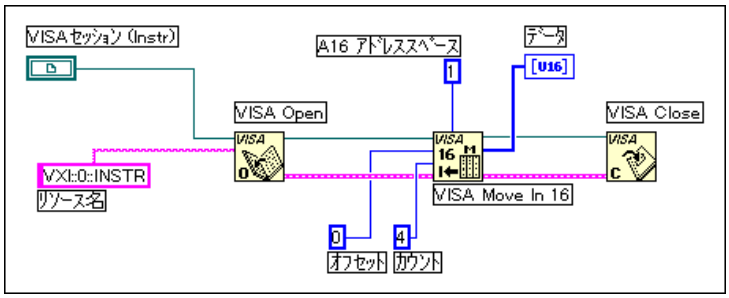
VIの中の上位VISAアクセス関数の例を、次の簡単なプログラムで示します。



このブロックダイアグラムは、論理アドレス0の位置にあるVXIデバイス用の最初の構成レジスタを、VISA In 16 VIを使用して読み込む方法を示します。オフセットパラメータは、アクセスされるVXIアドレス領域内でデバイスが要求するメモリ範囲に対する相対値であり、この場合は0となります。アドレス領域パラメータは、アクセスされるVXIアドレス領域を示

します。この場合デバイスは論理アドレス 0 の位置にあります。デバイスの構成レジスタは、アドレス領域 A16 の 0xC000 から 0xC03F の間に配置されています。オフセットを 0 にした場合に VISA In 16 が実際にレジスタを読み込む位置は 0xC000 です。

このプログラムは、指定されたリソース（VXI::0::INSTR）の指定されたオフセット（0）の位置にある、A16 アドレス領域の 16 ビットレジスタから読み込みを行います。このブロックダイアグラム内で実行される一連の VISA VI でエラーが発生した場合は、Simple Error Handler がダイアログボックスを返してユーザにエラーを知らせ、VISA エラーコードに対応するテキストメッセージを表示します。基本レジスタ転送
次のブロックダイアグラムは、論理アドレス 0 の位置にある VXI デバイスの最初の 4 つの構成レジスタを、VISA Move In 16 VI を使用して読み込む方法を示します。



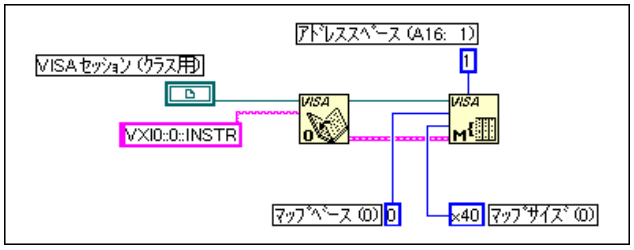
一連の VISA Move In VI は、VXI デバイスから大きいデータブロックを読み込む場合に使用されます。データは、4 つの 16 ビット値の配列として返されます。大きいデータブロックを VXI デバイスに転送するには、これに対応する一連の VISA Move Out VI があります。Move In VI と Move Out VI には 8、16、32 ビットのバージョンがあります。どの VI が適するかは、アクセスするレジスタのサイズによって決まります。

下位アクセス関数

下位アクセス（Low-Level Access: LLA）関数は、レジスタベースの通信を行うのに非常に効率的な方法を提供します。ある種のアクセスに対しては、LLA 関数のオーバーヘッドの負担は上位アクセス（High-Level Access: HLA ）関数に比べてはるかに少なくなります。LLA 関数は HLA 関数と同じ手順で実行しますが、HLA 関数が実行する個々のタスクは LLA の下で行う個々の関数である点が違います。

VISAを使用して下位レジスタアクセスを実行する

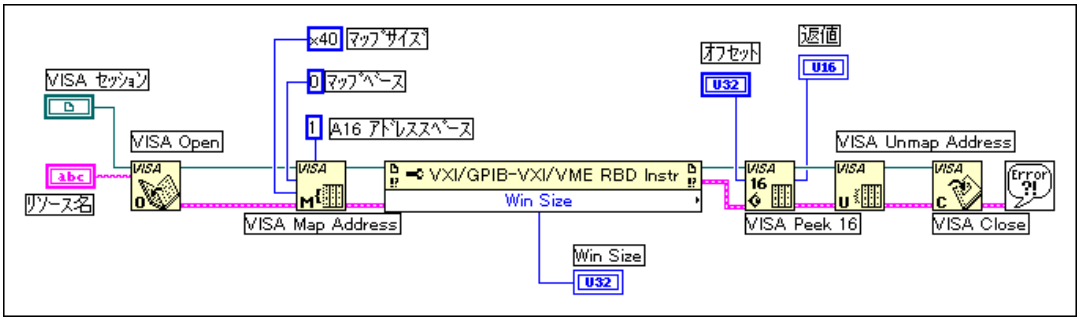
デバイスレジスタにアクセスするために最初に呼び出す必要がある LLA 操作は、VXI アドレス領域にアクセスできるようハードウェアウィンドウを設定するための[VISA Map Address] 操作です。[VISA Map Address」操作は、前項で説明したようにローカル CPU アドレスを VXI アドレスにマップできるようにハードウェアをプログラムします。A16 アドレス領域にアクセスするようにハードウェアをプログラムするコードの例を、次に示します。



このサンプルコードは、A16 領域がオフセット 0 から始まる 0x40 バイトに対応するようにハードウェアを設定します。オフセットは、[VISA] セッションにより通信を行う相手のデバイスのベースアドレスに対する相対値であり、A16 領域そのもののベースからではないことを忘れないでください。したがって、オフセット 0 は A16 領域のアドレス 0 ではなく、デバイスの A16 メモリの始点です。

☞ **注**　**MEMACC セッションによりデバイスレジスタにアクセスするには、VXIbus の絶対アドレス（デバイスのベースアドレス＋デバイスのアドレス領域内のレジスタのオフセット）を指定する必要があります。**

現在のところ、VISA では 1 つのセッションにつき 1 つのマップしかサポートしておりませんので、1 つのデバイスに対して複数のマップが必要な場合は、そのデバイスに対する第 2 のセッションを開く必要があります。2 つのセッションがあっても、セッション自身は多くのメモリを必要としませんので、オーバーヘッドはきわめて少なくて済みます。ただし、2 つのセッションのハンドルを記憶しておく必要があります。また、これはシステム上で可能な最大ウィンドウ数とは別であることに注意してください。コントローラ用に使用しているハードウェアには、サポート可能な単一のウィンドウの数に制限があります。ウィンドウでの操作を終了した場合や、マップを別のアドレスまたはアドレス領域に変更する必要がある場合は、まず [VISA Unmap Address] 操作を使用してウィンドウのマップを解除する必要があります。



この例は、64（hex 40）バイトを、論理アドレス 1 のデバイスのアドレスからのオフセットが 0 の位置で始まる A16 アドレス領域にマップします。LLA 操作を使用する場合は必ず、アクセスするアドレス範囲に対応できるサイズのマップを指定してください。このためには、プロパティーノードを使用して、デバイスが使用するアドレス領域の大きさを決定します。プロパティーノードについては、この章の後の部分で説明します。

☞ **注**　**マップが可能なデフォルトの最大ウィンドウは、通常64kBです。MITEベースのコントローラを使用している場合は64kB以上を要求できますが、ユーザウィンドウサイズを大きくする必要があります。この操作は、各コントローラのリソースエディタ、すなわちT&M Explorer、VXIEdit、またはVXItediのいずれかで行います。コントローラの付属資料を参照してください。**

バスエラー

LLA 操作でバスエラーが報告されることはありません。実際に、[VISA Peek] と[VISA Poke] ではエラー状態は報告されませんが、HLA操作ではバスエラーが報告されます。LLA 操作を使用する場合は、アクセスしているアドレスが有効であるか確認してください。

上位アクセスと下位アクセスの比較

速度

アプリケーションの開発期間という観点から言うと、HLA 操作は非常に短期間で実装とデバッグを行えます。これは、インタフェースが簡単である上に、各アクセスの後にステータス情報が得られるからです。たとえばHLA 操作には、ハードウェアウィンドウのマッピングとマッピング解除が含まれますが、これは、[VISA Map Address] や[VISA Unmap Address] を個別に呼び出す必要がないことを意味します。

実行速度に関しては、マップされた1つのウィンドウ内で複数のランダムレジスタI/Oアクセスに使用する場合は、LLA操作の方が高速です。後に続くいくつかのアクセスが1つのウィンドウ内で行われることがわかっている場合は、マップ処理を1回行うだけで済むため、各アクセスでのオーバーヘッドを最小限に押さえることができます。

HLA操作は、それぞれの呼び出しの中でマッピングを行い、アクセスし、マッピングを解除する必要があるため、速度は遅くなります。また、アクセスできるようにウィンドウが正しくマップされている場合でも、HLAは、マッピングをやり直す必要があるかないかを判断するための何らかのチェックだけは少なくとも行わなくてはなりません。さらに、HLA操作にはLLA操作にはない多くのステータスチェック機能が含まれているため、HLA操作におけるソフトウェアのオーバーヘッドは大きくなります。このような理由から、多くの場合HLAはLLAよりも遅くなります。

注　　ブロック転送の場合、上位VISA Move操作の実行速度が最も速くなります。

使い安さ

HLA操作にはLLA操作には含まれない多くのステータスチェック機能が含まれているため、HLA操作のソフトウェアオーバーヘッドが大きくなり実行速度は遅くなりますが、使用方法は簡単です。また、HLA操作にはハードウェアウィンドウのマップとマップ解除が含まれますが、これは、[VISA Map Address]や[VISA Unmap Address]を個別に呼び出す必要がないことを意味します。

複数のアドレス領域へアクセスする

LLA操作を使用すると、現在1つのVISAセッションにマップされているアドレス領域にしかアクセスできません。1つのセッションで、異なるアドレス領域にアクセスするにはマップし直す必要があり、[VISA Unmap Address]と[VISA Map Address]を呼び出す処理が必要になります。このため、同時に複数のアドレス領域にアクセスする場合はLLAの方がプログラミングが複雑になります。

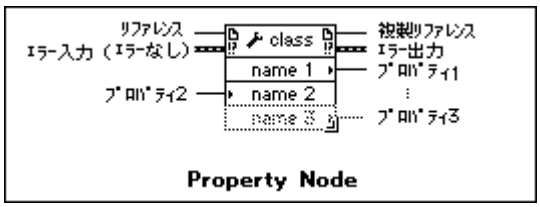
また、同じデバイスまたは異なるデバイスに対してレジスタI/Oを実行している複数のセッションがある場合は、使用可能なハードウェアウィンドウの数が限られているために競合することになります。LLA操作を使用する場合は、ウィンドウを割り当てて、プログラムに必要なウィンドウ数が使用可能なウィンドウ数を超えないようにしなければなりません。HLA操作では、この問題を避けるため、HLA操作の実行終了後にウィンドウの設定を元に戻します。LLA操作ですべてのウィンドウが現在使用されている場合でもHLAを使用することはできますが、これは、HLAではウィンドウのステータスを保存し、マッピングをやり直してアクセスしてからウィ

ンドウを元に戻しているからです。このため、HLA 操作を使用中制約を受けません。

VISA プロパティ

これまで VISA におけるメッセージベースのリソースとレジスタベースのリソースに関連する基本操作について紹介してきました。これらの操作を使用すると、レジスタアクセスやメッセージベースの通信を行うことができます。基本的な通信操作の他に、VISA リソースには、読み込んだりプログラム内で設定可能な値を持つさまざまなプロパティ（属性）があります。

LabVIEW プログラムでは、これらのプロパティもフロントパネルの制御器や表示器のプロパティが扱われるのと同様に、プログラム的に扱われます。VISA プロパティの値の読み込みと設定には、プロパティーノードが使用されます。プロパティノードを次の図に示します。



☞ **注**　**プロパティーノードは一般的なノードであり、ActiveX/OLE や VI Server プロパティの設定にもこのノードを使用できます。**

プロパティーノードをブロックダイアグラム上に配置した後、プロパティを VISA クラス用に設定することができます。この操作を行うには、次のいずれかの方法を使用することができます。

- プロパティーノードの基準入力端子に VISA セッションを配線する。
- プロパティーノードをポップアップして、**VISA クラスを選択**メニューから Instr を選択する。

ブロックダイアグラム上にプロパティーノードを最初に配置したとき、プロパティーノードには 1 つのプロパティ端子が含まれていますが、必要な数の端子が含まれるようにプロパティーノードのサイズを変更することができます。VISA プロパティーノードの初期端子は読み込み端子です。つまり、その端子内で選択されたプロパティの値が読み込まれることを意味します。端子の右端で右側を指している小さい矢印が、このことを示しています。変更したいプロパティでポップアップすれば、多くの端子を個々に読み込み端子から書き込み端子に変更することができます。

☞ **注**　**一部のプロパティは読み込み専用ですので、これらの値を設定することはできません。**

プロパティーノードの各端子内の**プロパティ**を選択するには、プロパティーノードの端子をポップアップしてProperty を選択します。これにより、プログラム内で設定可能なすべてのプロパティのリストが得られます。プロパティ選択肢の下に表示される異なるプロパティの数を制限するには、プロパティーノードのVISAクラスを変更します。

VISAクラスを変更するには、VISA プロパティーノードをポップアップして **VISA クラスを選択**を選択します。このオプションでは、すべての可能なVISAプロパティを含むデフォルトのINSTRクラスの他に、いくつかの異なるクラスを選択することができます。これらのクラスは、すべてのVISA プロパティの代わりに選択されたクラスに関連するプロパティのみを表示します。プロパティーノードの**リファレンス**入力端子にセッションが接続されると、VISA クラスは、そのセッションに対応するクラスに設定されます。

最初、VISA プロパティには何となくわかりにくい部分があり、名前からだけではその正確な性質がわかりません。『LabVIEW オンラインリファレンス』には、プロパティに関する説明が掲載されています。個々のプロパティについての簡単な説明は、シンプルヘルプウィンドウにも表示されます。特定のプロパティについての簡単な説明を表示するには、プロパティーノードのいずれかの端子内のプロパティを選択してヘルプウィンドウを開きます。VXI LA プロパティのヘルプウィンドウを次の図に示します。

ヘルプウィンドウにはプロパティの特定の変数タイプと、プロパティの機能に関する簡単な説明が表示されることに注意してください。プロパティの読み込みや書き込みに使用する変数タイプがよくわからない場合は、プロパティーノードをポップアップして**定数を作成**、**制御器を作成**、または**表示器を作成**を選択すると、適切な変数タイプが自動的に選択されます。

VISA プロパティには、ローカルプロパティとグローバルプロパティという2つの基本タイプがあります。ローカルプロパティがセッションに固有なものであるのに対し、グローバルプロパティはリソースに固有なものです。たとえばVXI LA プロパティはグローバルプロパティの例です。VXI LA プロパティは、そのリソースに対して開かれたすべてのセッションに

対して適用されます。ローカルプロパティは、特定のリソースに対する個々のセッションによって異なる場合があります。ローカルプロパティの例としてはタイムアウト値があります。各リソースタイプに共通のプロパティのいくつかを、次のリストに示します。

シリアル

Serial Baud Rate ― シリアルポートのボーレート。

Serial Data Bits ― シリアル送信に使用されるデータビット数。

Serial Parity ― シリアル送信に使用されるパリティ。

Serial Stop Bits ― シリアル送信に使用されるストップビット数。

GPIB

GPIB Readdressing ― 各書き込み動作の前にデバイスのアドレスを再指定するかしないかを指定します。

GPIB Unaddressing ― 読み込みや書き込み動作の後にデバイスのアドレス指定を解除するかしないかを指定します。

VXI

Mainframe Logical Address ― リソースと同じシャーシ内にあるデバイスの最下位論理アドレス。

Manufacturer Identification ― デバイスの構成レジスタから得られるメーカーのID番号。

Model Code ― デバイスの構成レジスタから得られるデバイスのモデルコード。

Slot ― シャーシ内のデバイスがあるスロット。

VXI Logical Address ― デバイスの論理アドレス。

VXI Memory Address Space ― リソースが使用するVXIメモリのアドレス領域。

VXI Memory Address Base ― リソースが使用するメモリ領域のベースアドレス。

VXI Memory Address Size ― リソースが使用するメモリ領域のサイズ。

ここにあげたものの他にも、多くのプロパティがあります。特定のインタフェースタイプに固有のものでないプロパティもあります。メッセージ

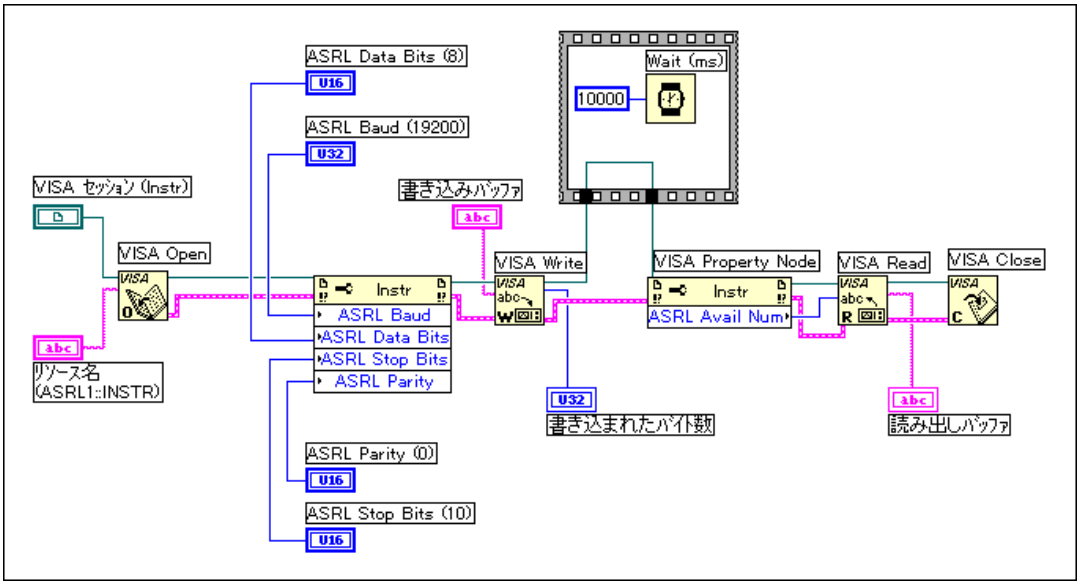
ベースのI/O動作で使用されるタイムアウトであるタイムアウトプロパティは、このようなプロパティのよい例です。プロパティについての資料として最も完全なのは『LabVIEW オンラインリファレンス』で、**ヘルプ→オンラインリファレンス**を選択することによりアクセスできます。

オンラインヘルプには、プロパティが適用されるインタフェースのタイプ、プロパティがローカルかグローバルか、プロパティのデータタイプ、およびそのプロパティに対する有効な値の範囲が表示されます。また、関連項目やプロパティに関する詳細な説明も表示されます。

VISAプロパティの例

シリアル書き込みとシリアル読み込み

ここでは、VISAプログラム内でプロパティを使用した3つの簡単な例を示します。次の図に示す最初の例は、シリアル計測器に文字列を書き込んで応答を読み込みます。

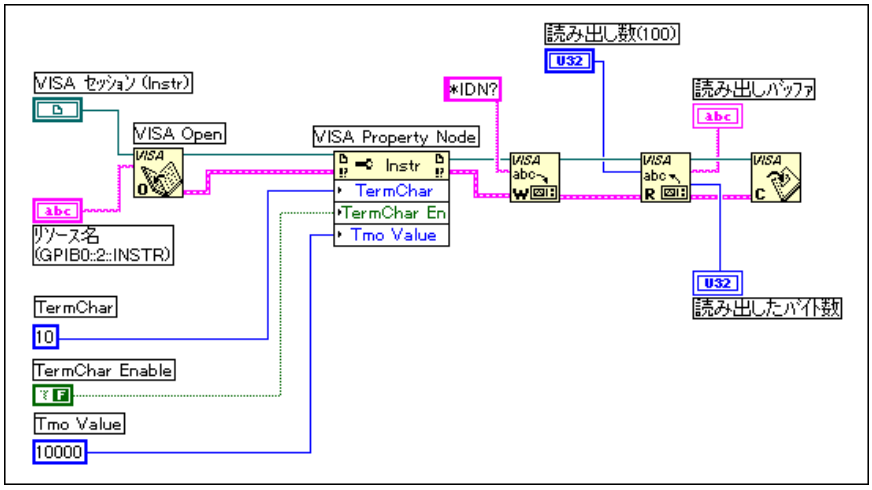


このVIはシリアルポートCOM 1に対してセッションを開き、このポートを19,200ボー、8データビット、パリティなし、1ストップビットに初期化します。このとき、ポートには文字列が書き込まれます。文字列を書き込んで10秒間待った後、別のVISAプロパティを使用して、デバイスから返されたバイト数を取得します。続いてこれらのバイトをポートから読み込みます。ここでは、ストップビット数を1に設定するために10という値

を使用することに注意してください（これは、VISA の仕様によるものであり、10は1 ストップビット、20は2 ストップビットに対応します)。

読み込み動作の終了文字の設定方法

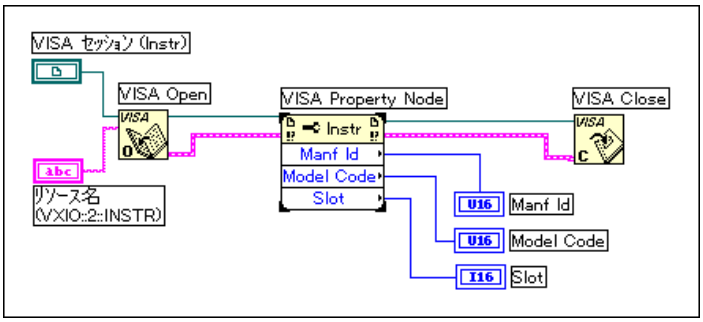
次の例は、プロパティを使用して VISA の読み込み動作に対する終了文字を設定する方法を示します。メッセージベースのデバイスの中には、それ以上送信するデータがない場合に特殊な終了文字を送るものがあります。



この VI は、基本アドレス 2 の位置にある GPIB 計測器に対するセッションを開きます。この VI は、Terminal Character（終了文字）を改行（10 進数の値 10）に設定し、別のプロパティで終了文字の使用ができます。また、この VI は Timeout Property（タイムアウトプロパティ）も 10,000 ミリ秒（10 秒）に設定します。続いて VI は計測器に文字列 *IDN? を書き込み、100 個の応答文字を読み込もうとします。終了文字を受信すると読み込み動作を終了します。VI が停止するのは、終了文字を受信した場合、100 バイトを読み込んだ場合、または 10 秒が経過した場合です。

VXIプロパティ

最後の例は、VXI計測器に関連する共通プロパティのいくつかを読み込む方法を示します。



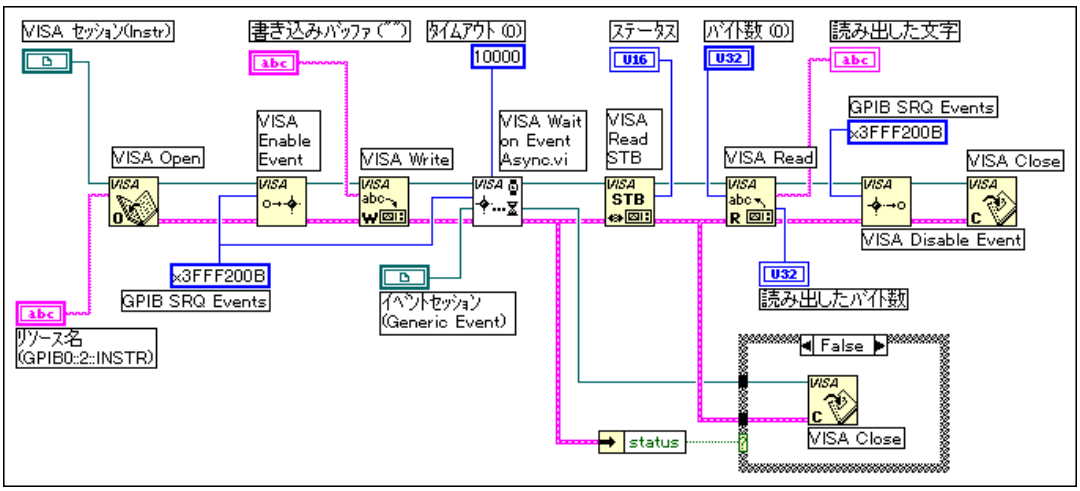
このVIは、論理アドレス2の位置にあるVXI計測器に対するセッションを開き、Manufacturing ID（メーカーID）、Model Code（モデルコード）、およびSlot（VXIモジュールのスロット）を読み込みます。

イベント

イベントとは、リソースとアプリケーションとの間でVISAによる通信を行うための手段です。イベントは、アプリケーションによる対処が必要な何らかの条件が発生したことを、リソースがアプリケーションに知らせるための方法です。さまざまなイベントの例を以下の項に示します。

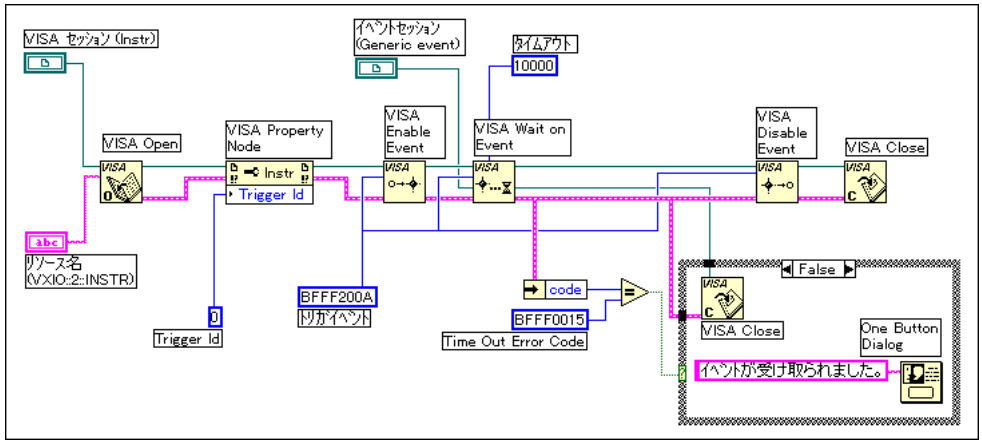
GPIBのSRQイベント

次のブロックダイアグラムは、VISAによりGPIBのサービスリクエスト（SRQ）イベントを処理する方法を示します。



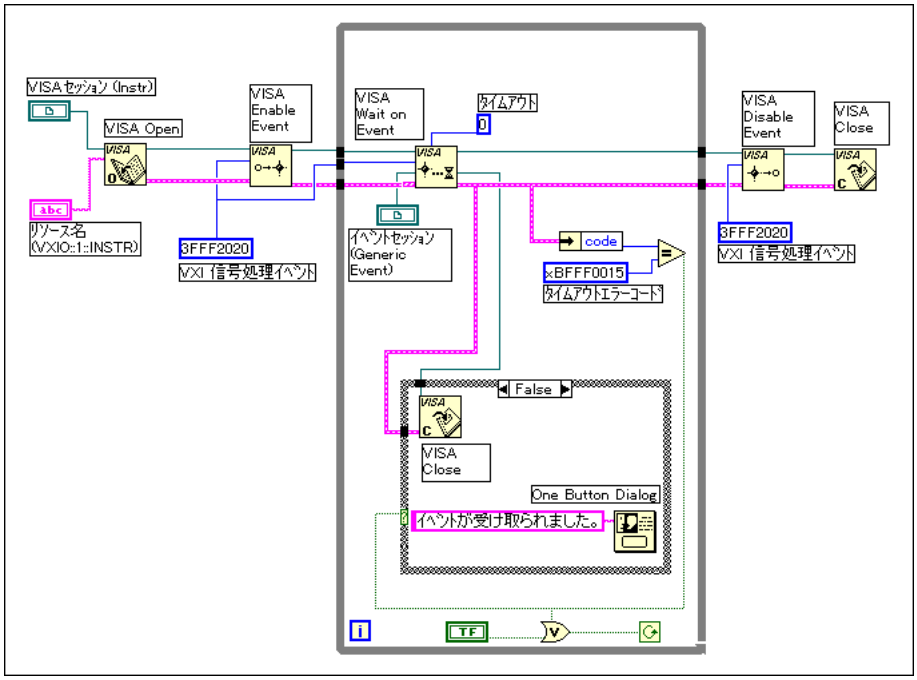
VIはサービスリクエストを可能にした後、計測器にコマンド文字列を書き込みます。計測器が文字列の処理を完了した後、SRQに対する応答を返すことになっています。Wait on Event Async VIを使用して、SRQイベントの発生を10秒間待ちます。SRQが発生すると、Read Status Byte VIにより計測器のステータスバイトが読み込まれます。ステータスバイトの読み込みは、GPIBのSRQイベントが発生した後に行わなければなりません。そうしないと、後でSRQイベントを正常に受け取ることができません。最後に、計測器からの応答が読み込まれ、表示されます。Wait on Event Asyncは、通常のWait on Event VIと異なり、イベントをポーリングするためにタイムアウトを0にした状態で、Wait on Eventを連続的に呼び出します。このため、イベント待機中も、並行処理される他のプログラム部分の実行用に時間が解放されます。

トリガイベント



このダイアグラムは、論理アドレス2の位置にあるデバイス用のTTLトリガライン0でトリガを検出する方法を示します。イベントが有効になる前に、VISAプロパティで検出するトリガイベントのタイプを設定する必要があります。VIは、イベントの受信を最大10秒間待ちます。イベントが正常に受信されると、そのイベントがVIの中で閉じられます。

インタラプトイベント

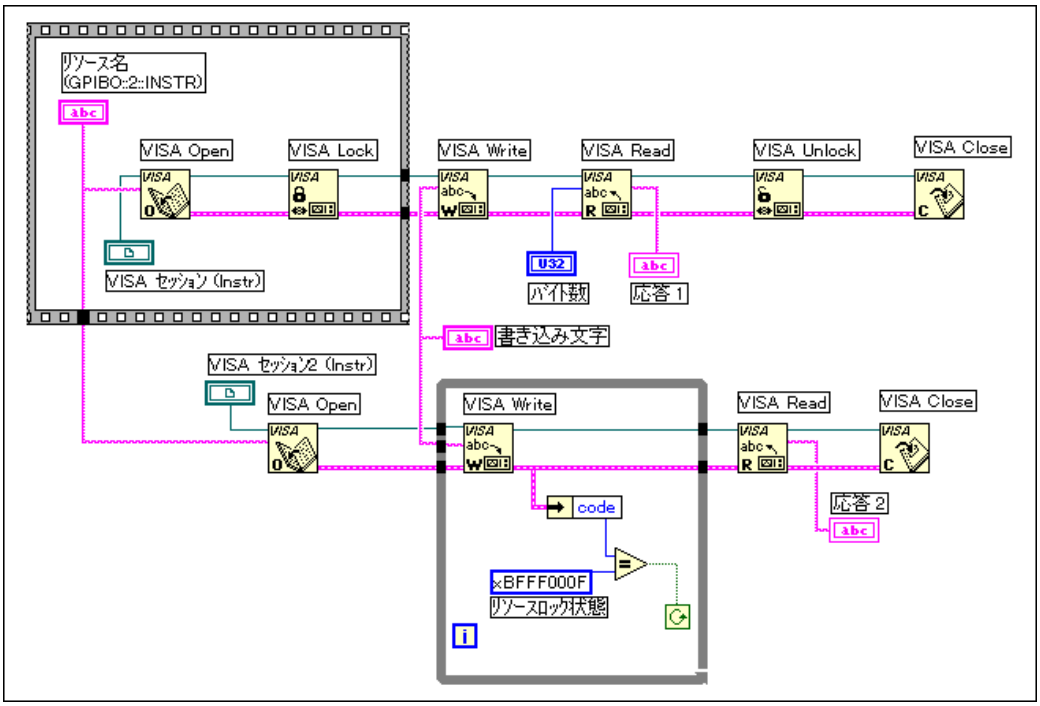


このダイアグラムは、論理アドレス1の位置にあるVXIデバイスによりアサートされたVXIインタラプトを、VISAのイベント処理機能を使用して検出する方法を示します。VIは、VXI信号処理イベントをイネーブルしてからループに入り、そのループはVISA Wait on Eventを繰り返し呼び出します。イベントが受信された場合またはフロントパネルの停止スイッチが選択された場合、ループが終了します。Wait on Event VIにはタイムアウト端子があり、値が0に設定されます。この場合VIは、何らかのイベントが受信されたかを単にチェックするだけで、イベント待ち列にイベントがない場合は直ちにタイムアウトエラーを返します。イベントが受信されると、イベントセッションが閉じられ、イベント通知が生成されます。イベント処理が完了すると、イベントは無効になります。

ロック

VISA を用いて、リソースのアクセス制御をロックできます。VISA では、GPIB や VXI アプリケーションは同じリソースに対して同時に複数のセッションを開いて、これらの異なるセッションを介して同時にリソースにアクセスすることができます。場合によっては、リソースにアクセスしているアプリケーションは、そのリソースにアクセスしないように他のセッションを制限する必要があります。たとえば、アプリケーションによっては、書き込みと読み込みの間に他の動作が行われないよう、書き込みと読み込みを 1 ステップで実行することが必要な場合があります。このようなアプリケーションでは、書き込み動作を起動する前にリソースをロックして読み込み動作の後にロックを解除すれば、2 つの動作を 1 つのステップとして実行することができます。

VISA のロックメカニズムは、リソースに対するアクセスの仲裁を個々に行う必要があります。セッションがリソースをロックした場合に、他のセッションにより呼び出された動作が行われるか、ロックエラーとなって戻るかは、その動作や使用されるロックのタイプによって決まります。

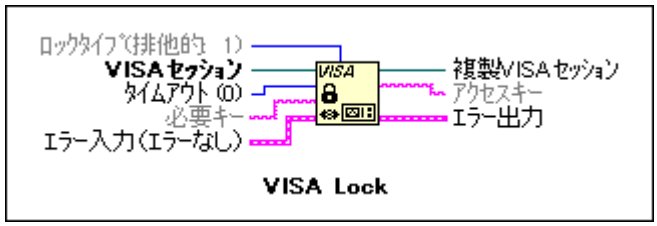


このVIは、同じリソースに対して2つのセッションを開きますが、最初のセッションをロックします。最初のセッションはこの後リソースにコマン

ドを書き込み、応答を読み込みます。書き込み／読み込みのシーケンスが完了すると、最初のセッションはリソースのロックを解除します。ここで第2のセッションが同じ書き込み／読み込みを実行しようとすると、今度はリソースロックエラーを受け取ることはなく正常に書き込み動作が行われます。ロックを使用できるのは、複数のアプリケーションが同じリソースにアクセスする可能性がある場合や、1つのアプリケーション内のリソースに対して複数のセッションが開かれる場合です。

共有ロック

リソースに対するアクセスをロックしながら選択的にこのアクセスを共有したい場合があります。次の図は、複雑なヘルプビューにおけるLock VIを示します。



デフォルトでは**ロックタイプ**は排他的になりますが、これを共有に設定することもできます。さらに**要求されたキー**に文字列を配線すると、この文字列を、他のアプリケーションがこのリソースにアクセスするときに必要なパスワードとすることができます。ただし、ユーザからの要求がない場合、VIは**アクセスキー**の1つを割り当てます。この場合ユーザは、このキーを使用すればロックされているリソースにアクセスできます。

プラットフォームに固有の問題

ここでは、NI-VISAドライバを使用するアプリケーションを開発するときのプログラミングについての注意点を説明します。

ドライバソフトウェアのインストールが終わると、VISAアプリケーションソフトウェアの開発を開始できます。NI-VISAドライバは、NI-488.2とNI-VXIを基本にしてドライバレベルのI/Oアクセスを行っていることを忘れないでください。

- Windows 95/NT**ユーザ**の場合 — VXIシステムやMXIシステムでは、T&M Explorerを使用して、VXIリソースマネージャの実行、ハードウェアの構成、VMEとGPIB-VXIアドレスの割り当てを行います。GPIBシステムの場合は、システムのデバイスマネージャを使用してハードウェアの構成を行います。シリアルポート経由で計測器を制御するに

は、T&M Explorer を使用してデフォルト設定を変更するか、または VISA 属性を設定することによって、必要なすべての構成を動作中に行います。

- **他のすべてのプラットフォーム** ― VXI システムや MXI システムの場合はやはり、VXIinit と Resman を実行し、VXIedit または VXItedit を使用して構成を行う必要があります。同様に、GPIB や GPIB-VXI システムの場合もやはり GPIB コントロールパネルアプレットまたは IBCONF を使用してシステムを構成する必要があります。シリアルポート経由で計測器を制御するには、VISA 属性を設定することによって、必要なすべての構成を動作中に行います。

プログラミングについての注意点

ここでは、NI-VISA I/O インタフェースソフトウェアを使用するアプリケーションを開発するときの注意点を説明します。

NI-VISA ドライバを使用する複数のアプリケーション

複数アプリケーションのサポートは、NI-VISA ドライバのすべての実装例における重要な機能です。NI-VISA を使用する複数のアプリケーションを、同時に実行することができます。アプリケーションがそのように設計されていれば、NI-VISA ドライバを使用する複数個の同じアプリケーションを同時に実行することもできます。NI-VISA ドライバを使用するのが 1 つのアプリケーションであっても、複数のアプリケーション（または複数個の同じアプリケーション）であっても、NI-VISA 動作は同じ動作を行います。

ただし、下位 VXIbus アクセス関数を使用する複数のアプリケーションまたはセッションがある場合は注意が必要です。VXIbus へのアクセスに使用されるメモリウィンドウは、制約のあるリソースです。viPeekXX() や viPokeXX() で下位 VXIbus アクセスを実行する前に、viMapAddress() 操作を呼び出す必要があります。アクセスの完了後は直ちに必ず viUnmapAddress() 操作を呼び出して、他のアプリケーションにメモリウィンドウを解放しなければなりません。

複数のインタフェースのサポートに関する問題

ここでは、いくつかのタイプのインタフェースに対して NI-VISA ソフトウェアを使用したり構成する方法について説明します。

VXI および GPIB プラットフォーム

NI-VISA は、NI-VISA がインストールされているオペレーティングシステム用のナショナルインスツルメンツの既存の VMI、GPIB、シリアルハードウェアをすべてサポートします。VXI の場合は、MXI-1、MXI-2 プラットフォーム、GPIB-VXI、および VXIpc が組み込まれた一連のコンピュー

タがサポートされます。GPIB の場合は、PCI-GPIB、NB-GPIB、GPIB-SPARC シリーズ、AT-GPIB/TNT ボードシリーズの全機種、および GPIB デバイスのリモートコントロールに使用可能な GPIB-ENET ボックスなどがあります。GPIB-ENET の場合は、GPIB-VXI コントローラを使用すると VXI デバイスをリモートコントロールすることもできます。

複数の GPIB-VXI のサポート

Windows 95/NT のユーザは、T&M Explorer ユーティリティを参照して複数のナショナルインスツルメンツ GPIB-VXI コントローラや、その他のベンダの GPIB-VXI コントローラをシステムに追加できます。Windows 3.*x* と UNIX のユーザがコントローラを追加するには、VISAconf ユーティリティを使用する必要があります。

シリアルポートのサポート

現在のところ、NI-VISA は指定されたシリアルポートで同時に 1 つのセッションしかサポートしておりません。NI-VISA が現在サポート可能なシリアルポートの数はどのプラットフォームでも最大で 32 です。デフォルトの場合のシリアルポートの番号付けはシステムによって異なります。

プラットフォーム	方法
Windows 3.x, Windows 95, Windows NT	`ASRL1` ～ `ASRL4` は `COM1` ～ `COM4` にアクセスします。 `ASRL10` ～ `ASRL13` は `LPT1` ～ `LPT4` にアクセスします。
Macintosh 68K, Macintosh PPC	`ASRL1` はモデムポートにアクセスします。 `ASRL2` はプリンタポートにアクセスします。
Solaris 1.*x*	`ASRL1` ～ `ASRL6` は `/dev/ttya` ～ `/dev/ttyf` にアクセスします。
Solaris 2.*x*	`ASRL1` ～ `ASRL6` は `/dev/cua/a` ～ `/dev/cua/f` にアクセスします。
HP-UX 9 HP-UX 10	`ASRL1` と `ASRL2` は HP-UX9 の `/dev/tty00` ～ `/dev/tty01` のシリアルポート1と2にアクセスします。HP-UX10 は `/dev/tty0p0` と `/dev/tty1p0` を使用します。増設ポートには ASRL3 から始まる連続番号が付けられ、`/dev/tty02` が使用されます。

VMEのサポート

システム内のVMEデバイスにアクセスするには、これらのデバイスが表示されるようにNI-VXIを構成する必要があります。Windows 95/NTのユーザは、T&M ExplorerのAdd Deviceウィザードを使用してNI-VXIを構成できます。他のプラットフォームのユーザは、VXIeditまたはVXItedit の中のNon-VXI Device Editorを使用する必要があります。デバイスのメモリが含まれる各アドレス領域ごとに、256から511の論理アドレスを持つ疑似デバイスエントリを個別に作成しなければなりません。たとえば、A24とA32という両方の領域の中にメモリが含まれるVMEデバイスには2つのエントリが必要です。また、デバイスが使用するインタラプトレベルも指定することができます。VXIデバイスとVMEデバイスがインタラプトレベルを共有することはできません。このような場合は、VXIデバイスの場合と同様に前に構成してあるアドレス領域とベースからのオフセットを指定することで、NI-VISAからデバイスにアクセスします。VMEデバイスに対するNI-VISAのサポートには、レジスタアクセス操作（上位と下位の両方）、ブロック転送操作、インタラプト受信機能が含まれます。

VISAプログラムをデバッグする

ここでは、VISAプログラムのデバッグについて説明します。VISAの性質上、VISAの異常をデバッグする際には、スタンドアロンのドライバをデバッグする場合よりも考慮すべき点が多くあります。VISAは、NI-VXI、NI-488、またはオペレーティングシステムのシリアルAPIを呼び出します。したがって、VISAの異常のように見える問題が、VISA自体ではなくVISAの呼び出すドライバに関係している場合もあります。

LabVIEW（計測器ドライバも含む）でVISA VIが全く動作していないように見える場合、とるべき対策の第一歩はVISA Find Resource VIです。このVIは、ブロックダイアグラム内に他のVISA VIがなくても動作します。このVIでおかしなVISAエラーが発生する場合に最も可能性が高いのは、インストールされたVISAのバージョンやインストール方法の間違いです。VISA Find Resource VIが正常に動作していれば、LabVIEWとVISAドライバは正常に動作しています。次に、LabVIEWプログラムのどのVIのシーケンスでエラーが発生しているかを調べます。

エラーが発生しているのがイベントの単純なシーケンスである場合、次に行うべきデバッグ手順は、VISAICユーティリティを使用して同じシーケンスを対話式に実行してみることです（次項を参照）。一般的に、初期段階でのプログラム開発は対話式に行うのが良いでしょう。対話式ユーティリティでは正常に動作するシーケンスがLabVIEWで正常に動作しない場合は、LabVIEWとVISAドライバとの間の相互動作に問題があることを示します。VISAICで同じシーケンスを対話式に動作させても同じ問題が発生する場合は、VISAが呼び出しているドライバのいずれかに異常がある可

能性があります。これらのドライバ（NI-VXI 用の VIC および NI-488.2 用の IBIC）に対して対話式ユーティリティを使用して同じ動作を行わせてみることもできます。このレベルで異常を解決できない場合は、下位ドライバまたはそのインストール状態に異常があることを示します。

Windows 95/NT 用のデバッグツール

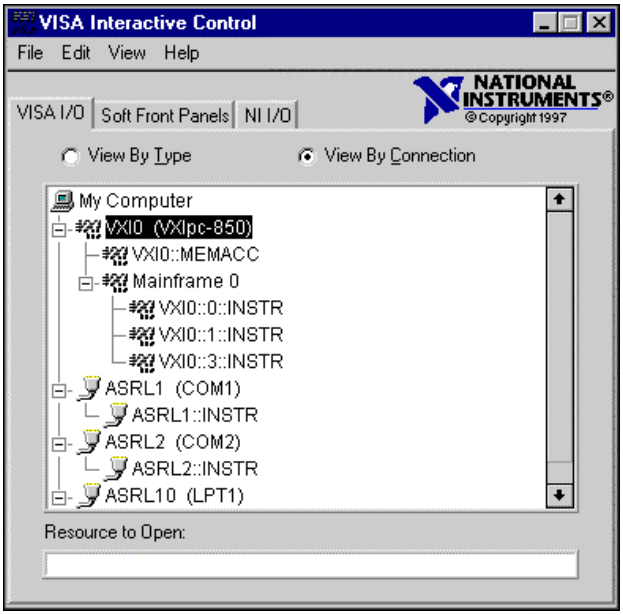
NI Spy は、NI-VISA、NI-VXI、NI-488.2 など、ナショナルインスツルメンツのテスト・測定用（T&M）ドライバに対してアプリケーションが行った呼び出しを追跡記録します。NI-488.2 のユーザは、NI Spy が GPIB Spy に似ていることにお気づきになると思います。

NI Spy では、エラーを返した関数がハイライト表示になりますので、開発中にどの関数で異常が発生したかをすばやく判断することができます。また、NI Spy ではこれらのドライバに対するプログラムからの呼び出しが記録されますので、必要に応じこれらをチェックしてエラーがないか確認できます。

VISAIC

VISA と LabVIEW をサポートする Macintosh 以外のどのプラットフォーム用の VISA にも、VISA 対話式制御（VISA Interactive Control : VISAIC）というユーティリティが添付されています。このユーティリティを使用すると、使いやすいグラフィカル環境内で、対話形式で VISA のすべての機能にアクセスできます。プログラム開発や VISA の学習を始める際には、VISAIC が役に立ちます。

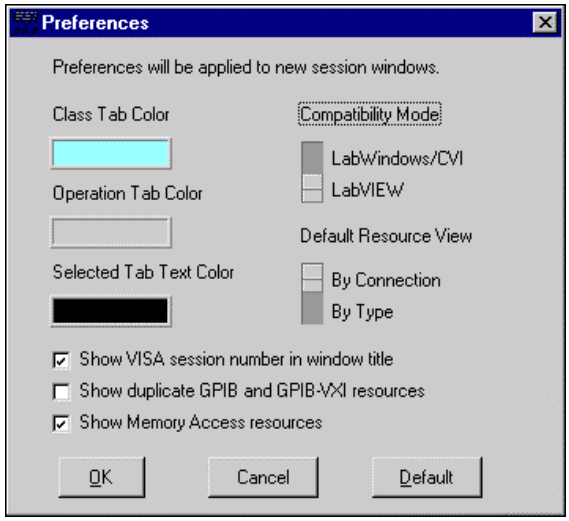
VISAIC を動作させると、システム内の使用可能なリソースをすべて自動的に検出し、これらのリソースの計測器デスクリプタを対応するリソースタイプの下にリスト表示します。VISAIC を開いたときのウィンドウを、次の図に示します。



VISAIC のメインパネルの Soft Front Panels タブには、システムにインストールされた任意のVXI **Plug&Play**計測器ドライバのソフトフロントパネルを起動するためのオプションが用意されています。

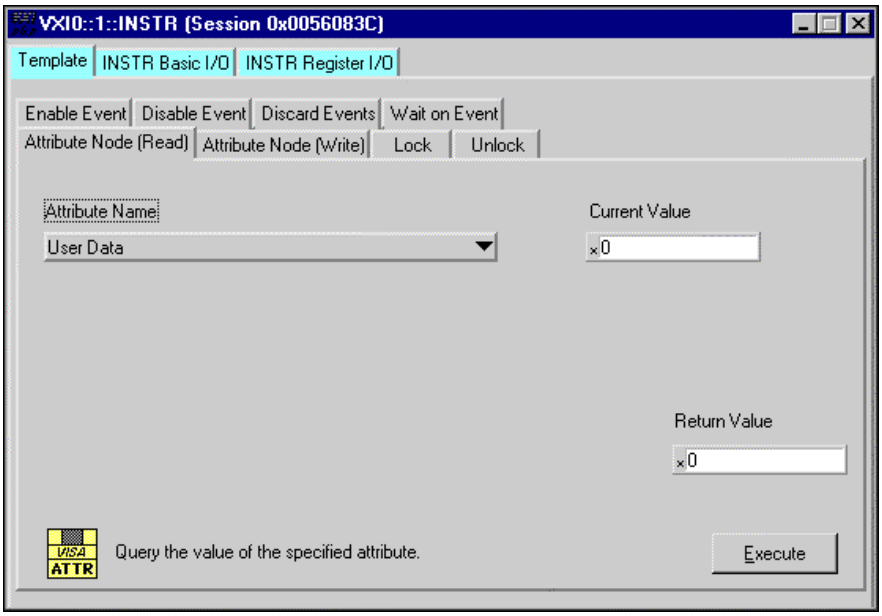
NI I/O タブには、対話式の NI-VXI ユーティリティまたは対話式の NI-488 ユーティリティを起動するためのオプションがあります。このタブは、VISAが呼び出すドライバ用の対話式ユーティリティにリンクされており、このレベルでデバッグを行いたい場合に便利です。

VISAIC ウィンドウに表示される計測器デスクリプタのいずれかをダブルクリックすると、その計測器に対するセッションが開かれます。計測器に対してセッションを開くと、VISA コマンドを対話式に実行するための一連のタブを持つウィンドウが表示されます。これらのタブの正確な外観は、VISAIC がどの互換性モードになっているかによって異なります。互換性モードや VISAIC のその他の環境設定にアクセスするには**編集→環境設定** ... を選択します。次の図のようなウィンドウが表示されます。



VISA の実装は、LabVIEW と LabWindows/CVI では多少異なります。これらの相違点はリソースに対するセッションを開くときに表示される操作タブに反映されます。デフォルトでは、互換性モードは Labwindows/CVI に設定されておりますので、LabVIEW に変更する必要があります。ユーザ設定を変更すると、これ以後開かれるセッションでは新しいユーザ設定が有効になります。

リソースに対して対話形式でセッションが開かれると、次のようなウィンドウが表示されます。



ウィンドウには3つのメインタブが表示されます。最初のタブはTemplateタブで、Event、Property、Lockを扱うすべての操作が含まれています。メインタブの下に、これらの各動作ごとに異なるタブがあることに注意してください。その他のメインタブとしては、INSTR Basic I/OとINSTR Register I/Oがあります。Basic I/Oタブにはメッセージベースの計測器に対する基本操作が含まれるのに対し、Register I/Oタブにはレジスタベースの計測器に対する基本操作が含まれています。VXI計測器の場合はRegister I/Oタブだけが表示されます。

9

LabVIEW の GPIB 関数の概要

この章では、汎用インタフェースバス（General Purpose Interface Bus: GPIB）の動作、およびIEEE 488 とIEEE 488.2 インタフェースの相違点について説明します。

GPIB にはIEEE 488 とIEEE 488.2 という2つの規格があります。ヒューレットパッカード社は、一連のプログラマブル計測器を相互に接続して制御するための GPIB を設計しました（もともとは HP-IB と呼ばれていました）。GPIB はその 1 Mbytes/s という最大データ転送速度のために、すぐにコンピュータ間の通信や周辺機器の制御など他のアプリケーションにも適用されました。その後この規格はIEEE 規格 488-1975 として認定され、さらに進化して ANSI/IEEE 規格 488.2-1987 になりました。その汎用性から、このシステムには汎用インタフェースバスという名前が付けられました。

ナショナルインスツルメンツは、高性能で高速なハードウェアインタフェースと豊富な機能を備えた総合的なソフトウェア専用の、ヒューレットパッカード製以外のコンピュータやデバイスのユーザのための GPIB を用意しました。LabVIEW 用の GPIB 関数は、IEEE 488.2 の仕様に適合します。

メッセージのタイプ

GPIB には、デバイスによって異なるメッセージとインタフェースメッセージがあります。

- デバイスによって異なるメッセージはしばしば**データ**あるいは**データメッセージ**と呼ばれ、プログラム命令、測定結果、マシンステータス、データファイルなど、デバイス固有の情報が含まれています。
- インタフェースメッセージは、バスそのものを管理します。このメッセージは普通、**コマンド**あるいは**コマンドメッセージ**と呼ばれています。インタフェースメッセージは、バスの初期化、デバイスのアドレス指定とアドレス指定解除、リモートプログラミングまたはローカルプログラミング用のデバイスモードの設定、などのタスクを実行します。

一部のデバイス命令も**コマンド**と呼ばれますが、その場合のコマンドと、ここで使用されているコマンドという用語を混同しないでください。そのようなデバイスに固有な命令は、実際にはデータメッセージです。

ANSI/IEEE 規格 488.2-1987 は、以前の IEEE 488.1 規格を拡張してコントローラが GPIB を管理する方法を厳密に記述したものであり、規格に準拠したデバイスが理解すべき標準メッセージ、デバイスエラーその他のステータス情報を報告するメカニズム、GPIB バスに接続された規格準拠デバイスを見つけて構成するためのさまざまなプロトコルなどが含まれています。

IEEE 488.2 では、GPIB 上のアクティブなデバイス（トーカとリスナ）を検出するのに必要な場合はいつでも、任意のバスラインをモニタできます。GPIB デバイスは、トーカ、リスナ、もしくはコントローラになることができます。たとえばデジタルボルトメータはトーカですが、リスナになることもあります。トーカは、1 つまたは複数のリスナにデータメッセージを送ります。コントローラは、すべてのデバイスにコマンドを送って GPIB 上の情報の流れを管理します。

GPIB は通常のコンピュータのバスと似ていますが、コンピュータではマザーボードのバスで回路基板が相互に接続されているのに対し、GPIB ではケーブルバスによってスタンドアロンのデバイスが相互に接続されています。

GPIB コントローラの役割は、コンピュータの CPU の役割と似ていますが、さらに似ているのが、街の電話システムの交換機センターです。交換機センター（コントローラ）は通信ネットワーク（GPIB）を監視します。センター（コントローラ）は利用者（デバイス）が電話をかけたい（データメッセージを送りたい）ことを認識すると送信側（トーカ）を受信側（リスナ）を接続します。

コントローラがトーカとリスナのアドレスを指定してからでないと、トーカはリスナにメッセージを送信できません。トーカがメッセージを送信した後、コントローラは両方のデバイスのアドレス指定を解除することができます。

バス構成によっては、コントローラが不要な場合もあります。たとえば、1 つのデバイスが常にトーカ（トーカ専用デバイスという）になり、1 つまたは複数のリスナ専用デバイスが存在する場合などがあります。

アクティブな、またはアドレス指定されているトーカやリスナを変更しなければならない場合は、コントローラが必要です。コンピュータは通常、コントローラの機能を果たします。

パーソナルコンピュータは、GPIB ボードやそれに対応するソフトウェアを使用して 3 つの役割をすべて果たします。

- コントローラ — GPIB を管理する
- トーカ — データを送信する
- リスナ — データを受信する

コントローラインチャージとシステムコントローラ

GPIB は複数のコントローラを持つことができますが、同時にアクティブコントローラまたコントローラインチャージ（CIC）になれるのは 1 つのコントローラだけです。現在のCICからアイドリング状態のコントローラに、アクティブ制御を渡すことができます。バス上の1つのデバイス ― システムコントローラ ― だけがそれ自体をCICにすることができます。通常は、GPIB ボードがシステムコントローラとなります。

互換性のあるGPIBハードウェア

以下のナショナルインスツルメンツの GPIB ハードウェア製品は、LabVIEW に対応しています。

Windows 95対応LabVIEW および日本語版Windows 95対応LabVIEW

- AT-GPIB/TNT、AT-GPIB/TNT (PnP)、AT-GPIB/TNT+、PCI-GPIB
- PCMCIA-GPIB、PCMCIA-GPIB+
- GPIB-ENET
- EISA-GPIB
- VXIpc Model 850
- NEC-GPIB/TNT、NEC-GPIB/TNT (PnP)
- GPIB-PCII/IIA
- PC/104-GPIB
- CPCI-GPIB
- GPIB-ENET
- PMC-GPIB

Windows NT対応LabVIEW

- AT-GPIB、AT-GPIB/TNT
- PCMCIA-GPIB
- PCI-GPIB
- VXIpc Model 850
- GPIB-ENET

Windows 3.1対応LabVIEW

- AT-GPIB、AT-GPIB/TNT、AT-GPIB/TNT (PnP)、AT-GPIB/TNT+、PCI-GPIB
- PCMCIA-GPIB、PCMCIA-GPIB+
- GPIB-ENET
- EISA-GPIB
- VXIpc Model 850
- NEC-GPIB/TNT（日本語）、NEC-GPIB/TNT (PnP)（日本語）、GPIB-PCII/IIA
- GPIB-232CT-A
- GPIB-485CT-A
- GPIB-1284CT
- PCII/IIA
- STD-GPIB
- EXM-GPIB
- MC-GPIB

Mac OS対応LabVIEW

- PCI-GPIB
- NB-GPIB/TNT, NB-GPIB-P/TNT
- PCMCIA-GPIB
- LC-GPIB
- GPIB-ENET
- GPIB-232CT-A
- GPIB-SCSI-A
- PC/104-GPIB
- NB-DMA2800（従来のGPIB機能のみ）

HP-UX対応LabVIEW

- GPIB-ENET
- EISA-GPIB
- AT-GPIB/TNT

Sun対応LabVIEW

- GPIB-ENET
- GPIB-SCSI-A
- SB-GPIB/TNT

Concurrent PowerMAX対応LabVIEW

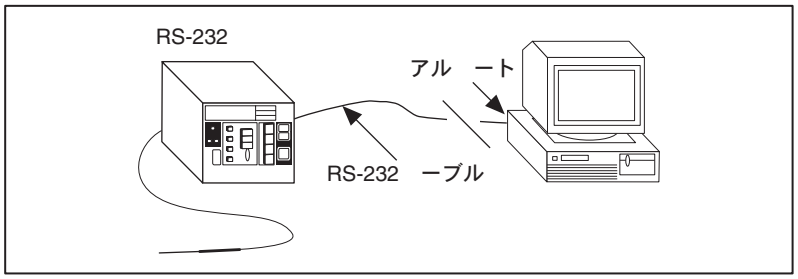
- GPIB-1014
- GPIB-1014D
- GPIB-1014P
- GPIB-1014DP

10

シルアルポートVI

この章では、シリアルポート操作用のVIを説明し、さらにシリアル通信に影響を及ぼす重要な要因について説明します。

シリアル通信は、コンピュータと、プログラマブル計測器などの周辺機器または他のコンピュータとの間のデータ転送によく使用される手段です。シリアル通信は、送信装置を使用して、1本の通信回線を通して一度に1ビットずつ受信側にデータを送ります。データ転送速度が遅い場合や、長距離でデータを転送しなければならない場合に、この方法を使用することができます。



シリアル通信が広く使用されるのは、ほとんどのコンピュータにシリアルポートが1つか2つ組み込まれているからです。また、多くのGPIB計測器にもシリアルポートが用意されています。ただし、シリアル通信では、シリアルポートは1つのデバイスとしか通信できないという制約があります。

周辺機器の中には、データ文字列の送信を終了させるための文字が必要なものがあります。よく使用される終了文字としては、復帰文字、改行文字、セミコロンがあります。終了文字が必要かどうかを判断するには、各デバイスのマニュアルを調べてください。

シリアルポートVIの使用方法の例は、`examples¥instr¥smplserl.llb`を参照してください。

ハンドシェイクモード

シリアル通信でよく問題になるのは、送／受信の両方が確実にデータ転送に追随できるようにすることです。シリアルポートドライバは情報をバッファに蓄えることができますが、バッファのサイズには限りがあります。バッファがいっぱいになると、バッファから十分な量のデータが読み込まれ新しい情報を格納するための領域ができるまで、コンピュータは新しいデータを無視します。

ハンドシェイクを行うと、バッファのオーバーフローを防止することができます。ハンドシェイクを行った場合、送信側と受信側は相手のバッファがいっぱいになったことを認識できます。このような場合、送信側は、シリアル通信の相手が新しいデータを受信できる状態になるまで、新しい情報の送信を停止します。

LabVIEW では、ソフトウェアハンドシェイクとハードウェアハンドシェイクという 2 種類のハンドシェイクを行えます。Serial Port Init VI を使用すると、どちらの形のハンドシェイクでも、オン／オフすることができます。デフォルトでは、VI はハンドシェイクを使用しません。

ソフトウェアハンドシェイク ― XON/XOFF

XON/XOFFは、ソフトウェアによりハンドシェイクを行うためのプロトコルであり、これを使用すると、シリアルポートバッファのオーバーフローを防止することができます。受信側のバッファがいっぱいに近くなると、受信側はXOFF（<Ctrl-S>10進数の19）を送信して相手のデバイスにデータ送信を停止するように知らせます。受信側のバッファに十分な空き領域ができると、受信側はXON（<Ctrl-Q>10 進数の 17）を送り、送信を再開できることを知らせます。XON/XOFF を有効にすると、デバイスは <Ctrl-Q> と <Ctri-S> を必ずXON と XOFF 文字として解釈し、データとしては解釈しません。XON/XOFF を無効にした場合は、<Ctrl-Q> と <Ctrl-S> をデータとして送信することができます。バイナリデータの転送には XON/XOFF を使用しないでください。これは、データ内に <Ctrl-Q> や <Ctrl-S>が含まれている場合に、デバイスがこれらをデータでなくXONやXOFFとして解釈してしまうからです。

エラーコード

エラーコードパラメータは、エラー処理 VI のいずれかと接続ができます。これらの VI を使用すると、エラーの説明が表示され、ユーザはエラーが発生したときの対処方法を選択することができます。

シリアルポート VI により返されるエラーコードの中には、プラットフォームに固有なものがあります。エラーコードのリストについては、各システムのマニュアルを参照してください。

ポート番号

Windows 95/NT、および Windows 3.*x*

Windows 95 や Windows 3.*x* でシリアルポート VI を使用する場合は、**ポート番号**パラメータを以下の値にすることができます。

0:	COM1	5:	COM6	10:	LPT1
1:	COM2	6:	COM7	11:	LPT2
2:	COM3	7:	COM8	12:	LPT3
3:	COM4	8:	COM9	13:	LPT4
4:	COM5				

Windows 95 または Windows NT でシリアルポート VI を使用する場合の**ポート番号**パラメータは、COM1 の場合は 0、COM2 の場合は 1 などとなります。

Macintosh

Macintosh では、ポート 0 はモデムであり、ドライバ `.ain` と `.aout` を使用します。ポート 1 はプリンタであり、ドライバ `.bin` と `.bout` を使用します。Macintosh でさらに多くのポートを使用するには、ドライバを装備した別のボードをインストールする必要があります。

UNIX

Sun SPARCstation で Solaris 1 と Concurrent PowerMAX を使用している場合のシリアルポート VI 用の**ポート番号**パラメータは、`/dev/ttya` の場合は 0、`/dev/ttyb` の場合は 1、のようになります。Solaris 2 では、ポート 0 は `/dev/cua/a`、ポート 1 は `/dev/cua/b`、になります。HP-UX の場合は、ポート番号 0 は `/dev/tty00`、ポート番号 1 は `/dev/tty01`、になります。

Concurrent PowerMAX では、ポート 0 は /dev/console、ポート 1 は /dev/tty1、ポート 2 は /dev/tty2 などとなります。

他社のシリアルポートボードでは任意にデバイス名が付けられていることがありますので、LabVIEW では、ポートの番号設定が簡単にできるような簡易インタフェースを開発しました。Sun、HP-UX、Concurrent PowerMAX 用の LabVIEW には、シリアルポートのアドレス指定状態を LabVIEW に指示する構成オプションがあります。LabVIEW は、標準 UNIX デバイスを使用するすべてのボードをサポートします。一部のメーカーは、自社のボードには tty デバイスノードではなく cua を使用するように推奨しています。LabVIEW は、どちらのタイプのノードでもアドレス指定できます。

ファイル .labviewrc には、LabVIEW 構成オプションが含まれています。シリアルポート VI の使用するデバイスを設定するには、構成オプション labview.serialDevices を、使用する予定のデバイスのリストに設定します。

たとえば、デフォルトでは次のようになります。

```
labview.serialDevices:/dev/ttya:/dev/ttyb:/dev/ttyc:...
:/dev/ttyz.
```

☞ **注**　**このためには、他社のシリアルボードがインストールされている場合はすべて、標準 /dev ファイル（ノード）を作成する方法が含まれ、ユーザがそのファイル名を知っていることが必要です。**

第III部

解析

第III部では、集録データの解析、信号処理、信号生成、線形代数、カーブフィット、確率、統計に関する基本的な事項を説明します。

「第III部 解析」は、以下の章から構成されています。

- 「第11章 LabVIEWにおける解析の概要」では、サポートされている機能、表記法と命名法に関する規則、サンプリング信号の方法など、すべての解析アプリケーションに適用される概念を紹介します。
- 「第12章 信号生成」では、標準周波数を使用して信号を生成する方法、およびシミュレーションによる関数生成器の作成方法を説明します。
- 「第13章 デジタル信号処理」では、高速フーリエ変換と離散的フーリエ変換の基礎を説明し、スペクトル解析におけるこれらの使用方法を示します。
- 「第14章 スムージングウィンドウ」では、ウィンドウを使用してスペクトルの漏洩を防ぎ、集録された信号の解析を改善する方法を説明します。
- 「第15章 スペクトルの解析と測定」では、振幅と位相のスペクトルの測定方法、スペクトルアナライザの開発方法、信号の全高調波歪み（THD）の測定方法を示します。
- 「第16章 フィルタ処理」では、無限インパルス応答フィルタ（IIR）、有限インパルス応答フィルタ（FIR）、非線形フィルタを使用して、信号から不必要な周波数成分を除去する方法を説明します。
- 「第17章 カーブフィット」では、データセットから情報を抽出して関数についての記述を取得する方法を示します。
- 「第18章 線形代数」では、行列演算と解析の実行方法を説明します。
- 「第19章 確率と統計」では、確率と統計に関するいくつかの基本概念を説明し、実際の問題解決におけるこれらの概念の使用方法を示します。

11

LabVIEW における解析の概要

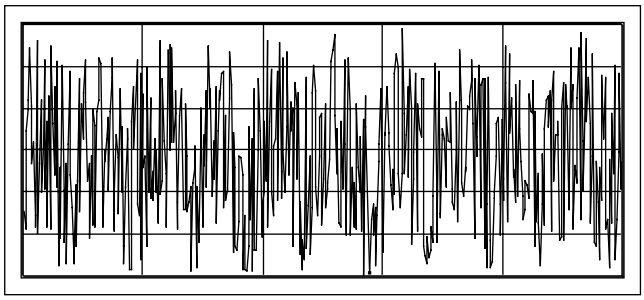
デジタル信号は、私たちの周囲の至る所にあります。電話会社はデジタル信号を使用して人間の音声を表現しています。ラジオ、テレビ、hi-fi 音響システムはいずれも、その優れた忠実度、雑音制御、信号処理の柔軟性から、次第にデジタル分野に移行しつつあります。データは、衛星から地上ステーションにデジタル形式で送信されています。NASA による遠距離惑星や外宇宙の画像は、多くの場合、ノイズを除去して有用な情報を抽出するためにデジタル処理されます。経済データ、国勢調査結果、株価などもすべて、デジタル形式のものを使用できます。デジタル信号処理には多くの長所があるため、アナログ信号も、デジタル形式に変換してからコンピュータで処理されます。

この章では、デジタル信号処理に関する基本事項を説明し、信号処理や解析を行うための数百の VI から構成される LabVIEW 解析ライブラリの概要を示します。

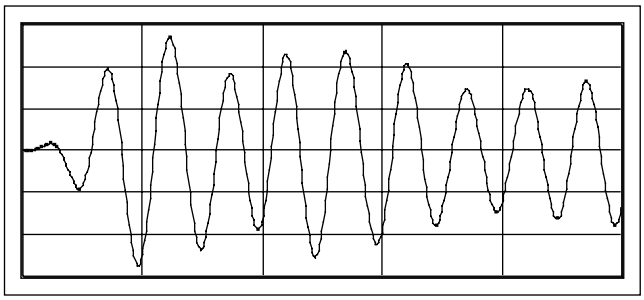
データ解析の重要性

最新の高速浮動小数点数プロセッサおよびデジタル信号プロセッサは、リアルタイムの解析システムにとってますます重要になってきました。考えられる多くのアプリケーションの中には、少しではありますが、生医学データ処理、音声合成と認識、デジタル音声画像処理なども含まれます。

エンジニアリングステーションに解析ライブラリを組み込むのが重要なのは、次の図のような生のデータが必ずしも役に立つ情報にはならないからです。多くの場合、信号を変換したり、妨害となるノイズを除去したり、装置の不備による不完全なデータを修正したり、温度や湿度などの環境影響を考慮して補正する必要があります。



デジタルデータを解析して処理すると、ノイズの中から役立つ情報を抽出したり、生のデータよりもわかりやすい形で表示することができます。処理後のデータを次の図に示します。



LabVIEW のブロックダイアグラムによりプログラミングを行う方法と広範囲に渡る解析VIにより、解析アプリケーションの開発が簡単にできるようになります。

LabVIEWの解析VIには、互いに配線することができるVIを使用した最新のデータ解析技術が含まれています。従来のプログラミング言語のように解析ルーチンの実装に関する詳細事項に煩わされずに、データ解析上の問題解決に集中することができます。

開発システム

基本解析 VI ライブラリは、上級解析 VI ライブラリのサブセットです。基本解析ライブラリには、統計解析、線形代数、数量解析用の VI が含まれています。上級解析ライブラリには、これらの分野の VI がより多く含まれているほか、信号生成、時間周波数領域のアルゴリズム、ウィンドウ処理ルーチン、デジタルフィルタ、評価、回帰などを行うための VI が含まれています。

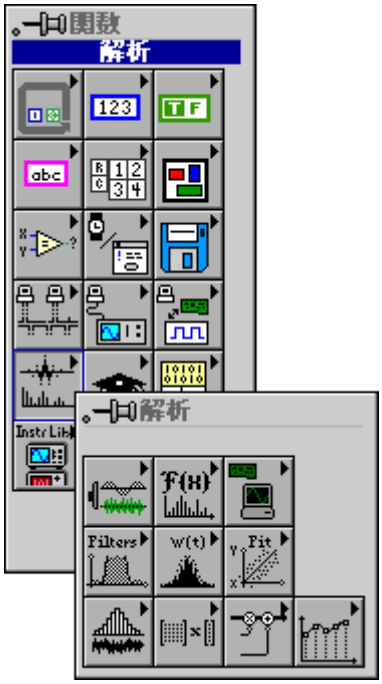
基本解析ライブラリに含まれている VI では不十分な場合は、LabVIEW 上級解析ライブラリを LabVIEW ベースパッケージに追加することができます。このアップグレードを行うと、開発システムで使用可能なすべての解析ツールが揃います。

解析 VI の概要

アナログ信号が A/D コンバータ（ADC）でデジタル形式に変換され、コンピュータでデジタル信号（一連のサンプル）として使用可能な状態になると、通常はこれらのサンプルを何らかの方法で処理したくなります。ここで言う処理には、サンプルを取り込んだシステムの特性を判断するための処理、信号の特定の特徴を測定するための処理、信号を人間の理解しやすい形に変換する処理などがあります。

解析ライブラリには、数量解析、信号生成、信号処理、カーブフィット、測定、その他の解析機能を実行する広範囲の VI が含まれています。LabVIEW 開発システムに含まれている解析ライブラリは、仮想計測器システムを構築する際の中心となる要素です。このライブラリには、多くの数学パッケージに含まれている解析機能が含まれている他、計測業界専用に設計された多くのユニークな信号処理関数や測定関数も含まれています。

LabVIEW の解析 VI は、LabVIEW または BridgeVIEW の**関数**パレットの**解析**サブパレットにあります。



解析 VI ライブラリは 10 個あります。主なカテゴリを以下に示します。

信号生成：デジタルパターンや波形を生成する VI。

デジタル信号処理：周波数領域の変換、周波数領域の解析、および Hertley 変換や Hilbert 変換などの変換を実行する VI。

測定関数：片側スペクトル、基準化時間領域ウィンドウィング、ピーク電力と周波数推定など測定指向の関数を実行する VI。

デジタルフィルタ：IIR、FIR、非線形デジタルフィルタ関数を実行する VI。

ウィンドウ処理関数：データのウィンドウ処理を実行する VI。

確率関数と統計関数：一連のデータの平均値や標準偏差を求めるなどの記述統計関数や、確率や分散解析（ANOVA）用の推論的統計関数を実行する VI。

カーブフィット関数：カーブフィット関数や補間を実行する VI。

線形代数関数：実数と複素数のベクトルと行列用の線形関数を実行する VI。

配列操作：線形評価やスケーリングなどの一般的な一次元、二次元数値解析を実行する VI。

その他の数値式：平方根演算、数値積分、ピーク検出を行うための数値方式を使用する VI。

以下の項では、解析ライブラリの VI を使用して、関数発生器や簡単で実用的なスペクトルアナライザを作成する方法を学習します。さらに、デジタルフィルタの使用方法、ウィンドウ処理の目的とさまざまなタイプのウィンドウの利点、簡単なカーブフィットに関する作業の実行方法その他、さまざまな内容も学習します。各項の作業では LabVIEW/BridgeVIEW 開発システムが必要です。さらに詳しく知りたい方のためには、解析 VI の使用方法を広範囲に示すサンプルが、`labview¥examples¥analysis` フォルダに用意されています。

解析ライブラリの他にも、ナショナルインスツルメンツでは解析用のアドオンソフトウェアを多数用意しており、これらを使用することで、LabVIEW は現在入手できるものの中で最も強力な解析ソフトウェアパッケージのひとつとなります。このようなアドオンソフトウェアとしては、従来のフーリエ解析では容易に得られなかった時間周波数要素を解析するための、ナショナルインスツルメンツの賞に輝いた Gabor spectrogram アルゴリズムが含まれている **Joint Time-Frequency Analysis Toolkit**、フォーミュラパーサーと同様の拡張された数学機能を持つ **G Math Toolkit**、微分方程式を最適化したり解くためのルーチン、さまざまなタイプの 2 次元プロットや 3 次元プロット、**Digital Filter Design Toolkit**、その他さまざまなものがあります。

表記法および命名規約

パラメータや演算の種類を識別しやすくするため、VIの説明の中に特に指定がない限り、本書のこの部分では表記法と命名に関して以下の規約を使用します。いくつかのスカラ関数と演算がありますが、ほとんどの解析VIでは、一次元配列（またはベクトル）と二次元配列（または行列）の形で、大きなデータブロックを処理します。

通常の小文字はスカラまたは定数を表します。以下に例を示します。

$$a,$$
$$\pi,$$
$$b = 1.234.$$

次のように、大文字は配列を表します。以下に例を示します。

$$X,$$
$$A,$$
$$Y = a\,X + b.$$

一般に、X、Yは1次元配列を示し、A、B、Cは行列を示します。

LabVIEWでは配列の指標は0から始まります。配列内の最初の要素の指標は、配列の次元に関わらず0となります。次の数列は、n個の要素を含む1次元配列Xを示します。

$$X = \{x_0, x_1, x_2, ..., x_{n-1}\}$$

次のスカラ数量は、数列Xのi番目の要素を示します。

$$x_i, \quad 0 \le i < n$$

合計でn個の要素がある場合、数列内の最初の要素はx_0、最後の要素はx_{n-1}となります。

次の数列は、*n* 行 *m* 列の 2 次元配列を示します。

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} & \dots & a_{0m-1} \\ a_{10} & a_{11} & a_{12} & \dots & a_{1m-1} \\ a_{20} & a_{21} & a_{22} & \dots & a_{2m-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n-10} & a_{n-11} & a_{n-2} & \dots & a_{n-1m-1} \end{bmatrix}$$

2 次元配列に含まれる要素の合計数は、*n* と *m* の積になります。最初の指標は行に対応し、2 番目の指標は列に対応します。次のスカラ量は、第 *i* 行第 *j* 列にある要素を示します。

$$a_{ij}, 0 \le i < n \text{ と } 0 \le j < m$$

A の最初の要素は a_{00}、最後の要素は $a_{n-1\,m-1}$ になります。

特に指定のない限り、配列演算に関して次のような簡単な表記法を使用します。

配列の要素をスカラ定数に設定する場合は、次のように表します。

$$X = a$$

これは、次の数列に対応し、

$$X = \{a, a, a, \dots, a\}$$

次の式の代わりに使用されます。

$$x_i = a, \quad i = 0, 1, 2, \dots, n-1$$

配列の要素とスカラの積は、次のように表します。

$$Y = a\,X$$

これは、次の数列に対応し、

$$Y = \{a\,x_0, a\,x_1, a\,x_2, \ldots, a\,x_{n-1}\}$$

次の式の代わりに使用されます。

$$y_i = a\,x_i, \quad i = 0, 1, 2, \ldots, n-1$$

同様に、2次元配列とスカラ定数の積は、次のように表します。

$$B = k\,A$$

これは、次の数列に対応し、

$$B = \begin{bmatrix} ka_{00} & ka_{01} & ka_{02} & \ldots & ka_{0m-1} \\ ka_{10} & ka_{11} & ka_{12} & \ldots & ka_{1m-1} \\ ka_{20} & ka_{21} & ka_{22} & \ldots & ka_{2m-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ ka_{n-10} & ka_{n-11} & ka_{n-12} & \ldots & ka_{n-1m-1} \end{bmatrix}$$

次の式の代わりに使用されます。

$$b_{ij} = k\,a_{ij}, \quad i = 0, 1, 2, \ldots, n-1, \quad j = 0, 1, 2, \ldots, m-1$$

要素のない配列を空の配列と言い、次のように示します。

$$\text{Empty} = \text{NULL} = \emptyset = \{\ \}$$

一般に、空の配列に対して演算を行うと、空の出力配列または不定の結果が得られます。

データのサンプリング

信号のサンプリング

デジタル信号処理技術を使用するためには、まずアナログ信号をデジタル表記に変換する必要があります。実際には、この処理は A/D 変換器によって行われます。Δt 秒ごとにサンプリングされるアナログ信号 $x(t)$ の場合を考えてみましょう。時間間隔 Δt は、**サンプリング間隔**または**サンプリング周期**と呼ばれます。その逆数 $1/\Delta t$ は**サンプリング周波数**と呼ばれ、単位はサンプル／秒になります。t = 0, Δt, $2\Delta t$, $3\Delta t$ における $x(t)$ の離散的なそれぞれの値のことを**サンプル**と言います。つまり、x(0), x(Δt), x($2\Delta t$), ... はいずれもサンプルです。したがって、信号 $x(t)$ は、次のような離散的な一連のサンプルで示すことができます。

$$\{x(0), x(\Delta t), x(2\Delta t), x(3\Delta t), \ldots, x(k\Delta t), \ldots\}$$

図 11-1 は、アナログ信号と、アナログ信号をサンプリングしたものを示します。サンプリング間隔は Δt です。サンプルが時間軸に沿った離散的な点で定義されていることに注意してください。

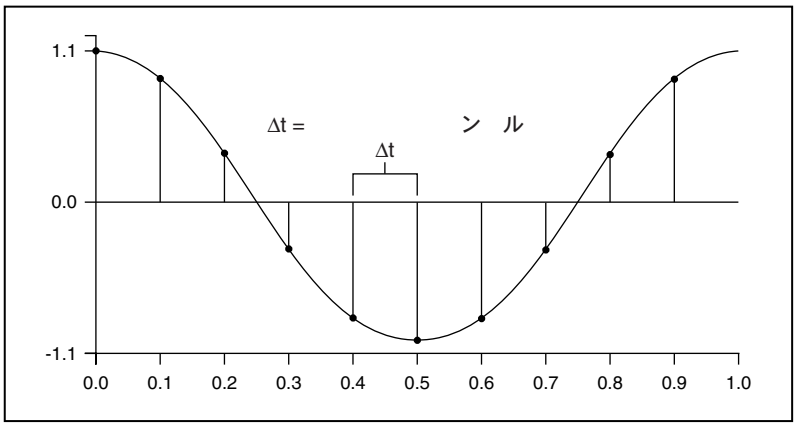


図 11-1 アナログ信号、およびサンプルバージョン

次の表現法は、個々のサンプルを示します。

$$x[i] = x(i\Delta t), \quad i = 0, 1, 2, \ldots$$

信号 $x(t)$ から N 個のサンプルが得られる場合、$x(t)$ は次の数列で示すことができます。

$$X = \{x[0], x[1], x[2], x[3], \ldots, x[N-1]\}$$

これを、$x(t)$ の**デジタル表記**または**サンプルバージョン**と言います。ここで、数列 $X = \{x[i]\}$ は整数変数 i により指標付けされていますが、サンプリングレートについての情報は全く含まれていないことに注意してください。したがって、X に含まれるサンプルの値がわかっても、それだけではサンプリングレートについては何もわかりません。

サンプリングについての考察

A/D 変換器（ADC）は、ナショナルインスツルメンツの DAQ ボードに不可欠な部分です。アナログ入力システムの最も重要なパラメータの一つに、DAQ ボードが入力信号からサンプリングする速度があります。アナログからデジタルへの（A/D）変換が行われる頻度は、サンプリグレートによって決まります。サンプリングレートを速くすると、一定の時間内でより多くの点を集録できるため、多くの場合は、サンプリングレートが遅い場合に比べて元の信号をより正確に表現できます。サンプリング速度が遅すぎると、アナログ信号を表現する精度が低下する恐れがあります。図 11-2 は、適切なサンプリングが行われた場合と、サンプリング数が少なすぎた場合の影響を示します。サンプリング数が少なすぎると、実際のものとは異なる周波数の信号のようになります。このように信号が誤って表現されてしまうことを**エイリアス**と言います。

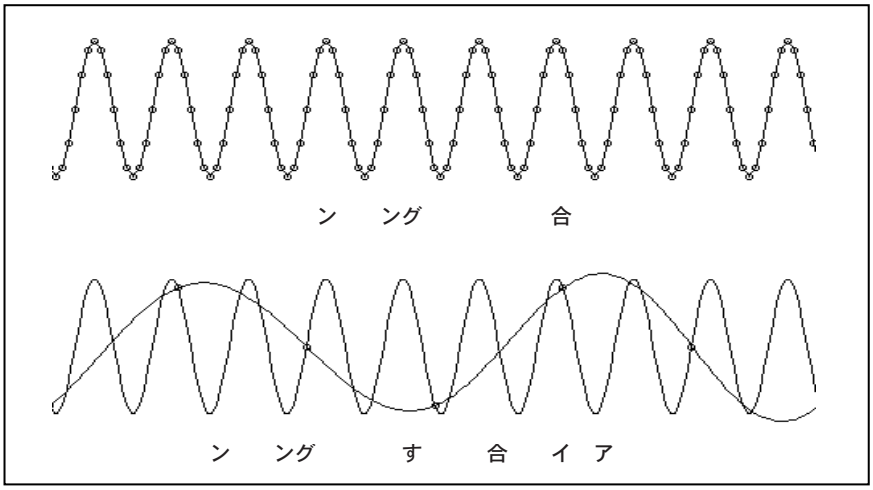


図 11-2 不適切なサンプリングレートによるエイリアスの影響

ナイキスト定理によると、エイリアスを防止するためには、集録する信号に含まれる最大周波数成分の 2 倍以上の速度でサンプリングを行う必要があります。ある一定のサンプリングレートで、エイリアスを起こさずに正確に表現できる最大周波数のことを、ナイキスト周波数と言います。ナイキスト周波数は、サンプリング周波数の 1/2 になります。信号の中にナイキスト周波数を超える周波数成分があると、エイリアスとなって DC とナイキスト周波数の間に現れます。エイリアス周波数は、エイリアスとなって入力信号の周波数とサンプリングレートの整数倍に最も近い値との差の絶対値です。図 11-3 と図 11-4 に、この現象を示します。たとえば、サンプリング周波数 *fs* が 100 Hz で、入力信号に 25 Hz、70 Hz、160 Hz、510 Hz の周波数が含まれている場合、これらの周波数は図11-3のようになります。

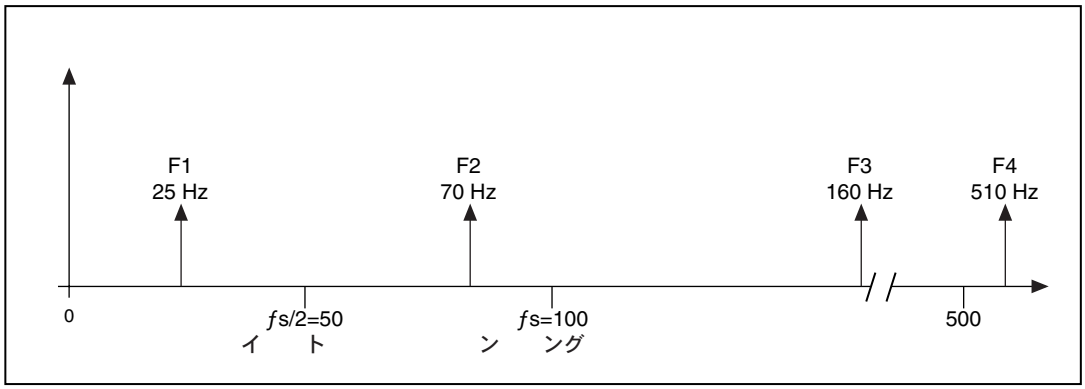


図 11-3 実際の信号の周波数成分

図 11-4 では、ナイキスト周波数（*fs*/2 = 50 Hz）未満の周波数は正しくサンプリングされることがわかります。ナイキスト周波数を超える周波数はエイリアスとして現れます。たとえば、F1（25 Hz）は正しい周波数で表現されますが、F2（70 Hz）、F3（160 Hz）、F4（510 Hz）はそれぞれ 30 Hz、40 Hz、10 Hz のエイリアスとなります。エイリアス周波数を計算するには次の式を使用します。

エイリアス周波数＝ ABS（サンプリング周波数の整数倍に
最も近い周波数 – 入力周波数）

ABS は「絶対値」を意味します。たとえば、

F2 のエイリアス = |100 – 70| = 30 Hz
F3 のエイリアス = |(2)100 – 160| = 40 Hz
F4 のエイリアス = |(5)100 – 510| = 10 Hz

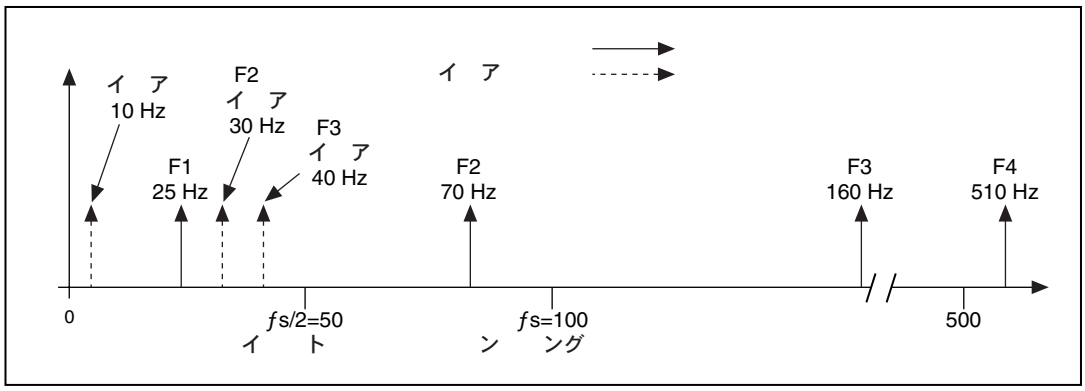


図 11-4 信号の周波数成分とエイリアス

よく質問をされるのが、「どの位のスピードでサンプリングをしたらよいか」という問題です。最初は普通、自分の DAQ ボードで可能な最大レートでサンプリングしようと考えます。しかし、非常に長時間にわたり高速なサンプリングを行うと、データを格納するためのメモリやハードディスク領域が足りなくなってしまいます。図 11-5 に、さまざまなサンプリングレートによる違いを示します。A の場合は、周波数 f の正弦波を、fs（サンプル／秒）= f（周期／秒）という同じ周波数で、つまり 1 周期で 1 サンプルずつサンプリングしています。再生される波形は、DC のエイリアスとなります。B のようにサンプリングの数を 7 サンプル／4 サイクルに増やすと、波形の周波数が高くはなりますが元の信号より周波数の低い（4 サイクルでなく 3 サイクルの）エイリアスとなります。B の場合のサンプリングレートは fs = 7/4f です。C の場合のようにサンプリングレートを fs = 2f まで大きくすると、デジタル化された波形は正しい周波数（同じ周期数）になり、元のものと同じ正弦波として再生することができます。時間領域

で処理する場合に、サンプルでより元の信号に近い値を表現できるようにするには、サンプリングレートを大きくすることが重要になります。D の場合のように、サンプリングレートを f より十分大きい値（たとえば fs = 10f または 10 サンプル／周期）にすると、波形を正確に再生できます。

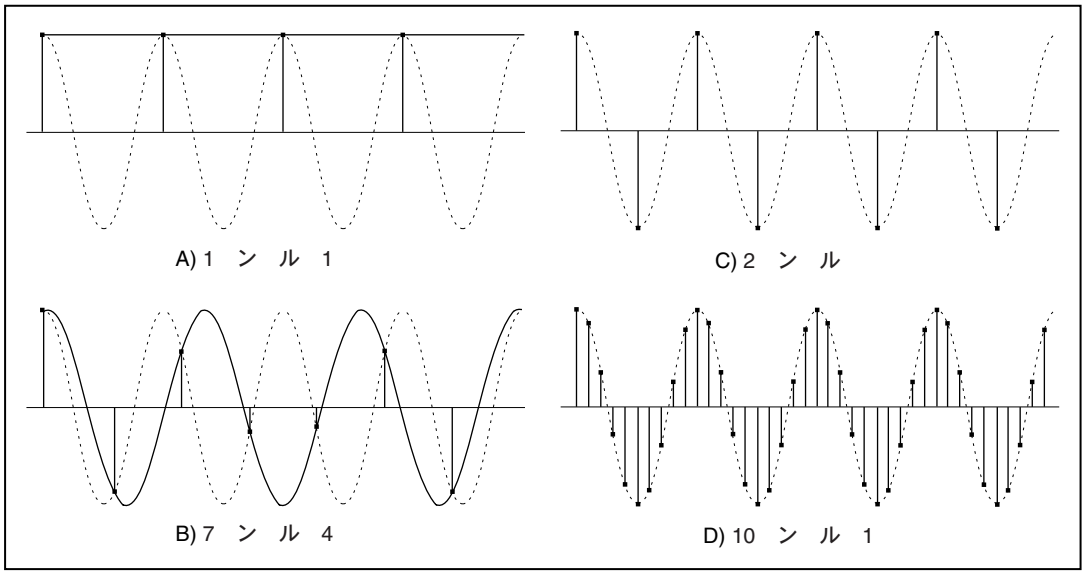


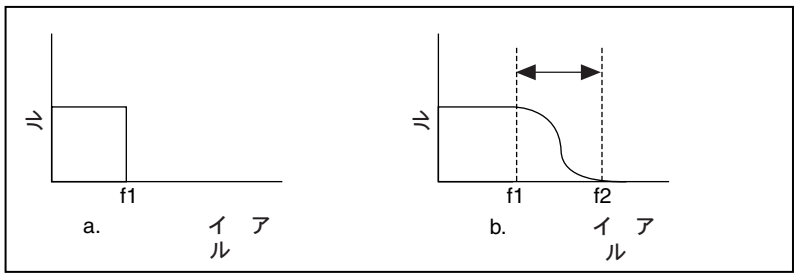
図 11-5 さまざまなレートでサンプリングを行った場合の違い

エイリアス防止フィルタが必要な理由

これまで、サンプリングレートは、サンプリングする信号の最大周波数の 2 倍以上でなければならないことがわかりました。言い換えると、入力信号の最大周波数はサンプリングレートの半分以下でなければなりません。しかし、実際に確実にこうなるようにするには、どうしたらよいでしょうか？　測定対象となる信号の周波数には上限があることがわかっていたとしても、(送電線周波数やローカルラジオ局などの) ストレー信号から収集したものの中にナイキスト周波数より高い周波数が含まれていることがあり得ます。このような周波数があると、望ましい周波数範囲の中にそのエイリアスが混入し、誤った結果となってしまう恐れがあります。

入力信号の周波数成分の範囲を確実に制限するため、サンプラと ADC の前にローパスフィルタ（低い周波数を通し高い周波数を減衰させるフィルタ）を追加します。このフィルタは（ナイキスト周波数より）高い周波数を減衰させることでエイリアス成分がサンプリングされるのを防止するので、**エイリアス防止**フィルタと呼ばれます。この段階で（サンプラと ADC の前）はまだアナログ信号ですので、エイリアス防止フィルタはアナログフィルタです。

理想的なエイリアス防止フィルタを、次の図の (a) に示します。



このフィルタは、必要なすべての（*f1* 未満の）入力周波数を通し、不要なすべての（*f1* を超える）周波数を削除します。ただし、このようなフィルタを物理的に実現することは不可能です。実際のフィルタは、上図の (b) のようになります。このフィルタは、*f1* 未満のすべての周波数を通し、*f2* を超えるすべての周波数を削除します。*f1* と *f2* の間の領域を**過渡周波数帯域**と言い、この領域では入力信号が次第に減衰していきます。周波数が *f1* 未満の信号だけを通したいのですが、過渡周波数帯域内の信号がエイリアスの原因となることがあります。このため実際には、サンプリング周波数を、過渡周波数帯域の最高周波数の 2 倍以上にする必要があります。したがって、この周波数は最大入力周波数（*f1*）の 2 倍よりも大きくなります。これが、サンプリングレートを最大入力周波数の 2 倍より大きくする一つの理由です。設計するフィルタのタイプによるフィルタの過渡周波数帯域の違いについては、後の項で説明します。

デシベルを使用する理由

一部の計測器では、振幅表示の目盛りを線形にするかデシベル（dB）にするかを選択できます。線形目盛りでは振幅がそのまま表示されますが、デシベル目盛りでは線形目盛りが対数関数的な目盛りに変換されます。このような変換が必要になる理由を次に説明します。

振幅が非常に大きい部分や非常に小さい部分を含む信号を表示する場合を考えてみます。表示部分の高さが 10cm で、最大振幅のときに表示部の高さ全体を使用するとします。この場合、信号の最大振幅が 100V だとすると、表示上の 1cm の高さは 10V に対応します。信号の最小振幅が 0.1V の場合、その高さは 0.1mm になってしまい、ディスプレイ上ではほとんど認識できません。

最大の場合から最小の場合まですべての振幅を確認できるようにするには、振幅目盛りを変更する必要があります。アレクサンダー・グラハム・ベルは、大きな振幅を圧縮し小さな振幅を拡大する対数関数的なベルという単位を考案しました。が、ベルと言う単位は大きすぎるため、通常はデ

シベル（ベルの 1/10）という単位が使用されます。デシベルは次のように定義されます。

$$1\ \mathrm{dB} = 10\log_{10}（電力の比）= 20\log_{10}（電圧の比）$$

次の表に、デシベルと電力比、電圧比の関係を示します。

dB	電力比	電圧比
+40	10000	100
+20	100	10
+6	4	2
+3	2	1.4
0	1	1
-3	1/2	1/1.4
-6	1/4	1/2
-20	1/100	1/10
-40	1/10000	1/100

このように、広い範囲の振幅を小さい範囲の数値に圧縮する場合に dB が有効であることがおわかりいただけたでしょう。デシベル目盛りは、音響や振動の測定、周波数領域の情報の表示などによく使用されます。

12

信号生成

この章では、正規化周波数を使用して信号を生成する方法、およびシミュレーションにより関数発生器を作成する方法を説明します。信号生成VIの使用方法については、examples¥analysis¥sigxmpl.llb の中にある例を参照してください。

ここでは、解析ライブラリに含まれるVIを使用してさまざまな信号を生成する方法を学習します。信号生成の適用例には、次のようなものがあります。

- 実際の信号を使用できない場合にユーザのアルゴリズムをテストするための信号をシミュレーションする（たとえば実際の信号を取り込むためのDAQボードがない場合や、実際の信号にアクセスできない場合など）。
- D/A変換器に加える信号を生成する。

正規化周波数

アナログの世界では、信号の周波数は、Hzまたは1秒あたりのサイクル数を単位として計測されますが、デジタルシステムでは、アナログ周波数とサンプリング周波数の比であるデジタル周波数がよく使用されます。

デジタル周波数＝アナログ周波数／サンプリング周波数

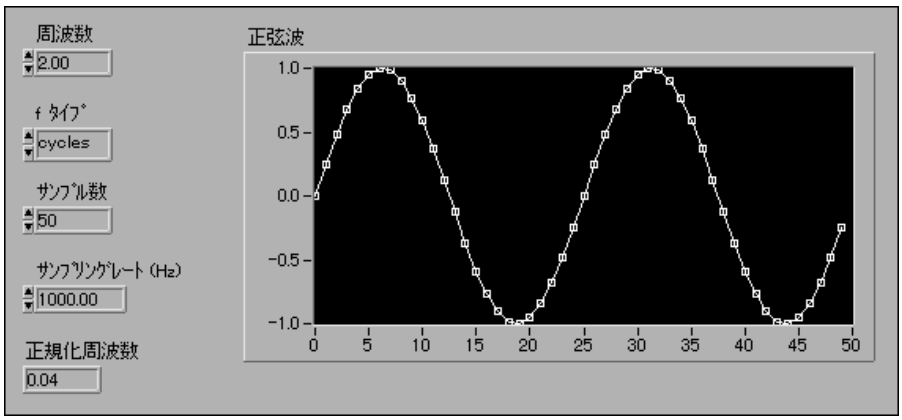
このデジタル周波数を**正規化周波数**と言い、単位は周期／サンプルです。

信号生成VIの中には、入力周波数制御器*f*を使用するものがあり、**サイクル／サンプル**を単位とする**正規化周波数**が使用されると考えられています。この周波数は0.0～1.0で、0からサンプリング周波数*fs*までの実際の周波数範囲に対応します。この周波数は1.0になると0に戻るため、正規化周波数が1.1の場合は0.1と同じになります。たとえば、ナイキスト周波数（*fs/2*）でサンプリングされた信号は、1サイクルに2回ずつ（すなわち2サンプル／サイクル）サンプリングされていることになります。これは、正規化周波数では1/2サイクル／サンプル＝0.5サイクル／サンプルに相当します。正規化周波数の逆数（*1/f*）は、信号が1サイクルにサンプリグされる回数です。

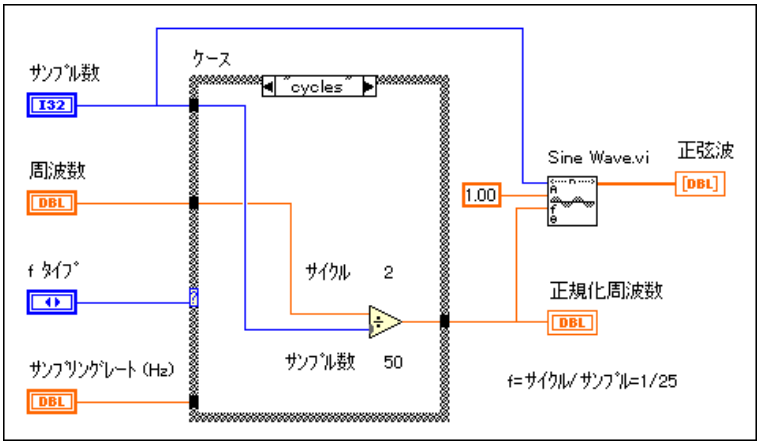
入力に正規化周波数を必要とするVIを使用する場合は、周波数の単位を正規化単位であるサイクル／サンプルに変換しなければなりません。以下のVIには、この正規化単位を使用する必要があります。

- 正弦波
- 方形波
- ノコギリ波
- 三角波
- 任意の波動
- チャープパターン

周波数の単位としてサイクルを使用するのに慣れている場合は、サイクル数を生成されるサンプル数で割るとサイクル／サンプルが求められます。次の図は、Sine Wave VIを使用して2サイクル分の正弦波を生成している様子を示します。



次の図は、サイクルをサイクル／サンプルに変換するためのブロックダイアグラムを示します。

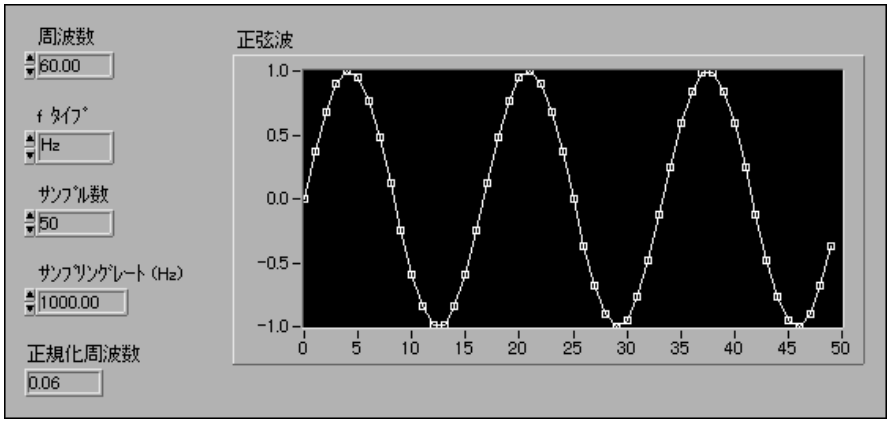


（サイクル単位の）周波数をサンプル数で割るだけで OK です。上の例では、2 サイクルという周波数を 50 個というサンプル数で割って、*f* = 1/25 サイクル／サンプルという正規化周波数が得られます。これは、1 サイクルの正弦波を生成するのに、25 個の（*f* の逆数）サンプルが取り込まれることを意味します。

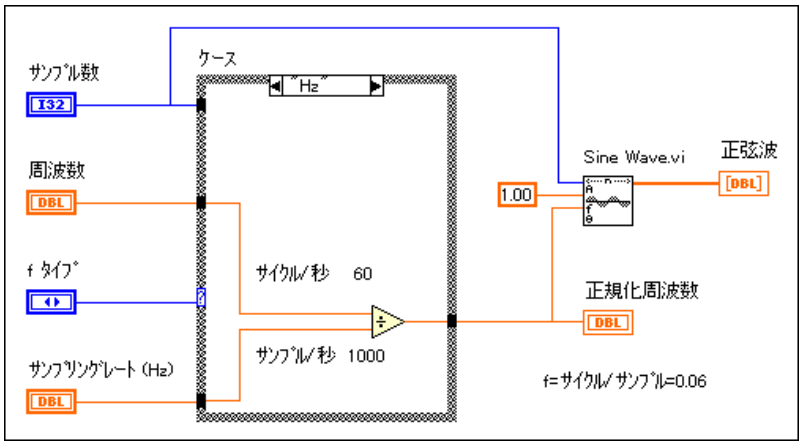
ただし、周波数単位 Hz（サイクル／秒）を使用しなければならない場合もあります。Hz（サイクル／秒）を（サイクル／サンプル）に変換する必要がある場合は、サイクル／秒の周波数を、サンプル／秒のサンプリングレートで割ってください。

$$\frac{\text{サイクル／秒}}{\text{サンプル／秒}} = \frac{\text{サイクル}}{\text{サンプル}}$$

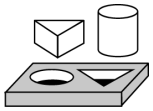
次の図は、Sine Wave VIを使用して60 Hzの正弦波を生成している様子を示します。



次の図は、Hz単位で正弦波信号を生成するためのブロックダイアグラムを示します。周波数60 Hzをサンプリングレート1000 Hzで割ると、**正規化**周波数f = 0.06サイクル／サンプルが得られます。したがって、この場合は1サイクルの正弦波を生成するのに約17個（1/0.06）のサンプルが取り込まれます。



信号生成VIは、ネットワーク解析やシミュレーションに必要な多くの共通の信号を生成します。また、信号生成VIをナショナルインスツルメンツのハードウェアとともに使用して、アナログ出力信号を生成することもできます。

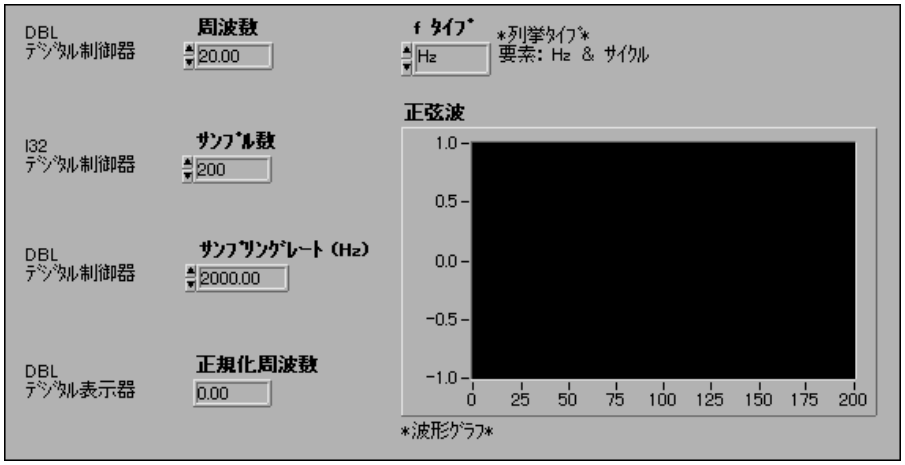


作業 12-1.　正規化周波数についてさらに詳しく学習する

ここでは、周波数、サンプリングレート、サンプル数を調整した場合の正弦波への効果を観測することで、正規化周波数についてさらに詳しく学習することが目的です。

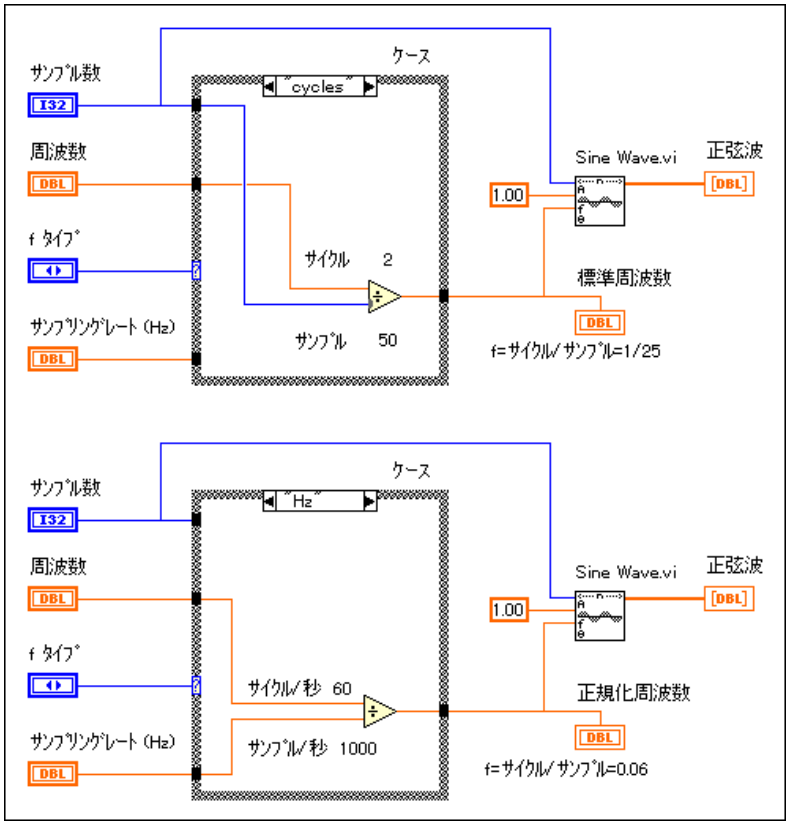
フロントパネル

1. 新しいフロントパネルを開き、次のようなオブジェクトを作成します。



ブロックダイアグラム

2. 次のようなブロックダイアグラムを作成します。

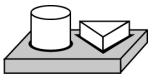


Sine Wave VI（**解析→信号生成**パレット）。

3. このVIを、`Normalized Frequency.vi`という名前で`LabVIEW¥Activity`ディレクトリに保存します。
4. 周波数に2サイクルを（**周波数** = 2、**fタイプ** = cycles）、**サンプル数**に100を選択し、VIを実行します。2サイクルのプロットが表示されることを確認してください。
5. **サンプル数**を150、200、250に増やすと、表示されるサイクル数はどうなりますか？
6. 次に、**サンプル数**を100にしたまま、サイクル数を3、4、5に増やします。表示されるサイクル数はどうなりますか？

このように、サイクルによって周波数を選択すると、プロット上に表示される入力波形のサイクル数が多くなります。この場合に、サンプリングレートは無関係であることに注意してください。

7. **fタイプ**をHzに、**サンプリングレート**（Hz）を1,000に変更します。
8. **サンプル数**を100に固定したまま、**周波数**を10、20、30、40に変更します。それぞれの場合にプロット上に表示される波形のサイクル数を確認し、観察結果について説明してください。
9. **周波数**を10に固定したまま**サンプル数**を100、200、300、400に変更して上記のステップを繰り返します。それぞれの場合にプロット上に表示される波形のサイクル数を確認し、観察結果について説明してください。
10. **周波数**を20に、**サンプル数**を200に固定したまま、**サンプリングレート（Hz）**を500、1,000、2,000に変更し、結果をよく理解できるか確認してください。



これで作業 12-1 は完了です。

波形VIとパターンVI

信号生成VIのほとんどは、その名前に**wave（波形）**または**pattern（パターン）**という語が含まれていることにお気づきでしょう。これら2つの異なるタイプのVIの動作には、基本的な違いがあります。VIを呼び出すたびに、VIの生成する信号の位相をVIが追跡できるかできないかという点です。

位相制御

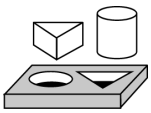
波形VIには**位相入力**制御器があり、これを使用すると、生成される波形の第1サンプルの初期位相（単位は度）を指定することができます。また、波形VIには、生成される波形の次のサンプルの位相を指定するための**位相出力**表示器もあります。**位相をリセット**制御器は、**波形**VIが呼び出されるときに生成される第1サンプルの位相が、**位相入力**制御器で指定された位相かどうか、あるいはVIが最後に実行されたときに**位相出力**表示器で使用可能な位相かどうかを決定します。**位相をリセット**の値をTRUEにセットすると初期位相が**位相入力**の値にセットされ、FALSEにセットすると最後にVIが実行されたときの**位相出力**の値にセットされます。

波形VIはすべて再入実行（内部で位相を追跡記憶）可能であり、正規化単位（サイクル／サンプル）で表現された周波数を受け取ります。現在のところ、正規化単位を使用する**パターン** VI は Chirp Pattern VI のみです。

ブール値**位相をリセット**を FALSE にセットすると、連続的なサンプリングシミュレーションが可能になります。

☞ **注**　　**波形 VI は再入実行であり、正規化単位による周波数を受け取ります。**

次の作業では、Sine Wave VI と Sine Pattern VI の両方を使用して正弦波を生成します。Sine Pattern VI に比べて Sine Wave VI の方がどの程度詳細に初期位相を制御できるかを確認します。

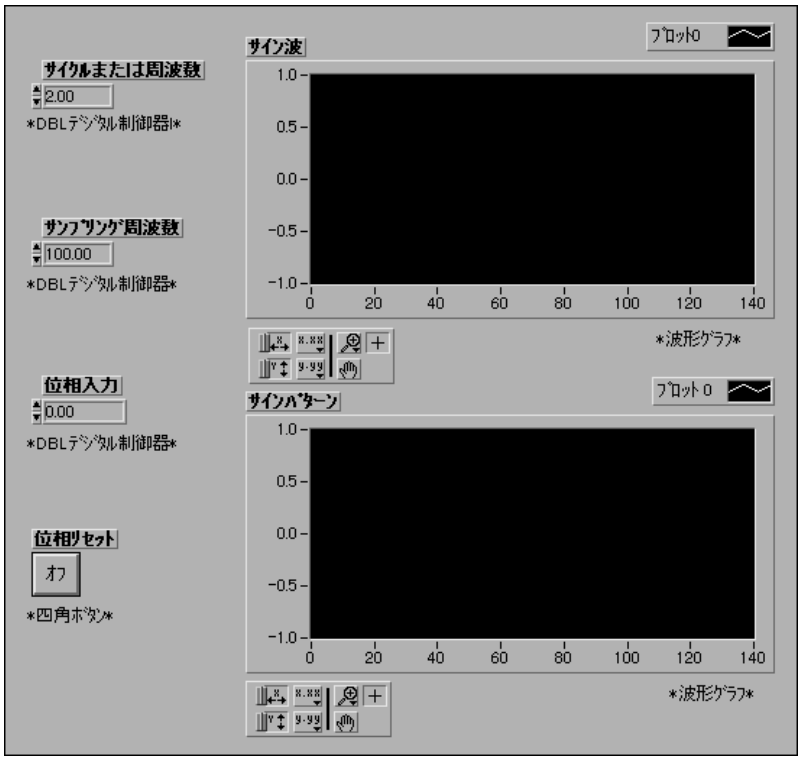


作業 12-2.　Sine Wave VI と Sine Pattern VI を使用する

ここでは、Sine Wave VI と Sine Pattern VI の両方を使用して正弦波波形を生成し、両者の違いを理解することが目的です。

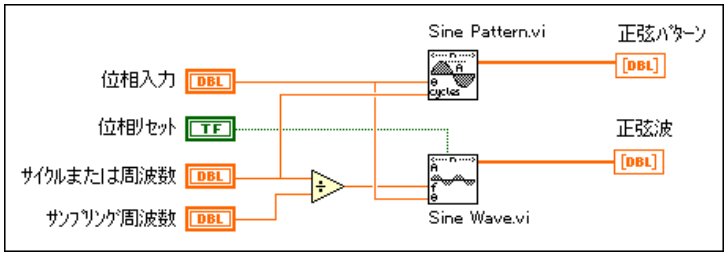
フロントパネル

1. 新しいフロントパネルを開き、次の図のようなオブジェクトを作成します。



ブロックダイアグラム

2. 次の図のようなブロックダイアグラムを作成します。



Sine Pattern VI（**解析→信号生成**パレット）

Sine Wave VI（**解析→信号生成**パレット）

3. このVIを`Wave and Pattern.vi`という名前で`LabVIEW¥Activity`ディレクトリに保存します。
4. 制御器を以下の値に設定します。

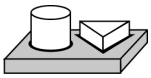
サイクルまたは周波数：	2.00
サンプリング周波数：	100
位相入力：	0.00
位相をリセット：	オフ

VIを数回実行します。

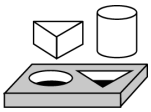
VIを実行するたびに**正弦波**のプロットが変わることを確認してください。**位相をリセット**がオフに設定されているため、正弦波の位相はVIを呼び出すたびに変化し、前回の呼び出し時の**位相出力**の値に等しくなります。ただし、正弦波パターンプロットは常に同じままで変わらず、2サイクルの正弦波波形が表示されます。正弦波パターンプロットの初期位相は、**位相入力**制御器に設定された値に等しくなります。

☞ **注**　　**位相入力と位相出力は度単位で指定されます。**

5. **位相入力**を90に変更してVIを数回実行します。前と同様、VIを実行するたびに正弦波プロットは変化します。ただし、正弦波パターンプロットは変化せず、正弦波パターンの初期位相が**位相入力**制御器で指定された値と同じ90度になります。
6. **位相入力**を90にしたままで、**位相をリセット**をONにし、VIを数回実行します。正弦波プロットと正弦波パターンプロットで表示される正弦波波形は90度から始まりますが、VIに対するそれ以後の呼び出しでは変わりません。
7. **位相をリセット**をONにしたままで、**位相入力**を45、180、270、360の各値にしてVIを数回実行します。VIを実行するたびに、生成される波形の初期位相を確認してください。



これで作業 12-2 は完了です。



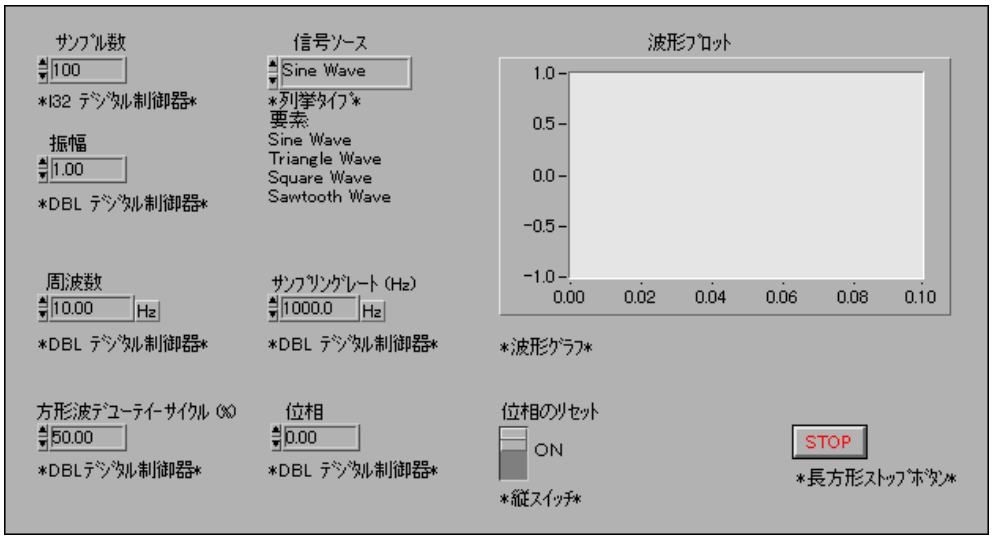
作業 12-3.　波形発生器を作成する

ここでは、以下の波形を生成する簡単な波形発生器を作成することが目的です。

- 正弦波
- 方形波
- 三角波
- ノコギリ波

フロントパネル

1. 新しいフロントパネルを開き、次の図のようなオブジェクトを作成します。



信号ソース制御器では、生成する波形のタイプを選択します。

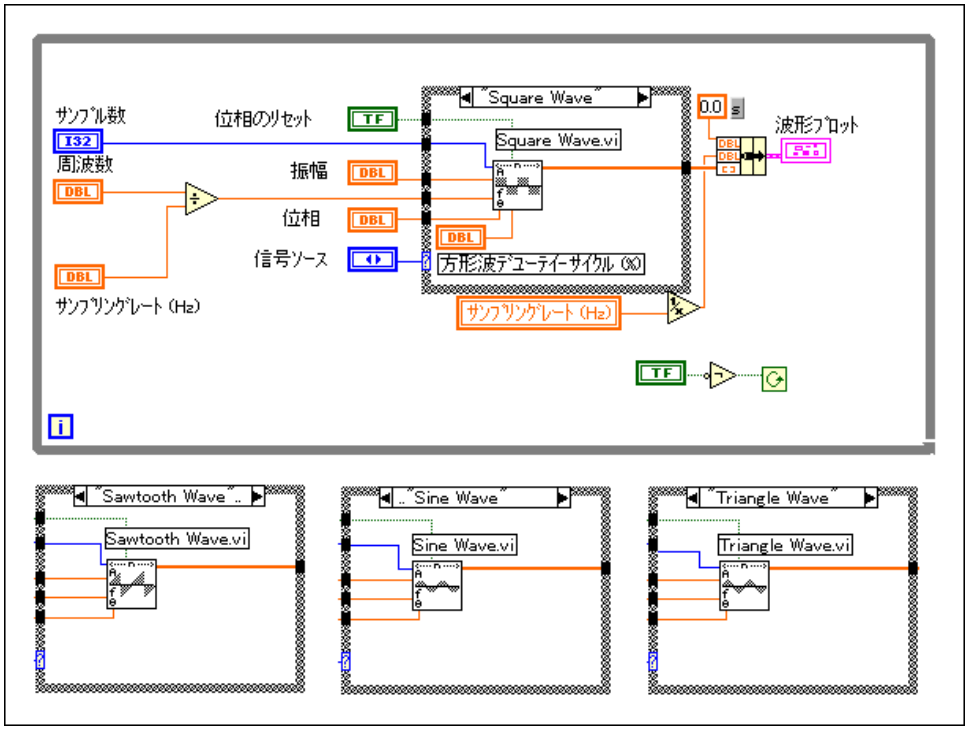
方形波デューティサイクル制御器は、方形波のデューティサイクルの設定のみに使用されます。

サンプル数制御器は、プロットに含まれるサンプル数を決定します。

これらはいずれも**波形**VIですので、周波数入力は正規化周波数でなければならないことに注意してください。したがって、**周波数**を**サンプリングレート**で割った結果である正規化周波数をVIの*f*入力に配線します。

ブロックダイアグラム

2. 次の図のようなブロックダイアグラムを作成します。



Sine Wave VI（**解析→信号生成**パレット）は、正規化周波数*f*の正弦波を生成します。

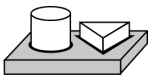
Triange Wave VI（**解析→信号生成**パレット）は、正規化周波数*f*の三角波を生成します。

Square Wave VI（**解析→信号生成**パレット）は、正規化周波数*f*の方形波を指定されたデューティサイクルで生成します。

Sawtooth Wave VI（**解析→信号生成**パレット）は、正規化周波数*f*のノコギリ波を生成します。

3. このVIを`Function Generator.vi`という名前で`LabVIEW¥Activity`ディレクトリに保存します。

4. **サンプリングレート** = 1000 Hz、**振幅** = 1、**サンプル数** = 100、**周波数** = 10、**位相をリセット** = ON、**信号ソース** = sine wave を選択します。**サンプリングレート** = 1000 で**周波数** = 10 ですので、各 100 サンプルが 1 サイクルに対応します。
5. VI を実行し、結果として得られるプロットの変化を確認します。
6. **サンプル数**を 200、300、400 に変更して、表示される波形のサイクル数を確認し、理由を説明してください。
7. **サンプル数**を 100 に設定した状態で**位相をリセット**をオフにし、プロットが変わるか、確認してください。
8. **周波数**を 10.01 Hz に変更し、どうなるか確認し、理由を説明してください。
9. **位相をリセット**をオンに変更し、どうなるか確認し、理由を説明してください。
10. **信号ソース**制御器でさまざまな波形を選択して、ステップ 4 ～ 9 を繰り返してください。



これで作業 12-3 は完了です。

13

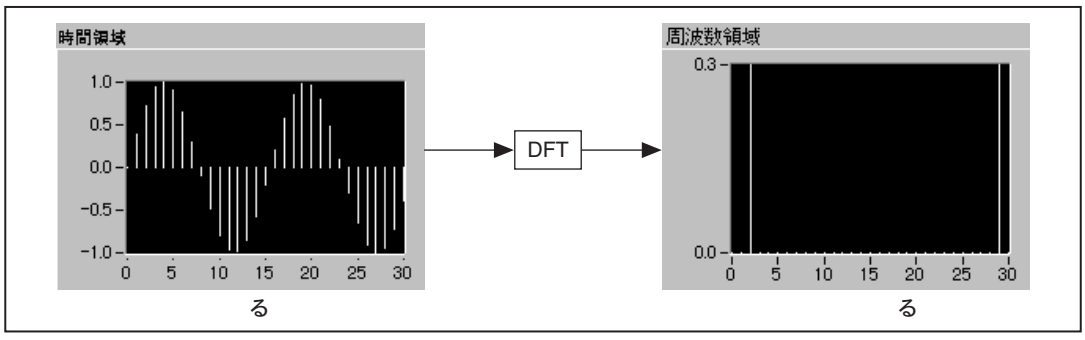
デジタル信号処理

この章は、高速フーリエ変換（Fast Fourier Transform: FFT）と離散フーリエ変換（Discrete Fourier Transform: DFT）の基礎、およびこれらをスペクトル解析に使用する方法を説明します。デジタル信号処理 VI の使用方法については、`exmples¥analysis¥dspxmpl.llb` の例を参照してください。

高速フーリエ変換（FFT）

DAQ ボードから得られた信号のサンプルは、信号を**時間領域**で表現したものです。この表現法では、サンプリングした**瞬間**における信号の振幅がわかります。しかし、個々のサンプルの振幅ではなく信号の周波数成分を知りたい場合も多くあります。信号の個々の周波数成分による表現のことを、信号の**周波数領域**による表現と言います。場合によっては、周波数領域による表現方法を使用した方が、信号や、その信号を生成したシステムについてより深く洞察できます。

時間領域から周波数領域にデータサンプルを変換するのに使用されるアルゴリズムは、**離散フーリエ変換**または DFT と呼ばれます。DFT は、時間領域による信号のサンプルと周波数領域におけるこれらの表現法との間の関係を確立するものです。DFT は、スペクトル解析、応用機械工学、音響学、医学用画像処理、数量解析、計測、遠隔通信などの分野で広く使用されています。



DAQ ボードから信号のサンプルが *N* 個得られたと仮定します。この時間領域の表現による *N* 個のサンプルに対して DFT を適用すると、結果としてやはり長さが *N* 個のサンプルが得られますが、これには周波数領域の表現に

よる情報が含まれています。時間領域におけるN個のサンプルと周波数領域における*N*個のサンプルとの間の関係について、次に説明します。

fs Hz のサンプリングレートで信号をサンプリングした場合、サンプル間の時間的な間隔（サンプリング周期）はΔtで次のようになります。

$$\Delta t = \frac{1}{f_s}$$

サンプル信号はx[i], 0 ≤ i ≤ *N*–1 で表されます（サンプル数は合計で*N*個）。次の式

$$X_k = \sum_{i=0}^{N-1} x_i e^{-j2\pi ik/N} \qquad (13\text{-}1)$$

で示される離散フーリエ解析を、*k*を次の値

$$k = 0, 1, 2, \ldots, N-1$$

にしてこれらの*N*個のサンプルに適用すると、結果として得られる出力（*X*[k], 0 ≤ k ≤ N–1）は、*x*[i] を周波数領域で表現したものになります。時間領域の*x*と周波数領域の*X*のいずれにも、合計で*N*個のサンプルがあることに注意してください。時間領域の*x*のサンプル間の**時間間隔**Δtに対応して、周波数領域の*X*の各成分の間には、次の式で示される周波数間隔があります。

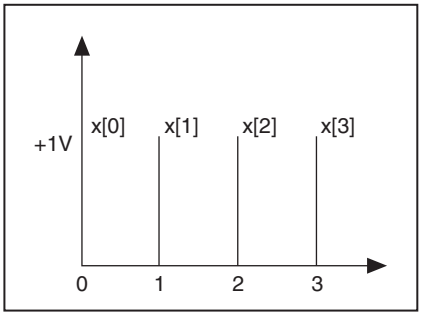
$$\Delta f = \frac{f_s}{N} = \frac{1}{N\Delta t}$$

このΔfは、周波数分解能とも言われます。周波数分解能を上げる（Δfを小さくする）には、サンプル数*N*を大きくする（*fs*が一定の場合）か、サンプリング周波数*fs*を小さくする（*N*が一定の場合）必要があります。

次の例では、式13-1に従って、DC信号に対するDFTを計算します。

DFTの計算例

次の項では、DFTの*N*個のサンプルに対応する正確な周波数について示します。ここでは、*X*[0] が信号のDCまたは平均値に対応するものと仮定します。式13-1を使用して波形のDFTを計算した結果を見るため、振幅が+1Vで一定のDC信号の場合を考えます。この信号の4個のサンプルを、次の図に示します。



各サンプルの値は+1であり、次のような時系列が得られます。

$$x[0] = x[1] = x[3] = x[4] = 1$$

式13-1を使用してこの時系列のDFTを計算し、次に示すオイラーの公式

$$\exp(-i\theta) = \cos(\theta) - j\sin(\theta)$$

を使用すると、次のようになります。

$$X[0] = \sum_{i=0}^{N-1} x_i e^{-j2\pi i 0/N} = x[0] + x[1] + x[2] + x[3] = 4$$

$$X[1] = x[0] + x[1]\left(\cos\left(\frac{\pi}{2}\right) - j\sin\left(\frac{\pi}{2}\right)\right) + x[2](\cos(\pi) - j\sin(\pi)) + x[3]\left(\cos\left(\frac{3\pi}{2}\right) - j\sin\left(\frac{3\pi}{2}\right)\right) = (1 - j - 1 + j) = 0$$

$$X[2] = x[0] + x[1](\cos(\pi) - j\sin(\pi)) + x[2](\cos(2\pi) - j\sin(2\pi)) + x[3](\cos(3\pi) - j\sin(3\pi)) = (1 - 1 + 1 - 1) = 0$$

$$X[3] = x[0] + x[1]\left(\cos\left(\frac{3\pi}{2}\right) - j\sin\left(\frac{3\pi}{2}\right)\right) + x[2](\cos(3\pi) - j\sin(3\pi)) + x[3]\left(\cos\left(\frac{9\pi}{2}\right) - j\sin\left(\frac{9\pi}{2}\right)\right) = (1 - j - 1 - j) = 0$$

したがって、DC成分を除く$X[0]$の値はすべて0になり、予測と一致します。しかし、計算された$X[0]$の値は、N（サンプル数）の値によって異なります。ここでは$N = 4$であったため、$X[0] = 4$になりました。$N = 10$の場合の計算値は$X[0] = 10$になります。このように$X[\ ]$がNによって異なるという現象は、他の周波数成分の場合にも起こります。したがって、正しい周波数成分の大きさの値を得るためには、通常、DFT出力をNで割ります。

振幅情報と位相情報

入力信号の*N*個のサンプルがDFTの*N*個のサンプルになることがわかりました。つまり、時間による表現と周波数による表現ではサンプル数が同じになります。式13-1から、入力信号*x*[i]が実数であっても複素数であっても、*X*[k]は必ず複素数になることがわかります（虚部が0になる場合もあります）。このようにDFTは複素数であり、振幅と位相という2つの情報が含まれます。DAQボードの1つのチャネルの出力から得られる信号のような実際の信号（*x*[i]の実部）の場合、DFTは、

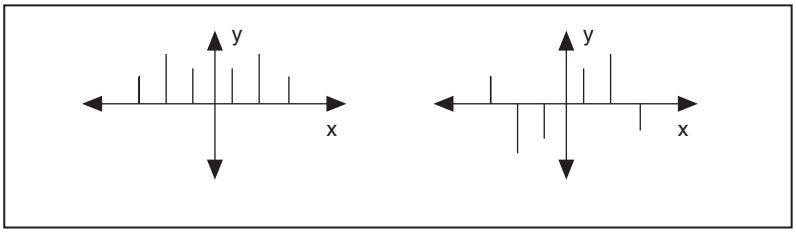
$$| X[k] | = | X[N-k] |$$

および

$$\text{phase}(X[k]) = -\text{phase}(X[N-k])$$

のように対称になることがわかります。

この対称性を説明するために、X[k]の振幅については**偶対称**、phase(X[k])については**奇対称**という用語が使用されます。偶対称の信号とはY軸について対称な信号であり、奇対称な信号とは原点について対処な信号です。この関係を、次の図に示します。



この対称関係による直接的な影響は、DFTの*N*個のサンプルに情報の反復が含まれることです。このような情報の反復があるため、実際に計算して表示する必要があるのはDFTのサンプルの半分だけであり、残りの半分はこのような反復によって得られます。

☞ **注**　**入力信号が複素数である場合はDFTが非対称になるため、この技法は使用できません。**

DFT/FFTのサンプル間の周波数間隔

サンプル周期がΔt秒の場合、最初の（$k = 0$の）データサンプルは0秒の位置にあり、その後のk番目の（kは正の整数）データサンプルは$k\Delta t$秒の位置にあります。同様に、周波数分解能が

Δf Hz（ $\Delta f = \frac{f_s}{N}$ ）の場合、DFTのk番目のサンプルは周波数が$k\Delta f$の位置にあります。（実際には、この後に説明するように、これが有効なのは最初の半分の周波数成分に対してだけであり、残りの半分は負の周波数成分を示します。） サンプル数（N）が偶数か奇数かによって、DFT の k 番目のサンプルに対応する周波数の解釈は異なります。

たとえば、Nが偶数であり $p = \frac{N}{2}$ であると仮定します。次の表は、複素数出力列Xの各形式要素が対応する周波数を示します。

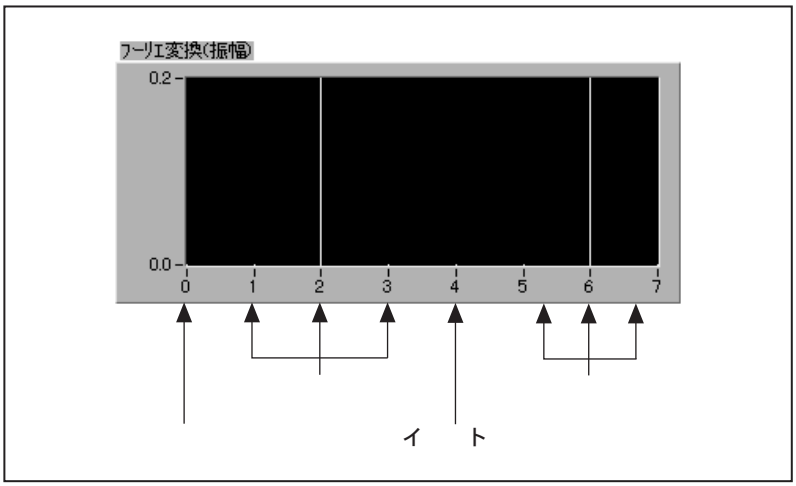
ここで、p 番目の要素（X[p]）がナイキスト周波数に対応することに注意してください。2 列目のナイキスト周波数を超える負の項目は、負の周波数を示します。

たとえば、$N = 8$、$p = N/2 = 4$ の場合は次のようになります。

X[0]	DC
X[1]	Δf
X[2]	$2\Delta f$
X[3]	$3\Delta f$
X[4]	$4\Delta f$ （ナイキスト周波数）
X[5]	$-3\Delta f$
X[6]	$-2\Delta f$
X[7]	$-\Delta f$

ここで、X[1] と X[7]、X[2] と X[6]、X[3] と X[5] はそれぞれ同じ振幅になります。違うのは、X[1]、X[2]、X[3] は正の周波数成分に対応し、X[4]、X[5]、X[6] は負の周波数成分に対応する点です。ただし、X[4] はナイキスト周波数の位置にあることに注意してください。

次の図は、*N* = 8 の場合のこの複素数列を示します。



正と負の周波数の両方が現れるような表現法を、**両側変換**と言います。

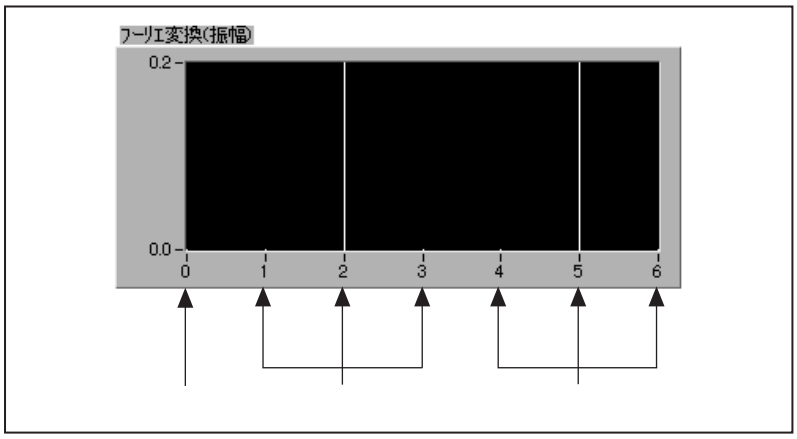
N が奇数の場合、ナイキスト周波数の位置には成分がありません。

たとえばN = 7、p = (N-1)/2 = (7-1)/2 = 3 の場合は、次のようになります。

X[0]	DC
X[1]	Δf
X[2]	$2\Delta f$
X[3]	$3\Delta f$
X[4]	$-3\Delta f$
X[5]	$-2\Delta f$
X[6]	$-\Delta f$

この場合は、X[1] と X[6]、X[2] と X[5]、X[3] と X[4] がそれぞれ同じ振幅になります。ただし、X[1]、X[2]、X[3] は正の周波数成分に対応し、X[4]、X[5]、X[6] は負の周波数成分に対応します。*N* が奇数であるため、ナイキスト周波数の位置には成分がありません。

次の図は、前の$N = 7$の表を示したものです。



正と負の両方の周波数がありますので、これも両側変換です。

高速フーリエ変換

N個のデータサンプルに対してDFTを直接実行するには、約N^2回の複素数演算が必要で、処理時間がかかります。一方、次のように数列のサイズが2の累乗である場合は、

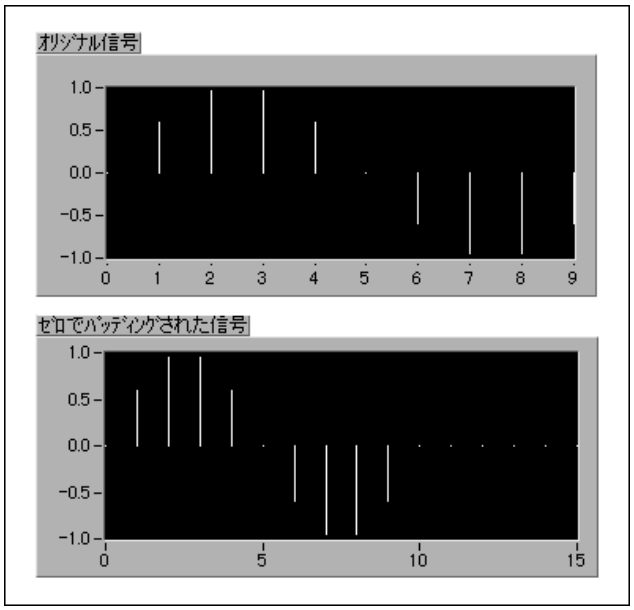
$m = 1, 2, 3,...$の場合、$N = 2m$

約$N\log_2(N)$回の計算でDFT演算を実行することができます。この場合はDFT演算が非常に高速に行われ、DSPの資料ではこのようなアルゴリズムのことを高速フーリエ変換（FFT）と呼んでいます。FFTは、サンプル数（N）が2の累乗である場合の高速DFT演算アルゴリズムというわけです。

FFTの長所としては、VIではFFTを計算する際、出力の計算に別のメモリバッファが不要なため、高速でメモリ効率が良いことがあります。ただし、入力数列のサイズは2の累乗でなければなりません。DFTは任意のサイズの数列を効率よく処理しますが、処理中に中間結果を格納するために別のバッファを割り当てる必要があるため、FFTよりもスピードが遅く、多くのメモリを使用します。

ゼロパッド

入力数列のサイズを2の累乗に等しくするためには、数列の最後に0をパッドして、サンプル全体の数が一つ上の2の累乗に等しくなるようにする方法が使用されます。たとえば、信号サンプルが10個ある場合は、0を6個付加するとサンプルの合計数を16個（$= 2^4 - 2$の累乗）にすることができます。この様子を次に示します。



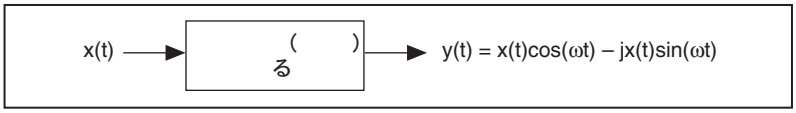
時間領域の波形の終わりに0を付加しても、信号のスペクトルには影響がありません。ゼロパットを行うと、サンプルの合計数を2の累乗にしてFFTを使用した高速演算を可能にするだけでなく、サンプル数Nを大きくすることによって周波数分解能（$\Delta f = fs/N$であることを思い出してください）を高くするのにも役立ちます。

解析ライブラリに含まれるFFT VI群

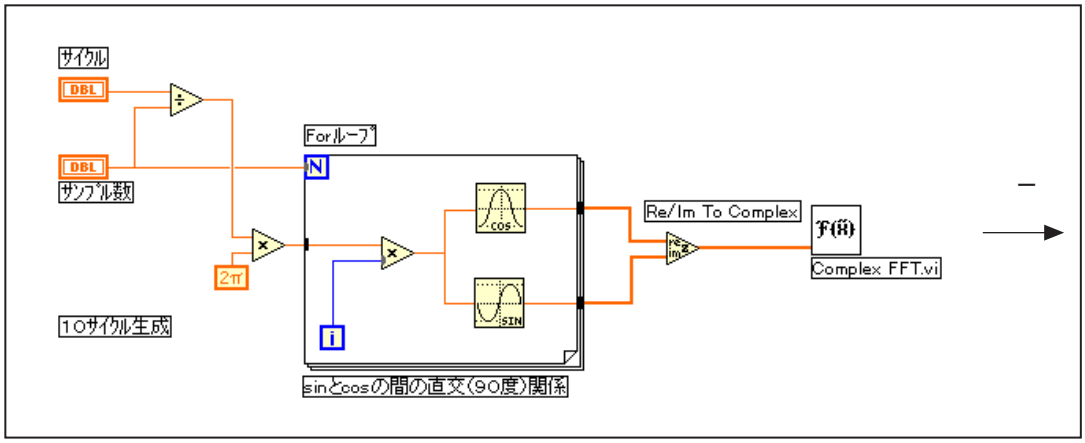
解析ライブラリには、信号のFFT演算を行うための、Real FFT VIおよびComplex FFT VIという2つのVIが含まれています。

2つのVIの違いは、Real FFT VIが実数値信号のFFTを計算するのに対し、Complex FFT VIは複素数値信号のFFTを計算することにあります。ただし、いずれのVIでも出力は複素数になるということに注意してください。

実際には信号のほとんどが実数値であるため、ほとんどのアプリケーションでReal FFT VIを使用することができます。もちろん、信号の虚部を0にすればComplex FFT VIを使用することもできます。Complex FFT VIを使用できるアプリケーション例としては、信号に実部と虚部の両方の要素が含まれている場合があります。通信分野にはこのような信号が数多くあり、複素数指数関数により波形が変調されます。複素数指数関数による変調処理を行うと、次のような複素数信号が得られます。



次に示すブロックダイアグラムは、10サイクルの複素数信号を生成できる簡単な方法を示します。

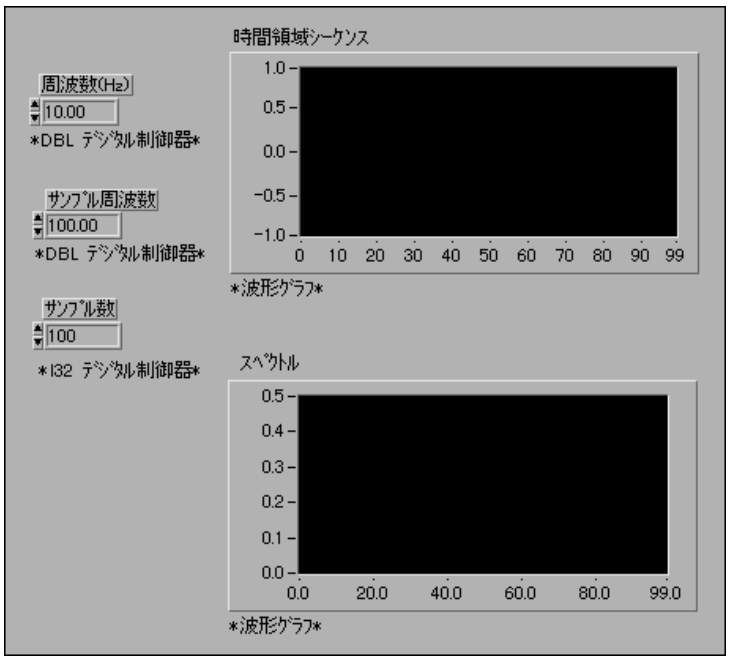


作業 13-1.　Real FFT VI を使用する

ここでは、Real FFT VI を使用して信号の両側フーリエ変換と片側フーリエ変換を表示し、周波数スペクトル内のエイリアス効果を観察することが目的です。

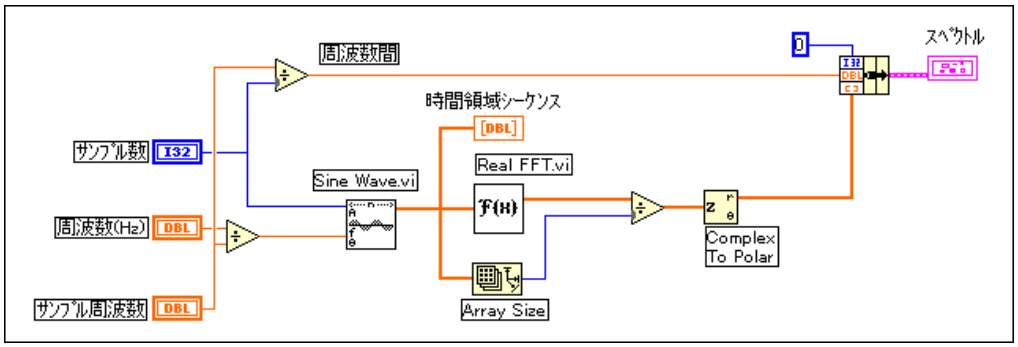
フロントパネル

1. 次の図のようなフロントパネルを作成します。



ブロックダイアグラム

2. 次の図のようなブロックダイアグラムを作成します。



Array Size 関数（**関数→配列**パレット）は、周波数成分の正しい振幅が得られるように、FFT の出力に対して**サンプル数**によるスケーリングを行います。

Sine Wave 関数（**関数→解析→信号生成**パレット）は、時間領域の正弦波波形を生成します。

Real FFT 関数（**関数→解析→デジタル信号処理**パレット）は、入力データサンプルの FFT 演算を実行します。

Complex to Polar 関数（**関数→数値→複素関数**パレット）は、FFT の複素数出力を実部と虚部（振幅部と位相部）に分離します。位相情報の単位はラジアンです。ここでは、FFT の振幅だけを表示します。

周波数間隔 Δf は、**サンプル周波数**を**サンプル数**で割ることにより得られます。

3. この VI を、`FFT_2sided.vi` という名前で `LabVIEW¥Activity` ディレクトリに保存します。
4. **周波数（Hz）**= 10、**サンプル周波数**= 100、**サンプル数**= 100 を設定し、VI を実行します。

時間波形と周波数スペクトルのプロットを確認してください。**サンプル周波数**=**サンプル数**= 100 ですので、1 秒間サンプリングが行われます。したがって、時間波形内に表示される正弦波のサイクル数は、ユーザが選択した**周波数（Hz）**に等しくなります。この場合は 10 サイクルが表示されます（**周波数（Hz）**を 5 に変更した場合は 5 サイクルが表示されます）。

両側 FFT

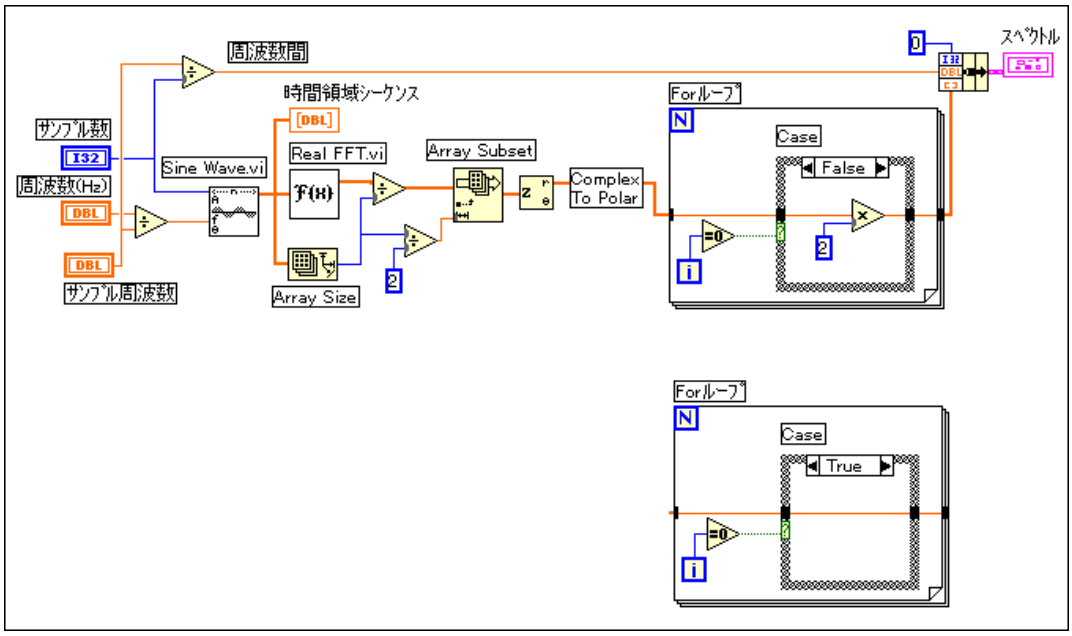
5. 周波数スペクトル（フーリエ変換）を確かめてください。10 Hz と 90 Hz の位置に 1 つずつ、2 つのピークがあることがわかります。90 Hz のピークは、実際には 10 Hz の負の周波数です。この場合に表示されるプロットは、正と負の周波数を表示するため、**両側 FFT** と呼ばれます。
6. **周波数（Hz）** = 10 にして VI を実行し、続いて**周波数（Hz）** = 20 にして VI を実行します。それぞれの場合の、スペクトルの両方のピークの移動を確認してください。

☞ **注**　**また、周波数（Hz）= 10、20 の場合の時間領域のプロットも調べ、どちらの方が正弦波が良く表現されているか確認し、その理由を説明してください。**

7. fs = 100 Hz であるため、正確にサンプリングできるのは周波数＜50 Hz（ナイキスト周波数 = fs/2）の信号のみです。**周波数（Hz）** を 48 Hz に変更してください。スペクトルプロットの ± 48 Hz の位置にピークが現れるはずです。
8. 次に**周波数（Hz）** を 52 Hz に変更してください。ステップ 5 の結果と今回のプロットの間に違いはありますか？　52 ＞ナイキストであるため、周波数 52 のエイリアスは|100–52| = 48 Hz となります。
9. **周波数（Hz）** を 30 Hz、70 Hz にして VI を実行し、両者の間に違いがあるか調べ、理由を説明してください。

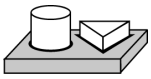
片側 FFT

10. VI のブロックダイアグラムを次のように修正します。FFT では正と負両方の周波数に関する情報が含まれるため情報の繰り返しがあることがわかりました。この修正を行うと、FFT の点の半分（正の周波数成分）しか表示されません。この表現法は**片側 FFT** と呼ばれます。片側 FFT では正の周波数成分しか表示されません。正しい振幅を得るには正の周波数成分に 2 をかける必要があることに注意してください。ただし、DC 成分はそのまま残されます。



Equal To 0? 関数（**関数→比較**パレット）は、配列の指標が 0 に等しいかどうかをテストします。0 である場合はDC 成分に対応し、2 をかける必要はありません。

11. 値を以下のように設定して VI を実行します。**周波数（Hz）**= 30、**サンプル周波数**= 100、**サンプル数**= 100。
12. この VI を、`FFT_1sided.vi` という名前で `LabVIEW¥Activity` ディレクトリに保存します。
13. **周波数（Hz）**の値を 70 に変更して VI を実行します。今回の結果とステップ 9 の結果の間に違いがあるか調べてください。



これで作業 13-1 は完了です。

パワースペクトル

実際の信号のDFT（またはFFT）は、実部と虚部を持つ複素数になることを説明しました。DFT/FFT で示される各周波数成分の**パワー**を求めるには、その周波数成分の振幅の2乗を計算します。したがって、*k* 番目の周波数成分（DFT/FFT の *k* 番目の要素）のパワーは|*X*[k]|^2 で示されます。各周波数成分の電力を示すプロットのことを、**パワースペクトル**と言います。実際の信号のDFT/FFTは対称ですので、正の周波数 $k\Delta f$ におけるパワーは、対応する負の周波数 $-k\Delta f$ におけるパワーと等しくなります（DC 成分とナイキスト成分は含まれません）。DC 成分とナイキスト成分の合計パワーはそれぞれ $|X[0]|^2$ および $\left|X\left[\frac{N}{2}\right]\right|^2$ になります。

位相情報の喪失

DFT/FFT の振幅を2乗することによりパワーが得られることから、パワースペクトルは常に実数になります。この方法の不利な点は、位相情報が失われることです。位相情報が必要な場合は、複素数出力が得られるDFT/FFT を使用する必要があります。

位相情報が不要なアプリケーション（たとえば信号に含まれる高調波のパワーを求める場合など）では、パワースペクトルを使用することができます。非線形システムに正弦波入力を加えると、システム出力に含まれる高調波のパワーを調べることができます。

サンプル間の周波数間隔

解析→デジタル信号処理サブパレットの Power Spectrum VI を使用すると、時間領域のデータサンプルのパワースペクトルを計算することができます。DFT/FFT の場合と全く同様に、Power Spectrum VI 出力からのサンプル数は、入力に加えられたデータサンプル数と同じになります。また、出力サンプルの間の周波数間隔は $\Delta f = fs/N$ になります。

まとめ

信号の時間領域における表現（サンプル値）は、離散フーリエ変換（DFT）というアルゴリズムによって周波数領域の表現に変換することができます。DFT を高速に計算するには、高速フーリエ変換（FFT）というアルゴリズムが使用されます。信号のサンプル数が 2 つ累乗である場合に、このアルゴリズムを使用することができます。

通常の DFT/FFT は、出力に正と負の周波数情報が含まれるため、両側 DFT/FFT となります。DFT/FFT の出力点を半分だけ使用することで、この出力を片側 DFT/FFT に変換することができます。DFT/FFT のサンプル間の周波数間隔は $\Delta f = fs/N$ となります。

パワースペクトルを求めるには、個々の周波数成分の振幅を 2 乗します。上級解析ライブラリに含まれる Power Spectrum VI は自動的にこの処理を行います。Power Spectrum VI の出力の単位は $\boldsymbol{Vrms}^2$ です。ただし、パワースペクトルでは位相情報が得られません。

DFT、FFT、およびパワースペクトルは、固定信号や過渡信号の周波数成分の測定に役立ちます。FFT では、信号が集録された時間全体にわたる信号の周波数成分の平均値が得られます。このため、FFT が使用されるのはほとんど、固定信号（信号の集録中に周波数成分が大きく変化しないもの）を解析する場合や、各周波数ラインの平均エネルギーだけを求めたい場合です。測定問題のほとんどは、この種類に属します。集録中に変化する周波数情報を測定するためには、時間／周波数共同領域解析ソフトウェア（joint time-frequency analysis: JTFA）ツールキット、またはウェーブレットとフィルタバンクデザイン（wavelet and filter banks designer: WFBD）ツールキットをご使用ください。

14

スムージングウィンドウ

この章では、ウィンドウを使用して、集録された信号のスペクトルの漏洩を防止し解析性能を改善する方法を説明します。解析ウィンドウ VI の使用方法については、examples¥analysis¥windxmpl.llb の例を参照してください。

スムージングウィンドウの概要

実際の信号サンプリングアプリケーションでは、慎重にサンプリング定理やサンプリング条件を守っても、信号から得られるレコード数には限りがあります。残念ながら離散的時間システムでは、サンプリングするレコード数が有限であることから、波形の一部が切り取られてしまい、時間的に連続した元の信号とはスペクトル特性が異なる波形になってしまいます。このような不連続があるとスペクトル情報の漏洩が起こるため、時間的に連続した元のスペクトルに比べて品質の低下した離散時間スペクトルになります。

サンプリングされた信号のスペクトル特性を改善する方法としては、スムージングウィンドウを適用するのが簡単です。有限の長さのデータに対してフーリエ解析やスペクトル解析を行う場合は、ウィンドウを使用することで、カットされた波形の過渡的なエッジを最小限に抑え、スペクトルの漏洩を少なくします。このような方法で使用した場合、スムージングウィンドウは、あらかじめ定義された狭帯域ローパスフィルタと同様に機能します。

スペクトルの漏洩とスムージングウィンドウについて

DFT/FFT を使用して信号の周波数成分を調べる場合、本質的にデータは定期的に繰り返す波形の 1 周期分であることが前提になっています。この様子を図14-1に示します。表示されているうちの最初の1周期分は、サンプリングされたものです。これ以後は、この周期の波形が時間とともに繰り返されることで周期的な波形が得られています。

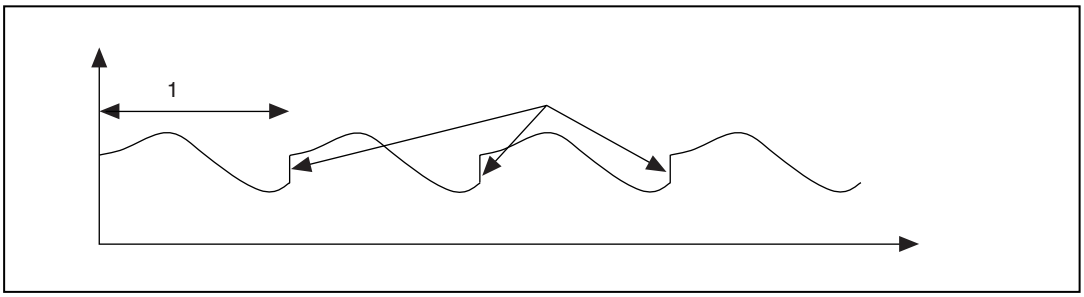


図 14-1 サンプリングされた周期から作成された周期的な波形

上図に示す通り、波形が周期的であるという前提があるために、隣合う周期の間に不連続部ができます。サンプリングのサイクル数が整数でない場合に、このような現象が起こります。このような**人工的原因**による不連続性は、信号スペクトル中で、元の信号には存在しなかった非常に高い周波数として現れます。これらの周波数はナイキスト周波数に比べてはるかに高い場合があり、前に説明したように、0 ～ fs/2 の範囲のどこかにエイリアスとなって現れます。したがって、DFT/FFT によって得られるスペクトルは、元の信号の本当のスペクトルでない、品質の低下したスペクトルになってしまいます。1 つの周波数のエネルギーが他のすべての周波数に**漏れ出した**ように見えることから、この現象を**スペクトル漏洩**と呼びます。

図14-2に、正弦波とそれに対応するフーリエ変換を示します。サンプリングされた時間領域波形をグラフ1に示します。フーリエ変換では周期性が前提となっているため、この波形が時間を追って繰り返され、グラフ1の正弦波の周期的な波形はグラフ2のようになります。これに対応するスペクトル表現はグラフ3のようになります。グラフ2の時間的なレコードは周期的であり不連続部がないため、そのスペクトルは、正弦波の周波数を示す1本の線となります。グラフ2の波形に不連続が全くないのは、時間波形をサンプリングしたサイクル数が整数（この場合は1）であったためです。

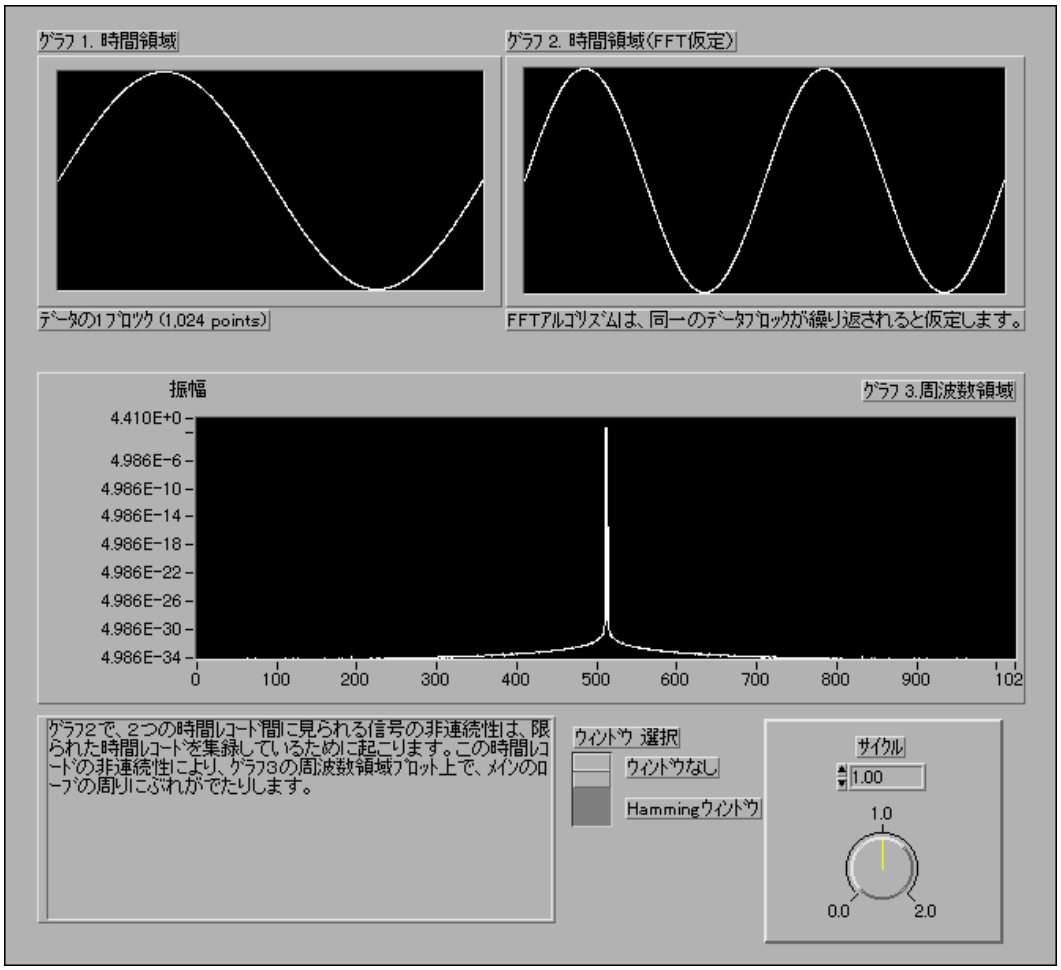


図 14-2 正弦波と対応するフーリエ変換

図14-3は、時間波形をサンプリングしたサイクル数が整数でない（この場合は1.25）場合のスペクトル表現を示します。この場合のグラフ1には1.25サイクル分の正弦波が含まれます。この波形を周期的に繰り返すと、グラフ2のような不連続部を含む波形になります。これに対応するスペクトルはグラフ3のようになります。今度はエネルギーが図のように広範囲の周波数に渡って広がってしまうことに注目してください。このようにエネルギーが広がってしまうのが**スペクトル漏洩**です。FFTのラインの1つからエネルギーが漏れて、他のすべてのラインに広がっています。

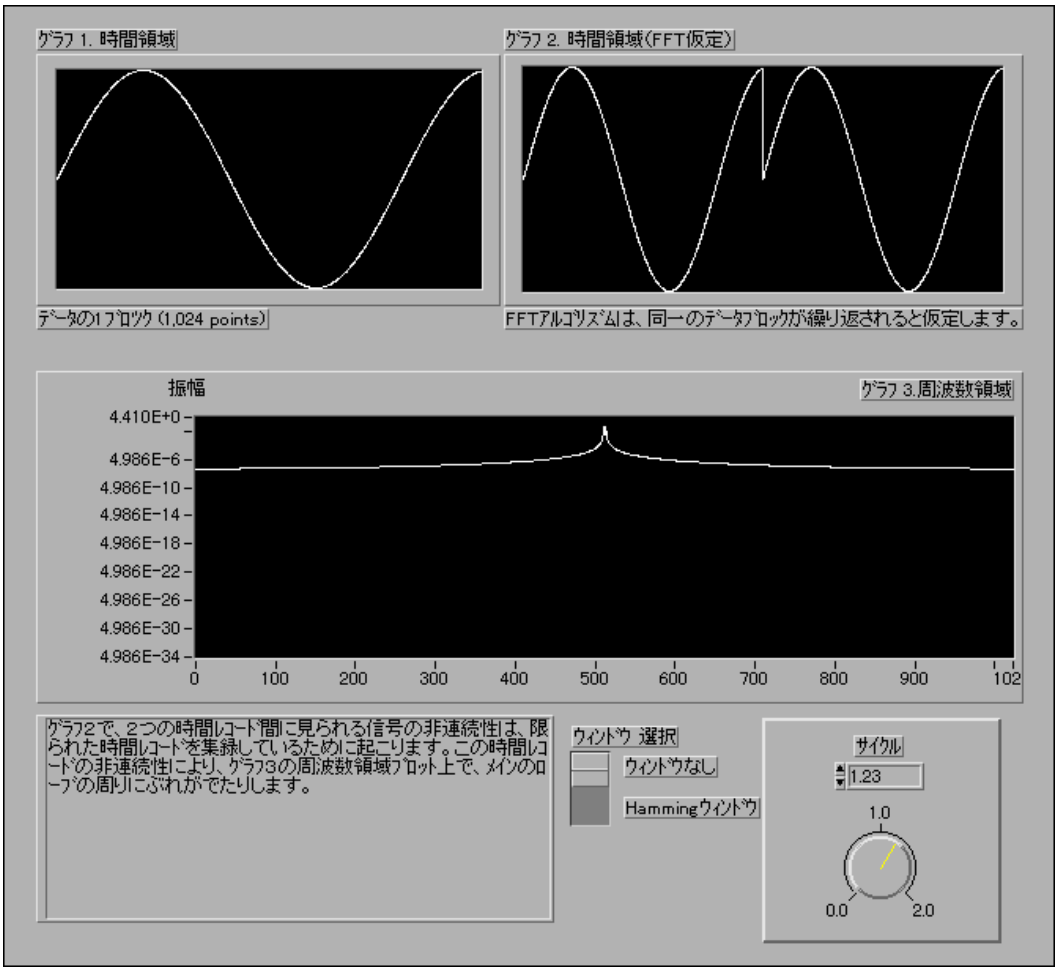


図 14-3 サンプリングしたサンプル数が整数でない場合のスペクトル表現

漏洩が起きるのは、入力信号の時間レコードが有限だからです。漏洩を解決する一つの方法は、－無限大から＋無限大までの無限の時間レコードを取り込むことです。こうすれば、正しい周波数の位置の1本のラインがFFTで計算されます。しかし、無限の時間を待つことは実際には不可能です。このように時間レコードには限界があることから、スペクトル漏洩を軽減するためのこれに代わる技術として、**ウィンドウ処理**という方法が使用されます。

スペクトル漏洩の量は、不連続部の振幅によって決まります。不連続部が大きいほど漏洩量は多く、小さいほど少なくなります。ウィンドウ処理を使用すると、各周期の境界部における不連続部の振幅を小さくすることができます。ウィンドウ処理では、時間レコードに、境界部で振幅が0に向かって滑らかに徐々に減衰していく有限長のウィンドウをかけ合わせます。この様子を図14-4に示します。この図では元の時間信号を、**Hamming**ウィンドウを使用してウィンドウ処理しています。ウィンドウ処理された信号の時間波形が端部に向かって徐々に0に減衰していくことに注目してください。したがって、有限長データに対してフーリエ解析やスペクトル解析を実行する場合、ウィンドウを使用するとサンプリングされた波形の過渡部のエッジを最小限に抑えることができます。データを周波数領域に変換する前にこのデータにスムージングウィンドウ関数を適用すると、スペクトル漏洩を最小限に抑えることができます。

図14-2のように時間レコードに含まれるサイクル数が整数である場合は、周期的であることが前提となっているため不連続部ができず、スペクトル漏洩は起こりません。問題となるのは、サイクル数が整数でない場合だけです。

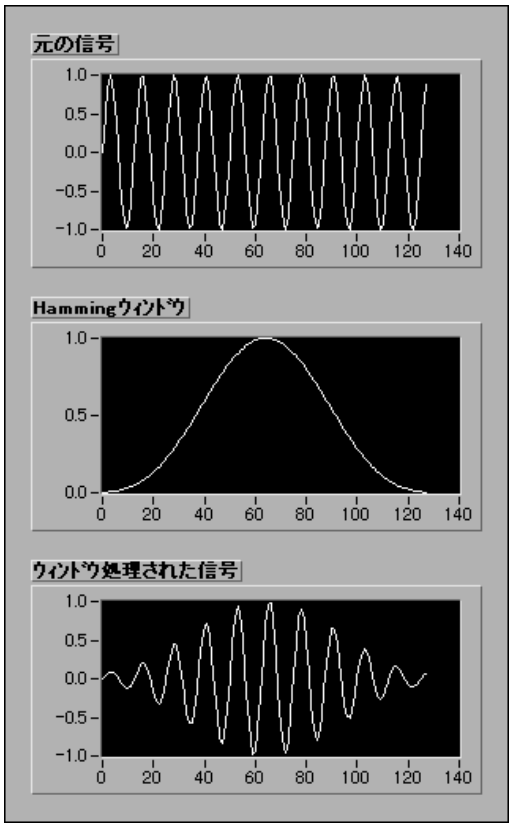


図 14-4 Hamming ウィンドウでウィンドウ処理された時間信号

ウィンドウ処理アプリケーション

ウィンドウ処理を使用するいくつかの理由を次に示します。

- 観測時間を定義する。
- スペクトル漏洩を抑制する。
- 互いの周波数が非常に近い振幅の小さい信号を、振幅の大きい信号から分離する。

さまざまなウィンドウ関数の特性

時間領域の信号にウィンドウを適用する（ウィンドウ処理する）ことは、信号にウィンドウ関数をかけることと同じです。時間領域での乗算は周波数領域でのくり込みと同じであるため、ウィンドウ処理された信号のスペクトルは、元の信号のスペクトルをウィンドウのスペクトルでくり込んだしたものになります。このように、ウィンドウ処理を行うと時間領域の信号が変形されるだけでなく、表示されるスペクトルにも影響を与えます。

LabVIEW 解析ライブラリには、タイプの異なる多くのウィンドウが用意されています。どのウィンドウが役に立つかは、ユーザのアプリケーションによって異なります。このようなウィンドウのいくつかを、以下に示します。

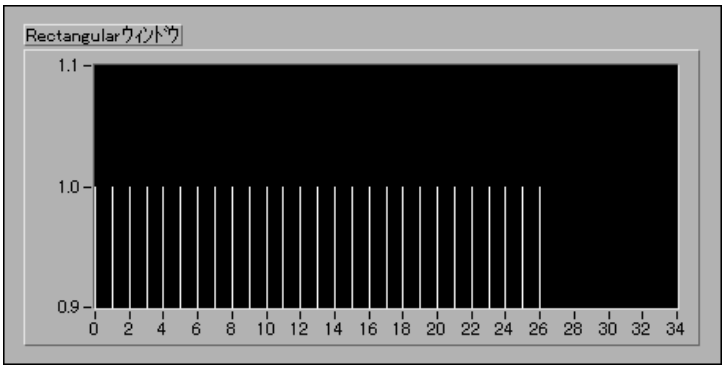
Rectangular（なし）

Rectangularウィンドウは、時間間隔上での値が1になり、数学的には次のように記述されます。

$$w[n] = 1.0 \quad n = 0, 1, 2.....N-1$$

ここで、*N*はウィンドウの長さです。Rectangularウィンドウを適用することは、ウィンドウを全く使用しないことと同じです。これは、Rectangular関数は信号を有限の時間間隔内で切り取るだけだからです。Rectangularウィンドウでは、スペクトル漏洩の量が最も大きくなります。

N = 32 の場合の Rectangular ウィンドウを次の図に示します。



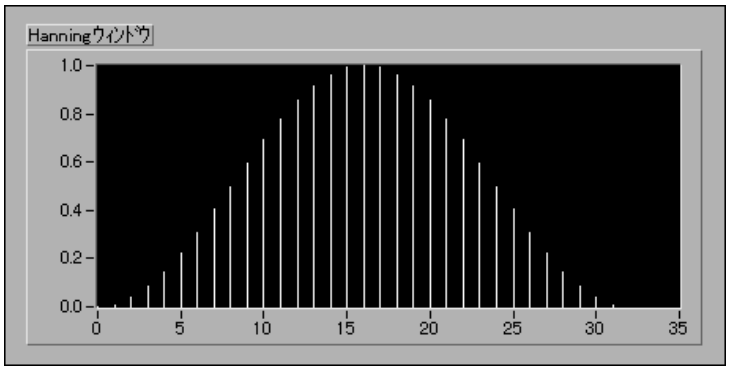
Rectangular ウィンドウはウィンドウの持続時間よりも短い過渡現象の解析に役立ちます。また、Rectangular ウィンドウは、有効サンプリングレートが回転する機械のシャフトの速度に比例する**オーダートラッキング**にも使用されます。このアプリケーションでは、Rectangular ウィンドウは機械の主振動モードとその高調波を検出します。

Hanning

このウィンドウの形状は、余弦波の 1/2 サイクルと似ており、定義式は次のようになります。

$$w(n) = 0.5 - 0.5\cos(2\pi n/N), \quad n = 0, 1, 2, ...N-1$$

N = 32 の場合の Hanning ウィンドウを、次の図に示します。



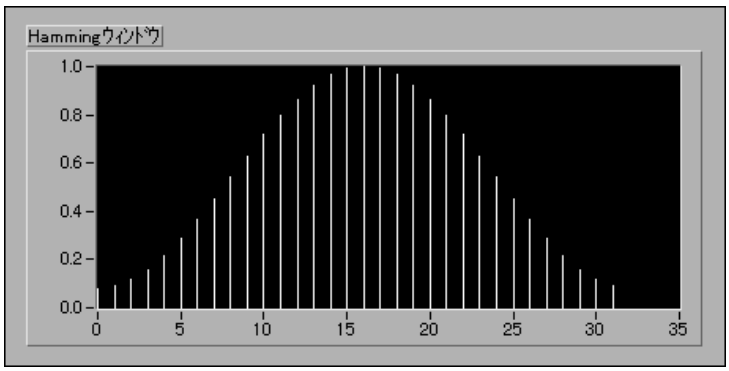
Hanning ウィンドウは、ウィンドウの持続時間よりも長い過渡現象の解析に役立つほか、汎用アプリケーションにも役立ちます。

Hamming

このウィンドウは、Hanning ウィンドウを修正したものです。このウィンドウの形状は余弦波にも似ており、次のように定義されます。

$$w(n) = 0.54 - 0.46\cos(2\pi n/N), \quad n = 0, 1, 2, ...N-1$$

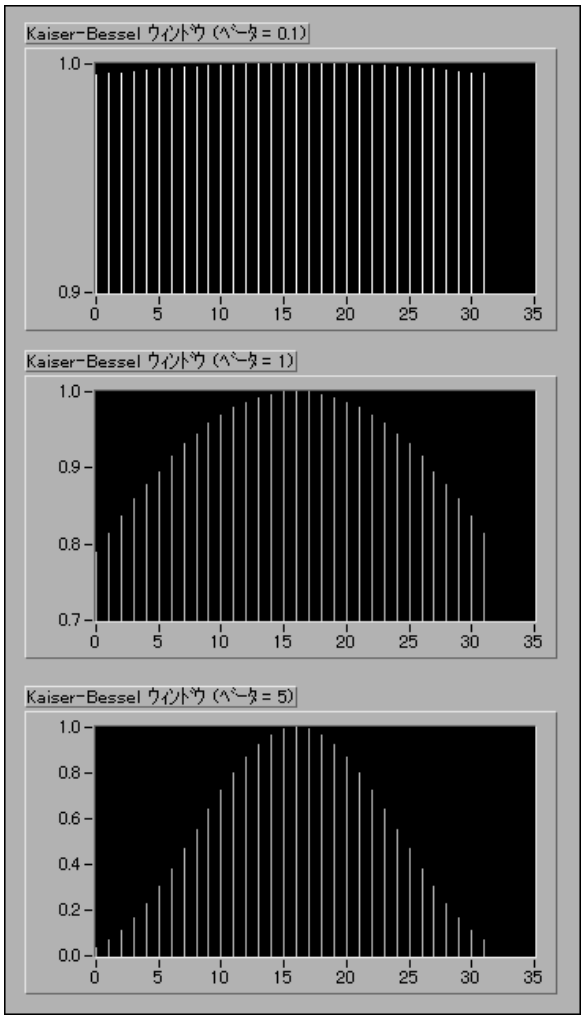
N = 32 の場合の Hamming ウィンドウを次に示します。



HanningウィンドウとHammingウィンドウは多少似ていることがおわかりでしょう。ただし、時間領域では Hamming ウィンドウはエッジ付近で Hanning ウィンドウほど 0 に近くならないことに注意してください。

Kaiser-Bessel

このウィンドウはいわゆる柔軟性の高いウィンドウであり、**ベータ**パラメータを変更することによりユーザが形状を変更できます。したがって、ユーザアプリケーションに合わせてウィンドウの形状を変更してスペクトル漏洩の程度を制御することができます。さまざまな**ベータ**の値における Kaiser-Bessel ウィンドウを次に示します。



ベータの値が小さい場合は形状が Rectangular ウィンドウに近くなることに注意してください。実際に、ベータ = 0.0 の場合には Rectangular ウィンドウになります。ベータを大きくしていくに従って、両サイドに向かうウィンドウの傾斜が強くなります。

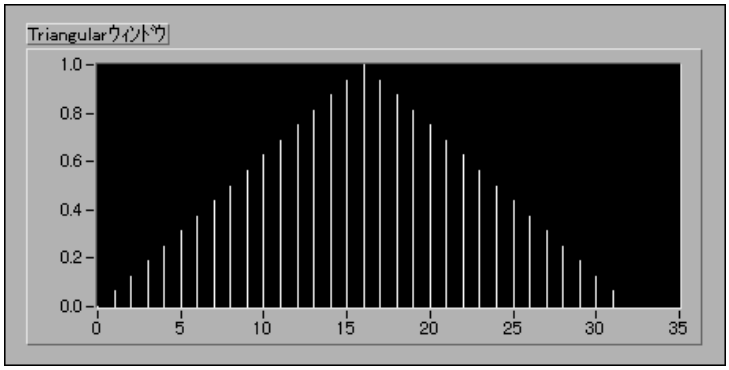
このウィンドウは、周波数がほとんど同じで振幅が大幅に異なる 2 つの信号を検出するのに適しています。

Trianglular

このウィンドウの形状は三角形であり、次の式で得られます。

$$w[n] = 1 - | (2n-N) / N |, \quad n = 0, 1, 2, ..., n-1$$

N= 32 の場合の Trianglular ウィンドウを次に示します。



Flat Top

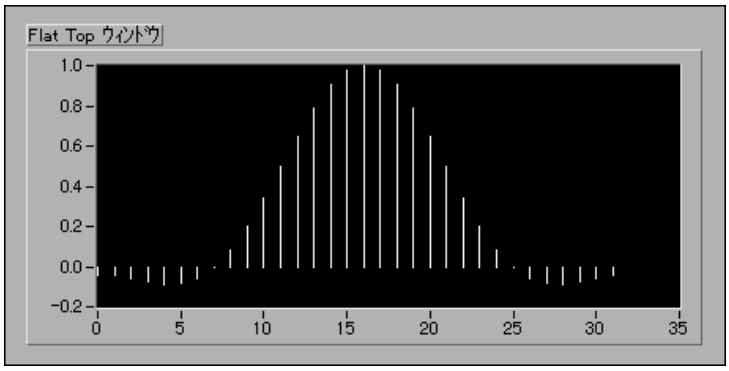
このウィンドウは、すべてのウィンドウ関数の中で最も高い振幅精度を持っています。振幅精度が高い（正確に完全なサイクルの間にある信号の場合に± 0.02 dB）代わりに、周波数の選択性が低くなっています。Flat Top ウィンドウは、信号内の近い位置にスペクトルエネルギーがほとんどないような 1 つの周波数成分の振幅を正確に測定するのに最適です。Flat Top ウィンドウは次のように定義されます。

$$w(n) = a0 - a1*\cos(2\pi n/N) + a2*\cos(4\pi n/N)]$$

ただし、

$$a0 = 0.2810638602$$
$$a1 = 0.5208971735$$
$$a2 = 0.1980389663$$

Flat Topウィンドウを次に示します。

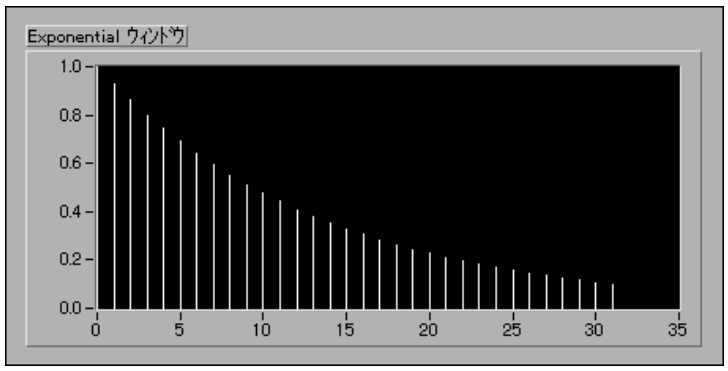


Exponential

このウィンドウは、指数関数的に減衰していく形状になり、数学的には次のように表現されます。

$$w[n] = e^{\left(\frac{n\ln(f)}{N-1}\right)} = f^{\left(\frac{n}{N-1}\right)}, \quad n = 0, 1, 2.....N-1$$

ここで、fは最終値です。ウィンドウの初期値は1であり、0に向かって徐々に減衰していきます。指数の最終値は0～1の間で調整できます。N = 32、最終値を0.1に指定した場合のExponentialウィンドウを、次に示します。



このウィンドウは、ウィンドウの長さより持続時間が長い過渡現象（短い持続時間の間だけ存在する信号）の解析に役立ちます。

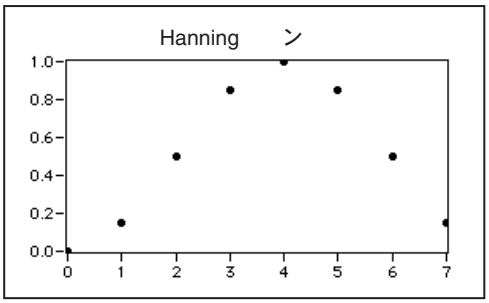
このウィンドウは、（ハンマーなどの）衝撃により励起された光の減衰をともなう構造体の応答のような、指数関数的に減衰する信号に適用できます。

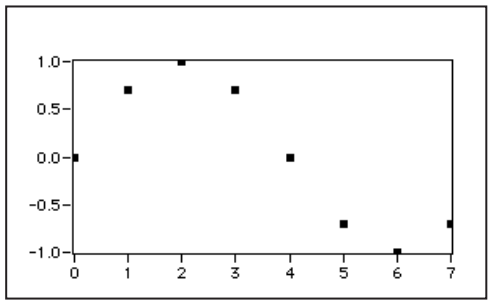
スペクトル解析用のウィンドウと係数設計用のウィンドウの比較

LabVIEW の解析ライブラリに組み込まれているウィンドウ VI は、スペクトル解析アプリケーション用に設計されています。これらのアプリケーションでは、入力信号はいずれかのウィンドウ VI を通して渡すことによりウィンドウ処理されます。ウィンドウ処理された信号は、この後、周波数領域での表示や解析を行うために DFT ベースの VI に渡されます。

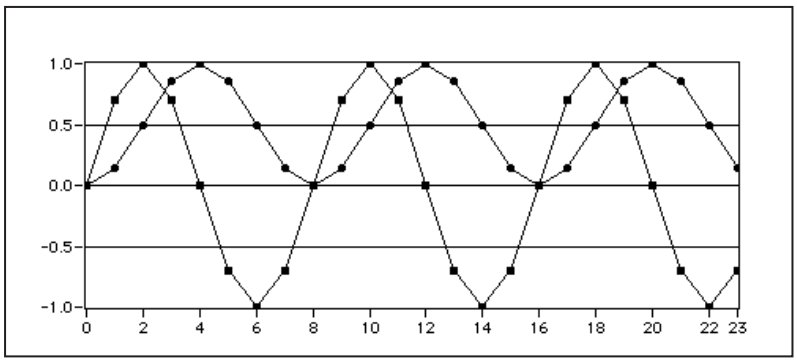
スペクトル解析用に設計されるウィンドウ関数は、**DFT- イーブン**（Fredric J. Harris が論文『On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform（Preceedings of the IEEE, Volume 66, No.1, January 1978）』で定義した用語）でなければなりません。ウィンドウ関数が DFT- イーブンになるのは、この関数と正弦波数列の積分サイクルとのドット積（内積）が完全に 0 に等しい場合です。DFT- イーブンの数列について考えるべきその他の点は、この数列の DFT には虚数成分が全く含まれないことです。

次の図は、Hanning ウィンドウと、サンプルサイズが 8 の正弦波パターンの 1 サイクル分を示します。次の図の DFT- イーブンの Hanning ウィンドウでは、正弦波パターンの完全な 1 サイクルのように中点において対称でなく、最後の点は最初の点と等しくなりません。





最後に、DFTでは入力数列は周期的であり、解析する信号は実際の入力信号からの連続とみなされます。次の図は、前の数列の3サイクル分を示し、DFT-イーブンのウィンドウと1サイクルの正弦波パターンを滑らかに周期的に展開したものを示します。



その他のタイプのウィンドウアプリケーションとしては、FIR フィルタ設計アプリケーションがあります。「第16章　フィルタ処理」の「ウィンドウ処理されたFIR フィルタ」の項を参照してください。このアプリケーションでは、中点において対称なウィンドウが必要です。

次のHanningウィンドウ関数の式は、DFT-イーブンのウィンドウ関数（スペクトル解析）と対称なウィンドウ関数（係数設計）の違いを示します。

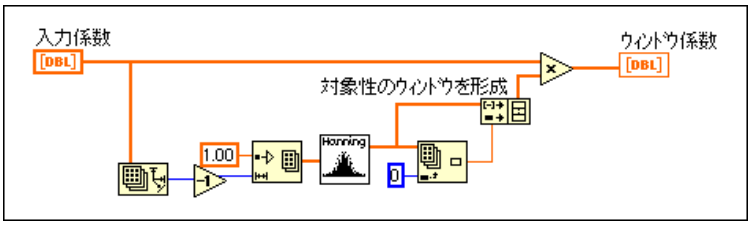
スペクトル解析用のHanningウィンドウ関数は、次のようになります。

$$w[i] = 0.5\left(1 - \cos\left(\frac{2\pi i}{N}\right)\right), \quad i=0,1,2,\ldots,N-1$$

対称な係数設計用のHanningウィンドウ関数は、次のようになります。

$$w[i] = 0.5\left(1 - \cos\left(\frac{2\pi i}{N-1}\right)\right), \quad i=0,1,2,\ldots,N-1$$

上記の2つの式は、DFT-イーブンなウィンドウの使用方法をわずかに変更するだけで対称なウィンドウ関数を実装できることを示しています。次の図は、Hanning Window VIを使用してフィルタ係数用の対称なウィンドウ処理を実装するためのブロックダイアグラムを示します。



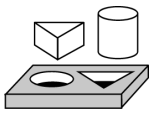
スムージングウィンドウに関する詳細は、「付録 A　解析に関する参考文献」を参照してください。

どのタイプのウィンドウを使用するか？

使用可能なさまざまなウィンドウのいくつかのタイプがわかりました。次に「どのタイプのウィンドウを使用するか？」が問題になります。その答えは、使用する信号のタイプや求める内容によって異なります。適切なウィンドウを選択するには、解析する信号に関してあらかじめある程度のことがわかっている必要があります。さまざまな信号のタイプと、それらに対して使用するのに適切なウィンドウをまとめて、次の表に示します。

信号のタイプ	ウィンドウ
持続時間がウィンドウの長さより短い過渡現象	Rectangular
持続時間がウィンドウの長さより長い過渡現象	Exponential、Hanning
汎用アプリケーション	Hanning
オーダートラッキング	Rectangular
システム解析（周波数応答の測定）	Hanning（乱数励起の場合）、 Rectangular（疑似乱数励起の場合）
周波数が互いに非常に近く振幅が大きく異なる2つの音の分離	Kaiser-Bessel
周波数が互いに非常に近く振幅がほとんど同じ2つの音の分離	Rectangular
単独の音の振幅の正確な測定	Flat Top

信号に関する内容が先にわからない場合も多くありますが、このような場合はさまざまなウィンドウを試して最適なものを見つける必要があります。

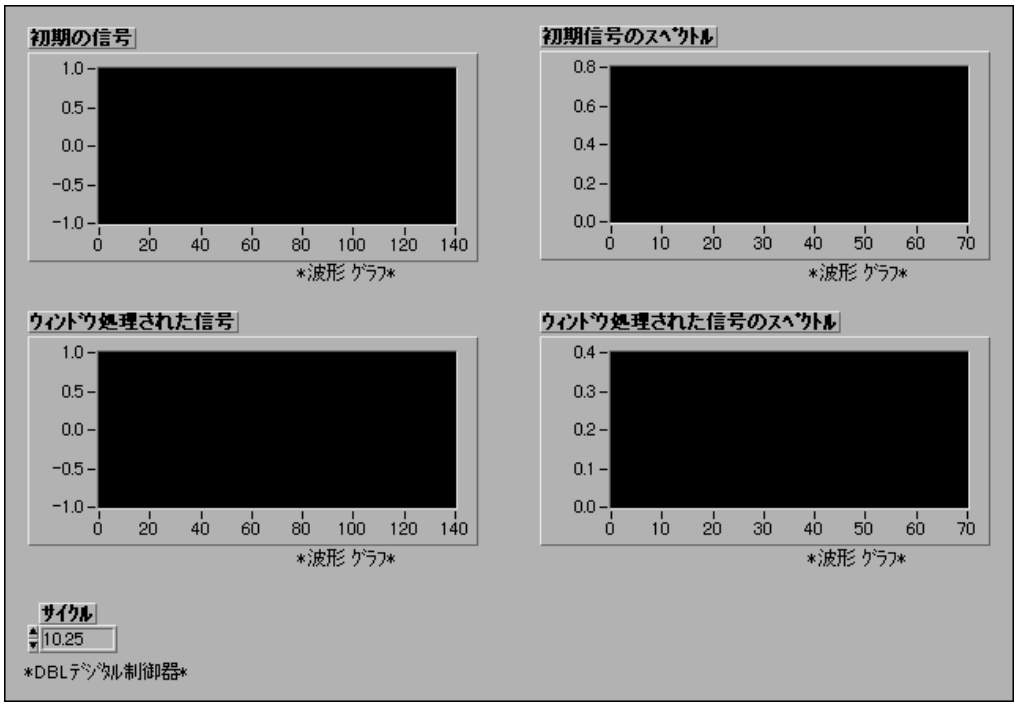


作業 14-1.　ウィンドウ処理された信号とウィンドウ処理されていない信号を比較する

ここでは、(時間領域と周波数領域の両方で) ウィンドウ処理された信号とウィンドウ処理されていない信号の相違点を調べることが目的です。

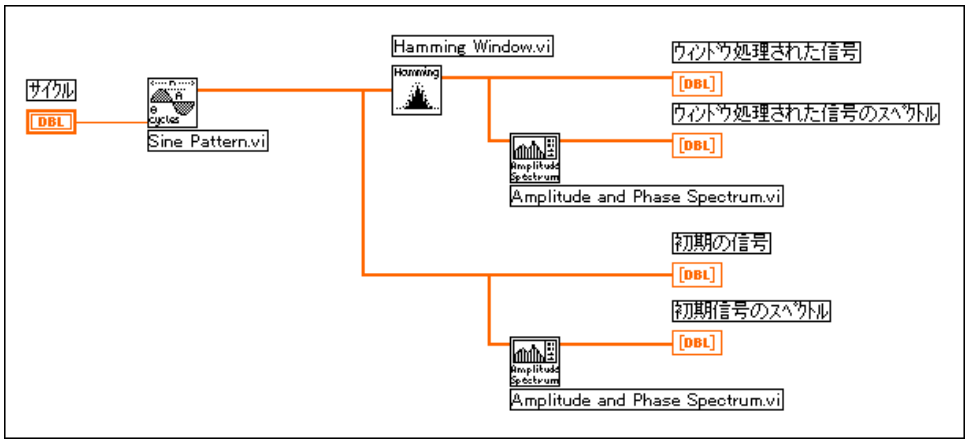
フロントパネル

1. 新しいフロントパネルを開き、次の図のようなオブジェクトを作成します。



ブロックダイアグラム

2. 次の図のようなブロックダイアグラムを作成します。



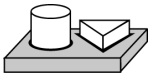
Sine Pattern VI（**関数→解析→信号生成**パレット）は、**サイクル**制御器で指定されたサイクル数の正弦波を生成します。

正弦波の時間波形は Hamming Window VI（**関数→解析→ウィンドウ処理**パレット）を使用してウィンドウ処理され、時間波形をウィンドウ処理したものとウィンドウ処理されていないものの両方が、フロントパネル左側の2つのプロット上に表示されます。

Amplitude and Phase Spectrum VI（**関数→解析→測定**パレット）はウィンドウ処理された時間波形とウィンドウ処理されていない時間波形の振幅スペクトルを求めます。これらの波形はフロントパネル右側の 2 つのプロット上に表示されます。

3. この VI を `Windowed & Unwindowed Signal.vi` という名前で `LabVIEW¥Activity` ディレクトリに保存します。
4. **サイクル**を 10（整数）に設定して VI を実行します。ウィンドウ処理された信号のスペクトルは、ウィンドウ処理されていない信号のスペクトルよりも広く（広帯域に）なることに注目してください。ただし両方のスペクトルは、X 軸の 10 の近くに集中しています。

5. **サイクル**を 10.25（整数でない数）に変更して VI を実行します。今度は、ウィンドウ処理されていない信号のスペクトルが前よりも外に広がっていることに注目してください。これは、サイクル数が整数でなく、この波形を繰り返して周期的なものにしたために不連続部ができたからです。ウィンドウ処理された信号のスペクトルはまだ集中していますが、ウィンドウ処理されていない信号のスペクトルは周波数領域全体に広がっています（すなわちスペクトル漏洩）。
6. **サイクル**を10.5に変更して周波数領域のプロットを確認してください。元の信号のスペクトル漏洩がはっきりわかります。



これで作業 14-1 は完了です。

15

スペクトルの解析と測定

この章では、信号の振幅と位相のスペクトルを測定し、スペクトルアナライザを開発し、全高調波歪み（THD）を測定する方法を示します。測定VIの使用方法については、examples¥analysis¥measure¥measxmpl.llbの例を参照してください。

測定VIの概要

一部の測定VIは通常、振幅と位相のスペクトル、信号のパワースペクトル、ネットワーク転送関数など、一般的に使用される時間領域から周波数領域への変換を実行します。その他の測定VIは、スケーリングされた時間領域のウィンドウ処理やパワーと周波数の評価などを実行するVIとの相互動作を行います。

測定VIは、以下のアプリケーションに使用できます。

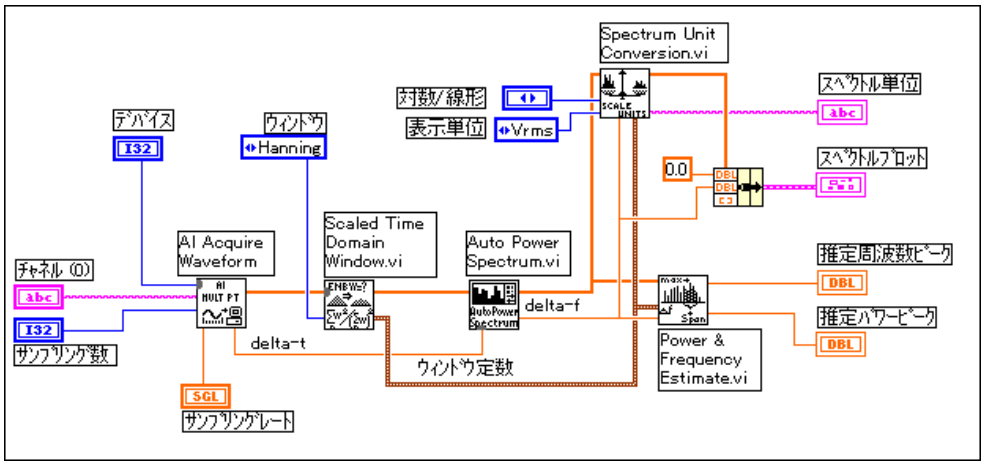
- スペクトル解析アプリケーション
  - 振幅と位相のスペクトル
  - パワースペクトル
  - 基準化時間領域のウィンドウ
  - パワーと周波数指定
  - 高調波解析と全高調波歪み（THD）の測定
- ネットワークとデュアルチャネル解析アプリケーション
  - インパルス応答関数
  - ネットワーク関数（コヒーレンス性も含む）
  - クロスパワースペクトル

DFT、FFT、およびパワースペクトルは、固定的な信号や過渡的な信号の周波数成分を測定するのに役立ちます。FFT では、信号が集録された時間全体に渡る信号の周波数成分の平均値が得られます。このため、FFT はたいてい静的な信号を解析する場合（信号が集録された時間全体に渡って信号の周波数成分が大幅に変動しない場合）、または各周波数ラインにおける平均エネルギーだけが必要な場合に使用されます。測定における問題のほとんどは、このような種類に属します。集録中に変動する周波数情報の測定には、Gabor Spectrogram のような時間周波数共同領域解析 VI を使用する必要があります。

測定 VI は、信号処理 VI の上に作成され、以下の特徴があります。これらは、従来のベンチトップ周波数解析計測器を模倣したものです。

- 実際の時間領域信号が入力されることが前提。
- 出力は振幅と位相であり、スケーリングされ、必要に応じ直ちにグラフ表示できるような単位で出力される。
- DC から $\frac{\text{サンプリング周波数}}{2}$ の範囲の片側スペクトル。
- 適切な X 軸の単位（Hz）でグラフ表示できるようにサンプリング周期から周波数間隔への変換が行われる。
- 必要に応じて使用するウィンドウに合わせた補正が行われる。
- 振幅精度の限界範囲内で各ウィンドウのピークスペクトル振幅が同じになるように、ウィンドウのスケールが設定される。
- パワースペクトルや振幅スペクトルは、デシベルや、V^2/Hz、$V/\sqrt{Hz}$ などのスペクトル密度単位を始めとするさまざまな単位形式で表示される。

次のスペクトルアナライザのダイアグラムに示すように、一般に測定 VI は、軸クラスタを介してデータ集録 VI の出力とグラフに直接接続することができます。



測定例としては、次のようなものがあります。

- 振幅スペクトルの例
- シミュレーションによるダイナミック信号解析の例
- 全高調波歪み（THD）の例

次の例は、ナショナルインスツルメンツのハードウェアで使用することができます。

- 簡易スペクトルアナライザとスペクトルアナライザ — どちらも、すべてのアナログ入力ハードウェアで動作します（高品質の測定結果を得るためにはダイナミック信号集録用ハードウェアを使用してください）。
- ダイナミック信号アナライザとネットワークアナライザ — どちらも、ダイナミック信号集録（DSA）ハードウェアで動作します。

学習内容

- 測定VIについて、およびこれらを使用してさまざまな信号処理を行う方法
- 適当な単位で時間領域の信号の周波数（振幅と位相）スペクトルを計算する方法
- 適当な単位でシステムの信号と応答信号を処理することによりシステムの周波数応答を計算する方法
- コヒーレンス性関数を計算する方法、およびこの関数を使用して周波数応答の測定値を解釈する方法
- 信号に含まれる全高調波歪みを測定する方法

スペクトル解析

信号の振幅と位相のスペクトルを計算する

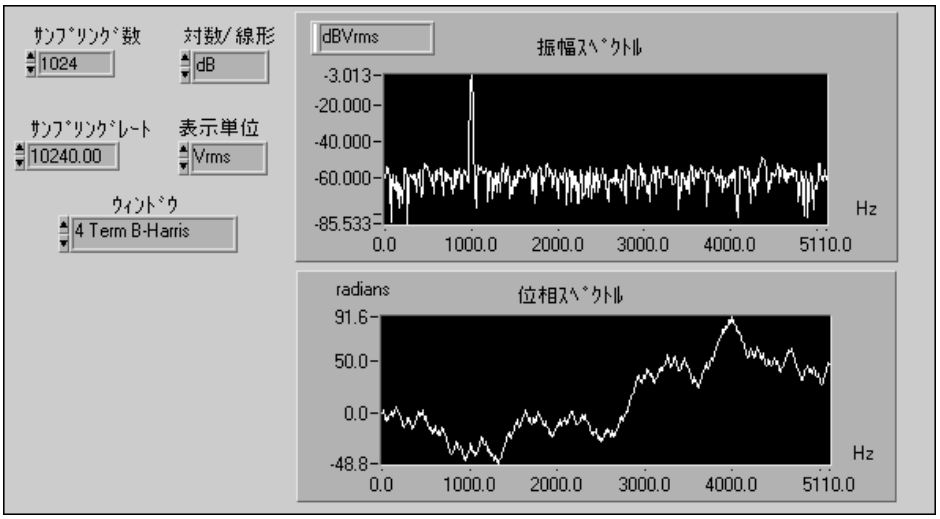
多くのアプリケーションでは、信号の周波数成分を知ることによって、信号を生成したシステムについて洞察することができます。得られた情報を使用することで、音の周波数成分の解析、計測器のキャリブレーション、機械の各部から発生するノイズや振動の量の評価などを行えます。次の作業では、Amplitude and Phase Spectrum VIを使用して信号の振幅や位相を測定する方法を説明します。

作業 15-1.　Amplitude and Phase Spectrum VI を使用する

この作業では、信号の振幅と位相のスペクトルを計算することが目的です。

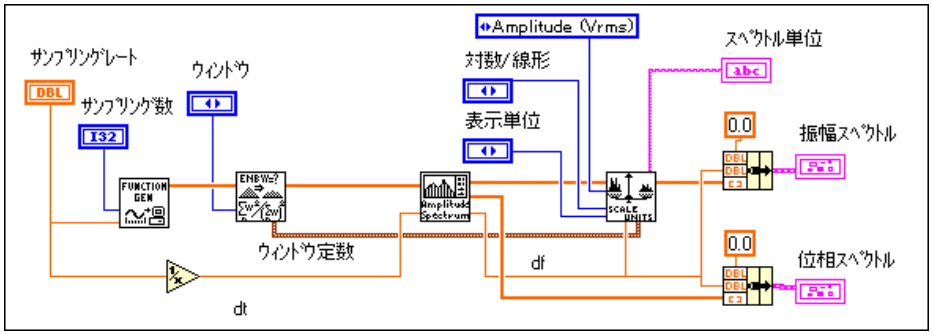
フロントパネル

1. `examples¥analysis¥measure¥measxmpl.llb` ライブラリに入っているAmp Spectrum Example VIを開きます。Simple Function Generator VIにより信号が生成されます。このVIは、付加的なホワイトノイズを含むマルチ関数発生器をシミュレーションします。



ブロックダイアグラム

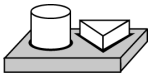
2. ブロックダイアグラムを開いて確認します。



Amplitude and Phase Spectrum VIは、時間領域の信号の振幅スペクトルと位相スペクトルを計算します。次に、このVIへの接続について説明します。

時間領域の入力信号はシグナル（V）制御器に加えられます。Amp Spectrum Mag (Vrms)出力とAmp Spectrum Phase (radians)出力にはそれぞれ、入力信号の振幅と位相のスペクトルが得られます。Spectrum Unit Conversion VIは、Amplitude and Phase Spectrumの元のVrms出力を、よく使用される別の単位（Vrms、Vpk、Vrms^2、Vpk^2、Vrms$\sqrt{Hz}$、Vpk$\sqrt{Hz}$、Vrms^2/Hz、およびVpk^2/Hz）に変換します。最後の4つの単位は、振幅スペクトル密度（Vrms$\sqrt{Hz}$、Vpk$\sqrt{Hz}$）とパワースペクトル密度（Vrms^2/Hzと Vpk^2/Hz）です。Scaled Time Domain Window VIからの**ウィンドウ定数**出力クラスタには、選択されたウィンドウ用のスペクトル密度の測定に必要な定数が含まれています。

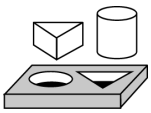
3. VIを実行します。
4. シミュレーションの周波数と波形のタイプおよび信号の振幅とノイズレベルを変更できるように、Simple Function Generatorフロントパネルを開いた状態で、Amp Spectrum Exampleを連続的に実行し、振幅スペクトルの変化を確認してください。



これで作業 15-1 は完了です。

システムの周波数応答を計算する

個々の信号の周波数成分の測定はそれ自体も役に立ちますが、システムの周波数応答は、電気部品のインピーダンスから動的な構造体の自然振動周波数の解析まで、あらゆる種類のネットワークの動作解析に広く利用されています。周波数応答には、与えられた入力に対するネットワークの応答特性が完全に示されています。

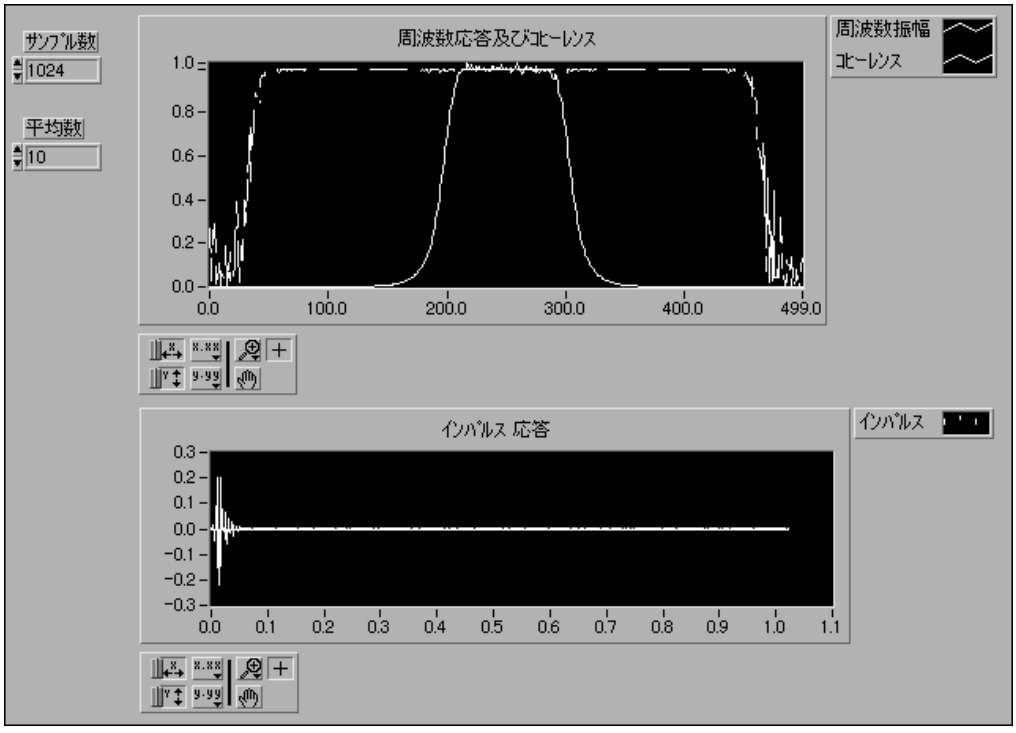


作業 15-2.　周波数応答とインパルス応答を計算する

ここでは、システムの周波数応答とインパルス応答を計算すること、およびコヒーレンス関数を計算し、この関数を使用して周波数応答の測定値の有効性をチェックする方法を理解することが目的です。

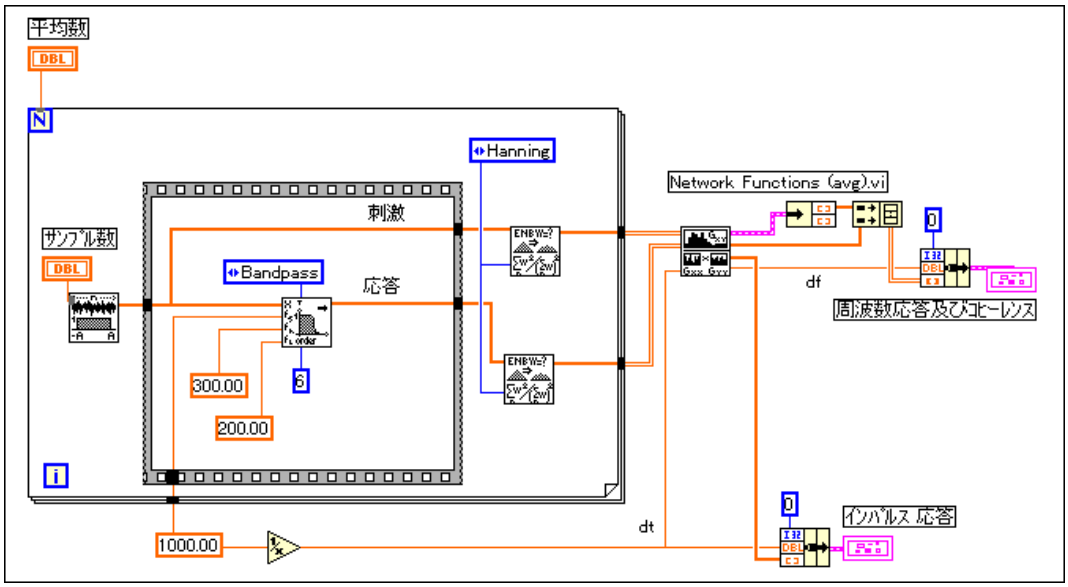
フロントパネル

1. 新しいフロントパネルを開き、次の図のようなオブジェクトを追加します。このフロントパネルには、バンドパスフィルタの周波数応答振幅とインパルス応答関数が表示されます。コヒーレンス関数もやはりスペクトル測定値ですので、周波数応答振幅と同じスケール上にプロットされます。



ブロックダイアグラム

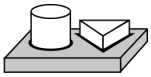
2. ブロックダイアグラムを開き、次の図のように修正します。ここでは、バンドパスフィルタ（Butterworth Filter VI）のシステム応答を測定します。このためには、システムへの刺激信号としてホワイトノイズ（Uniform White Noise VI）を渡し、システム応答であるフィルタ出力を収集します。刺激信号と応答はいずれもHanningウィンドウ（Scaled Time Domain Window VI）によりウィンドウ処理され、全システムのフレーム数または**平均値**がモニタされます。続いて刺激信号と応答データは Network Functions(avg) VI に送られ、システムの周波数応答に関係するすべての計算が実際に行われます。



Network Functions (avg) VI は、周波数応答（振幅と位相）、クロスパワースペクトル（振幅と位相）、コヒーレンス関数、およびインパルス応答を計算します。入力データと出力データのフレーム数を大きくする（フロントパネル上の平均値を大きくする）と、システム応答関数の計算値が向上します。このダイアグラムでは、周波数応答振幅、コヒーレンス、およびインパルス応答だけがプロットされます。

コヒーレンス関数は出力信号と入力信号の相関性を測定しますので、周波数応答の見積値の有効性を示すものが得られます。ノイズが加わったりシステムの動作が特定の周波数で非線形でない場合は、これらの周波数でコヒーレンス関数が均等レベルよりも低くなる原因となります。相関性のないシステムノイズの場合は、得られる平均値の数が多くなるほど、コヒーレンス関数の一様性が高くなり、周波数応答の計算は改善されます。最後

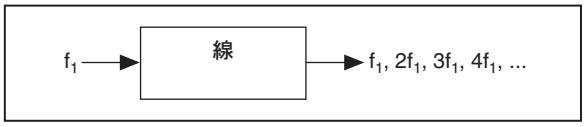
に、コヒーレンス関数についてもうひとつ覚えておいていただきたいのは、コヒーレンス関数が定義されるのは、入出力データの複数のフレームの平均値を計算する場合だけであるということです。平均値が 1 つしかない場合は、周波数応答推測が低い場合でも、コヒーレンスはすべての周波数で均一になってしまいます。



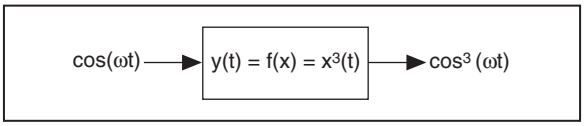
これで作業 15-2 は完了です。

高調波歪み

特定の周波数の信号 *x(t)* (たとえば f1) を非線形システムに通した場合、システムの出力には、入力周波数 (f1) だけでなくその高調波 (f2 = 2*f1、f3 = 3*f1、f4 = 4*f1 など) も含まれています。生成する高調波の数とそれぞれに対応する振幅は、システムの非線形性の程度によって決まります。一般に、非線形であるほど高調波は多く、線形であるほど高調波は少なくなります。



非線形システムの例としては、出力 *y(t)* が入力信号 *x(t)* の 3 乗であるようなシステムがあります。



したがって、次のような入力の場合、

$$x(t) = \cos(\omega t),$$

出力は、次のようになります。

$$x^3(t) = 0.5*\cos(\omega t) + 0.25*[\ \cos(\omega t) + \cos(3\omega t)\ ]$$

このように、出力には、入力の基本周波数 ω だけでなく第 3 高調波 3ω も含まれています。

全高調波歪み

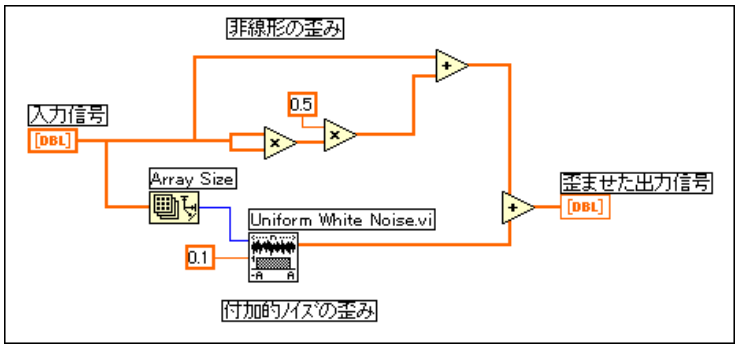
システムで生成する非線形歪みを測定するには、基本波の振幅に対する、システムで生成する高調波の振幅の相対比を測定する必要があります。高調波歪みは、基本波の振幅と比較した、高調波の振幅の相対的な測定値です。基本波の振幅が A_1、高調波の振幅が A_2（第 2 高調波）、A_3（第 3 高調波）、A_4（第 4 高調波）、... A_N（第 *N* 高調波）である場合、全高調波歪み（THD）は次の式で求められます。

$$THD = sqrt (A_1^2 + A_2^2 + A_3^2 + ... A_N^2)/A_1$$

全高調波歪みのパーセンテージ（%THD）は、次のようになります。

$$\% THD = 100*sqrt (A_1^2 + A_2^2 + A_3^2 + ... A_N^2)/A_1$$

次の作業では、正弦波を生成して非線形システムに通します。非線形システムのブロックダイアグラムを次に示します。



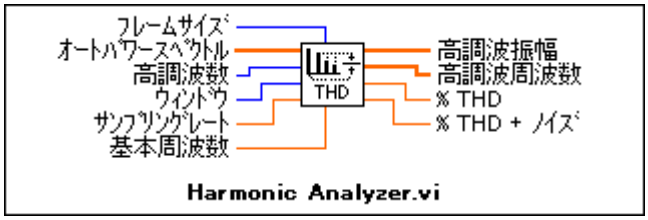
このブロックダイアグラムから、入力が x(t) = cos(ωt) である場合に出力が次のようになることを確認してください。

$$y(t)= \cos(\omega t) + 0.5\cos^2(\omega t) + 0.1n(t)$$
$$= \cos(\omega t) + [1 + \cos(2\omega t)]/4 + 0.1n(t)$$
$$= 0.25 + \cos(\omega t) + 0.25\cos(2\omega t) + 0.1n(t)$$

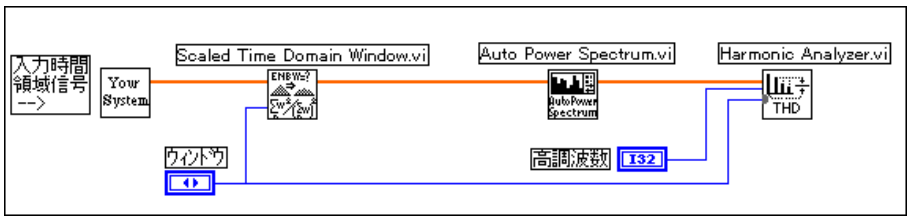
したがって、この非線形システムでは、付加的な DC 成分と基本波の第 2 高調波が生成されます。

Harmonic Analyzer VIを使用する

Harmonic Analyzer VIを使用すると、非線形システムの出力信号に含まれる%THDを計算することができます。このVIは、入力に加えられたパワースペクトルに含まれる基本波成分と高調波成分（振幅および対応する周波数）を見つけ、全高調波歪みのパーセンテージ（%THD）と全高調波歪みにノイズを加えたもののパーセンテージ（%THD+ノイズ）を計算します。Harmonic Analyzer VIへの接続を次に示します。



このVIを使用するには、THDを計算したい信号のパワースペクトルをこのVIに与える必要があります。したがって、この例では次のように接続する必要があります。



Scaled Time Domain Window VIは、非線形システム（Your System）の出力y(t)に対してウィンドウを適用します。つぎにこれがAuto Power Spectrum VIに渡され、このVIがパワースペクトルy(t)をHarmonic Analyzer VIに送り、さらにこのVIが高調波の振幅と周波数、THD、および%THDを計算します。

VIで検出したい高調波の数を**高調波数**制御器で指定できます。高調波の振幅および対応する周波数は、配列表示器**高調波振幅**と**高調波周波数**に返されます。

☞ **注**　**高調波数制御器で指定される数には基本波も含まれます。したがって、高調波数制御器に2という値を入力した場合は、基本波（たとえば周波数f1）と第2高調波（周波数f2 = 2*f1）を調べることを意味します。また、Nという値を入力した場合、VIは、基本波と対応する(N-1)個の高調波を調べます。**

その他のいくつかの制御器について以下に説明します。

基本周波数は、基本波成分の予測周波数です。（デフォルト値の）0のままにした場合、VIはDCでない成分のうち最も振幅が大きいものの周波数を、基本波の周波数として使用します。

ウィンドウは、元の時間信号に適用するウィンドウのタイプです。ウィンドウは、Scaled Time Domain Window VIで選択したウィンドウです。THDを正確に計算するには、ウィンドウ関数を選択することを推奨します。デフォルトでは均一なウィンドウとなります。

サンプリングレートは、Hz単位で示した入力のサンプリングレートです。

%THD+ノイズ出力については、他にいくつか説明する必要があります。**%THD+ノイズ**の計算は、高調波電力にノイズ電力も加算される点を除いては**%THD**の場合とほとんど同じであり、次の式で与えられます。

$$\% \text{ THD + Noise} = 100*\text{sqrt}(\text{sum(APS)})/A1$$

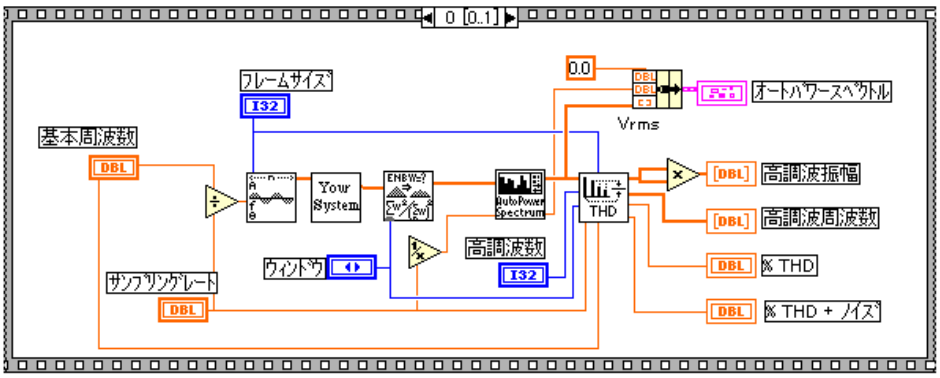
ただし、sum(APS)はAuto Power Spectrumの要素の合計値からDCに近い要素と基本波周波数の指標に近い要素を引いたものです。

作業 15-3.　高調波歪みを計算する

ここでは、Harmonic Analyzer VIを使用して高調波歪みを計算することが目的です。

ブロックダイアグラム

1. examples¥analysis¥measure¥measxmpl.llbのTHD Example VIを開き、ブロックダイアグラムを表示します。

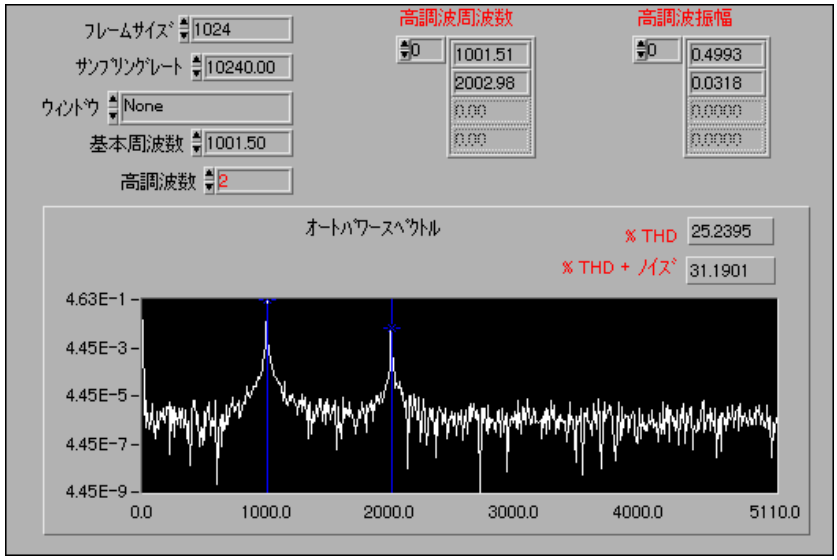


この中のいくつかは、すでにお馴染みのものです。Your Systemは、前に表示された非線形システムです。このシステムの出力はウィンドウ処理され、パワースペクトルが計算されて、Harmonic Analyzer VIに与えられます。

Sine Wave VIは、**基本周波数**制御器で指定された周波数の基本波を生成します。

フロントパネル

2. フロントパネルを表示します。下部に、非線形システムの出力のパワースペクトルのプロットが表示されています。右上には、基本波と高調波の周波数と振幅の配列表示器があります。配列のサイズは、**高調波数**制御器に入力された値によって決まります。



3. **基本周波数**を 1000 に、**高調波数**を 2 に変更して VI を数回実行します。実行のたびに、出力表示器（**高調波周波数**、**高調波振幅**、**%THD**、**%THD＋ノイズ**）の値を確認してください。

   VI を実行するたびに異なる値が得られるのはなぜでしょうか？

   %THD と %THD+ ノイズで、値が大きいのはどちらですか？　理由を説明できますか？

4. **ウィンドウ**制御器の選択を変更して VI を実行し、パワースペクトル内のピークを調べてください。

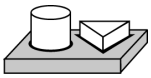
   ピークの幅が最も狭いのはどのウィンドウですか？　最も広いのはどれですか？　その理由を説明できますか？

5. 基本周波数を3000に変更してVIを実行します。

   エラーとなるのはなぜでしょうか？

   ヒント：ナイキスト周波数と高調波の周波数との間の関係を考えてください。

6. 確認が終わったら、VIを閉じてLabVIEWを終了します。



これで作業 15-3 は完了です。

まとめ

測定 VI は一般的な測定タスクを実行できることがわかりました。これらのタスクの中には、信号の振幅と位相スペクトル、高調波歪みの量の計算が含まれます。他の VI は転送関数、インパルス応答、入力信号と出力信号との間のクロスパワースペクトルなどのシステムプロパティを計算します。これらの測定を実行する既成の VI は、**解析→測定**サブパレットにあります。

16

フィルタ処理

この章では、無限インパルス応答フィルタ（infinite impulse response filter: IIR）、有限インパルス応答フィルタ（finite impulse response filter: FIR）、および非線形フィルタを使用して不要な周波数をフィルタ処理する方法を説明します。解析フィルタ VI の使用方法については、examles¥analysis¥fltrxmpl.llb の例を参照してください。

デジタルフィルタ関数の概要

アナログフィルタの設計は、電子関係の設計で最も重要な部分のひとつです。テスト済みの簡単なフィルタの設計例を売り物にしているアナログフィルタ設計に関する本もありますが、フィルタの設計には高度な数学的知識とフィルタに影響を与えるシステムの内部処理についての理解が要求されるため、多くの場合は専門家しかフィルタを設計できません。

サンプリングやデジタル信号処理を行うための最新のツールを使用すると、プログラムにより変更できる性能や柔軟性が要求されるアプリケーションにおいて、アナログフィルタをデジタルフィルタに置き換えることができます。こうしたアプリケーションにはオーディオ、通信、医療上の監視などがあります。

デジタルフィルタは、アナログフィルタに比べて次のような利点があります。

- ソフトウェアでプログラム可能。
- 安定性と再現性が高い。
- 温度や湿度によるドリフトがなく、高精度なパーツを必要としない。
- コストパフォーマンスがきわめて高い。

LabVIEW のデジタルフィルタを使用すると、フィルタの次数、カットオフ周波数、リプル量、停止帯域での減衰量などの引数を制御することができます。

この項で説明するデジタルフィルタ VI は、仮想計測の概念に基づいています。これらの VI は、デザインに関する問題演算、メモリ管理、実際のデータ内部のフィルタ処理などを、ユーザにわかりやすいかたちで処理します。ユーザがデジタルフィルタやデジタルフィルタ理論の専門家でなくても、データを処理することができます。

フィルタの引数や、これらの引数と入力引数との関係をよく理解していただくため、サンプリング定理について次に説明します。

サンプリング理論では、サンプリング周波数が時間領域信号周波数の最高周波数の2倍以上であれば、等間隔の離散的サンプルから連続的な時間信号を再生できることを示しています。対象となる時間信号を等間隔の区間Δtでサンプリングすると情報が失われないと仮定します。Δt引数をサンプリング間隔といいます。

サンプリングレートまたはサンプリング周波数*fs*は、次のようにサンプリング間隔から求められます。

$$f_s = \frac{1}{\Delta t}$$

つまり、サンプリング理論によると、デジタルシステムが処理可能な最高周波数は次のようになります。

$$f_{Nyq} = \frac{f_s}{2}$$

システムが処理可能な最高周波数をナイキスト周波数と言います。これはデジタルフィルタにも当てはまります。たとえば、サンプリング間隔が

$$\Delta t = 0.001 \text{ sec}$$

の場合、サンプリング周波数は

$$f_s = 1{,}000 \text{ Hz}$$

となり、システムが処理可能な最高周波数は

$$f_{Nyq} = 500 \text{ Hz}$$

となります。

次のようなフィルタ処理動作は、フィルタ設計技術に基づいています。

- スムージングウィンドウ
- 無限インパルス応答（IIR）または再帰デジタルフィルタ
- 有限インパルス応答（FIR）または非再帰デジタルフィルタ
- 非線形フィルタ

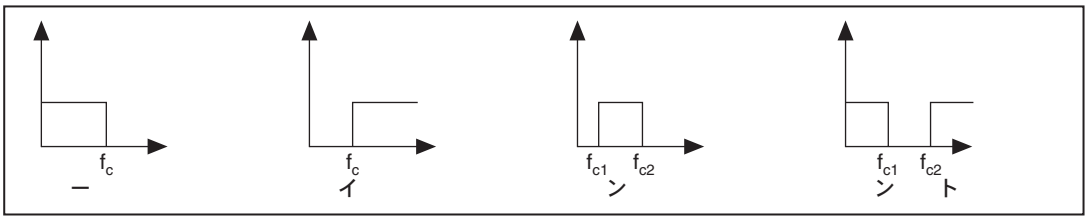
本章ではこれ以降IIR、FIR、および非線形的な技法の理論的背景を簡単に説明し、各技法に対応するデジタルフィルタ VI について説明します。スムージングウィンドウを実行するVIについての詳細は、「第14章　スムージングウィンドウ」を参照してください。

理想的なフィルタ

フィルタは、不要な周波数を変更したり除去します。通したり減衰させる周波数の範囲によって、フィルタは以下のタイプに分類できます。

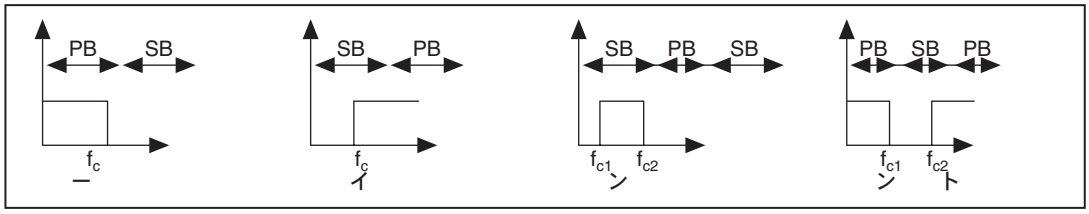
- **ローパスフィルタ**は、低い周波数を通し、高い周波数を減衰させます。
- **ハイパスフィルタ**は、高い周波数を通し、低い周波数を減衰させます。
- **バンドパスフィルタ**は、特定の周波数帯を通します。
- **バンドストップフィルタ**は、特定の周波数帯を減衰させます。

これらのフィルタの理想的な周波数応答を次に示します。



図からわかるように、ローパスフィルタはf_c未満のすべての周波数を通すのに対し、ハイパスフィルタはfcを超えるすべての周波数を通します。バンドパスフィルタはf_c1とf_c2の間のすべての周波数を通すのに対し、バンドストップフィルタはf_{c1}とf_c2の間のすべての周波数を減衰させます。周波数点f_c、ff_{c1}、f_{c2}をフィルタのカットオフ周波数と言います。フィルタを設計する場合は、これらのカットオフ周波数を指定する必要があります。

フィルタを通過する周波数範囲を、フィルタの**パスバンド**（PB）と言います。理想的なフィルタのパスバンドにおけるゲインは1（0 dB）で、信号の振幅は大きくも小さくもなりません。**ストップバンド**（SB）は、フィルタを全く通過せず排除（減衰）される周波数範囲のことです。各タイプのフィルタのパスバンドとストップバンドを次に示します。

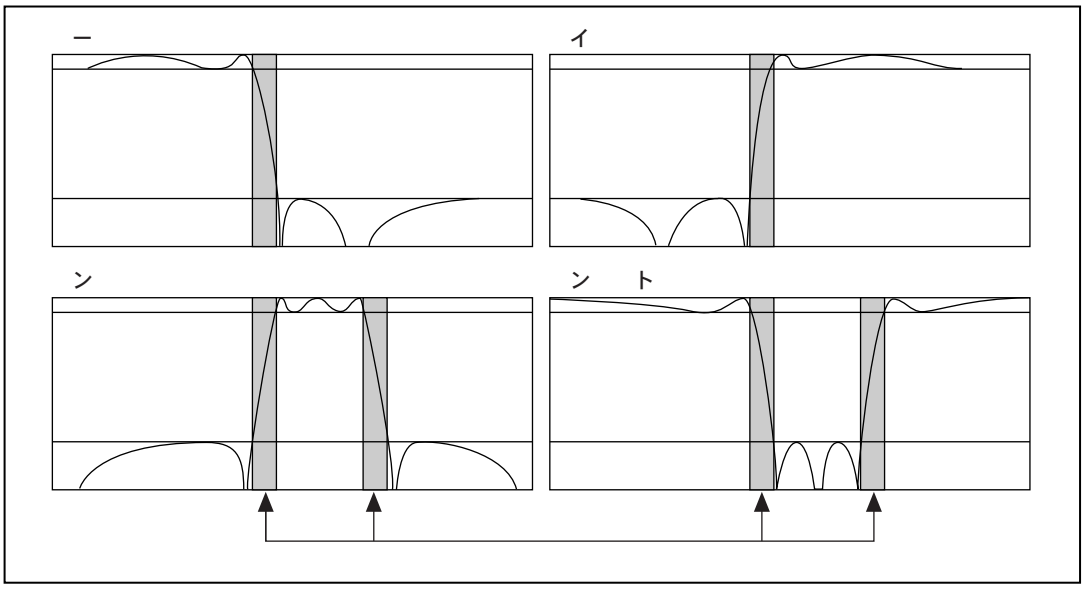


ローパスフィルタとハイパスフィルタにはパスバンドとストップバンドが各1つずつなのに対し、バンドパスフィルタにはパスバンドが1つとストップバンド2つが、バンドストップフィルタにはパスバンドが2つとストップバンド1つがあることに注意してください。

実際の（非理想的）フィルタ

遷移帯域

理想的には、フィルタのゲインは、パスバンドでは単位ゲイン（0 dB）で、ストップバンドでは0ゲイン（–無限大dB）でなければなりません。しかし、実際の実行ではこのような基準をすべて満たすことは不可能です。実際に、パスバンドとストップバンドの間には有限の遷移領域が存在します。この領域では、フィルタのゲインはパスバンドの1（0 dB）からストップバンドの0（–無限大dB）まで徐々に変化していきます。次の図に、さまざまな非理想的フィルタのパスバンド、ストップバンド、および遷移領域（TR）を示します。この場合、パスバンドはフィルタのゲインの変化が0 dBから–3 dBの周波数範囲内の領域となっていることに注意してください。



パスバンドのリプルとストップバンドの減衰

多くのアプリケーションでは、パスバンド内のゲインが一定でなく多少変化してもかまいません。パスバンドにおけるこのような変化を、**パスバンドのリプル**と言い、実際のゲインと望ましい均一なゲインとの差を示します。**ストップバンドの減衰**は、実際には無限大にすることは不可能ですので、満足できる値を指定する必要があります。パスバンドのリプルとストップバンドの減衰はどちらもデシベル（dB）単位で測定され、次のように定義されます。

$$dB = 20*\log_{10}(A_o(f)/A_i(f))$$

ただし、$\log_{10}$は10を底とする対数を示し、$A_i(f)$と$A_o(f)$はそれぞれ、フィルタ処理を行う前後の特定の周波数fの振幅です。

たとえば、パスバンドのリプルが–0.02 dBである場合の式は

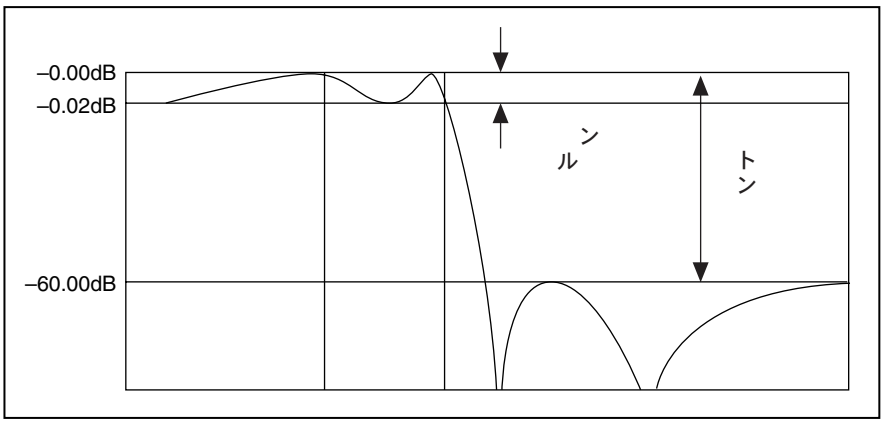
$$-0.02 = 20*\log_{10}(A_o(f)/A_i(f))$$
$$A_o(f)/A_i(f) = 10^{-0.001} = 0.9977$$

となり、入力と出力の振幅の比率がほぼ一定であることを示します。

ストップバンドの減衰がー 60 dB である場合は

$$-60 = 20*\log_{10}(A_o(f)/A_i(f))$$
$$A_o(f)/A_i(f) = 10^{-3} = 0.001$$

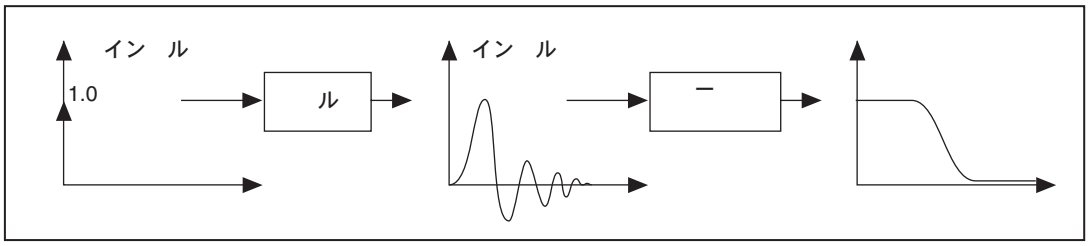
となり、出力振幅が入力振幅の 1/1000 になることを示します。目盛りは正確ではありませんが、次の図にこの概念を示します。



☞ **注**　減衰は普通、「マイナス」という語を付けずにデシベルで表現されますが、通常は負の dB 値になります。

IIR フィルタと FIR フィルタ

フィルタを分類する方法としてはこの他に、インパルス応答に基づく方法があります。では、インパルス応答とは何でしょうか？（$x[0] = 1$ でかつ $i \neq 0$ のすべての点で $x[i] = 0$ であるような）インパルスを入力に加えた場合のフィルタの応答を、そのフィルタの**インパルス応答**と言います（次の図を参照）。インパルス応答のフーリエ変換をそのフィルタの**周波数応答**と言います。フィルタの周波数応答は、さまざまな周波数に対するフィルタの出力、すなわちさまざまな周波数におけるフィルタのゲインを示します。理想的なフィルタのゲインは、パスバンドでは 1 でストップバンドでは 0 でなければなりません。したがって、パスバンド内ではすべての周波数が「そのまま」出力され、ストップバンド内では一切の周波数が出力されません。



フィルタのインパルス応答が有限時間の後 0 になる場合、このようなフィルタを**有限インパルス応答（FIR）**フィルタと言います。これに対しインパルス応答がいつまでも存在する場合、このようなフィルタを**無限インパルス応答（IIR）**フィルタと言います。インパルス応答が有限になるか無限になるか（フィルタが FIR または IIR のどちらになるか）は、出力の計算方法によって決まります。

FIR フィルタと IlR フィルタの基本的な違いは、FIR フィルタでは現在と過去の入力値によってのみ出力が決まるのに対し、IIR フィルタでは現在と過去の入力値だけでなく過去の出力値の影響も受けることです。

たとえばスーパーマーケットのキャッシュレジスタを考えてみましょう。顧客が現在購入する品目の価格を *x*[*k*]、前の品目の価格を *x*(k-1) とします。ただし $1 \le k \le N$ であり *N* は品目の合計数です。キャッシュレジスタは各品目の価格を加算して、「その時点での」合計額を計算します。この、*k* 番目の品目までの「その時点での」合計額 *y*[k] は、次のようになります。

$$y[k] = x[k] + x[k–1] + x[k–2] + x[k–3] + ...+ x[1] \qquad (16\text{-}1A)$$

したがって、*N* 個の品目の合計額は *y*[*N*] になります。*y*[*k*] は *k* 番目の品目までの合計額であり、*y*[*k*-1] は (*k*-1) 番目の品目までの合計額なので、式 16-1A は次のように書き換えることができます。

$$y[k] = y[k–1] + x[k] \qquad (16\text{-}1B)$$

8.25% の売上税を加算すると、式 16-1A と 16-1B は次のように書き換えることができます。

$$y[k] = 1.0825x[k] + 1.0825x[k–1] + 1.0825\,x[k–2] + 1.0825x[k–3] + ... + 1.0825x[1] \qquad (16\text{-}2A)$$

$$y[k] = y[k–1] + 1.0825x[k] \qquad (16\text{-}2B)$$

式16-2Aと16-2Bは、キャッシュレジスタの動作を説明する上では全く同じです。違うのは、16-2Aは入力の観点からのみ作成されているのに対して、16-2Bは入力と出力の両方の観点から作成されている点です。式16-2Aは**非再帰**的、つまりFIRとして作成されています。式16-2Bは**再帰**的、つまりIIRとして作成されています。

フィルタ係数

式16-2Aで、各項にかける定数は1.0825になっています。式16-2Bの乗数は、（$y[k\text{-}1]$に対しては）1、（$x[k]$に対しては）1.0825になっています。これらの乗数を、フィルタの**係数**と言います。また、IIRフィルタの入力にかける係数を**順係数**、出力にかける係数を**逆係数**と言います。

フィルタの動作を説明する16-1A、16-2A、16-1B、または16-2Bの形の式を、**差分**方程式と言います。

IIRフィルタの短所は、位相応答が非線形であることです。単に信号を監視するだけのような位相情報を必要としないアプリケーションの場合はIIRフィルタでもかまいませんが、線形の位相応答が必要なアプリケーションの場合はFIRフィルタを使用する必要があります。IIRフィルタは、その再帰的な性質のために設計や実行がより難しくなります。

無限インパルス応答フィルタ

無限インパルス応答フィルタ（IIR）は、理論的にはインパルス応答の長さが無限であるようなデジタルフィルタです。IIRフィルタを特徴付ける一般的な差分方程式は、次のようになります。

$$y_i = \frac{1}{a_0}\left(\sum_{j=0}^{N_b-1} b_j x_{i-j} - \sum_{k=1}^{N_a-1} a_k y_{i-k}\right) \qquad (16\text{-}3)$$

ただし、N_bは**順**係数（b_j）、Naは**逆**係数（a_k）の数です。

ほとんどのIIRフィルタ設計（およびLabVIEWのIIRフィルタのすべて）では、係数a_0は1になります。現在のサンプルの指標iにおける出力サンプルは、スケーリングされた現在の入力と過去の入力（x_iおよび$\neq 0$の場合の$x_i\text{-}j$）、およびスケーリングされた過去の出力（$y_{i\text{-}k}$）の合計値です。このためIIRフィルタは、再帰フィルタまたは移動平均の自動回帰（autoregressive moving-average: ARMA）フィルタと呼ばれます。

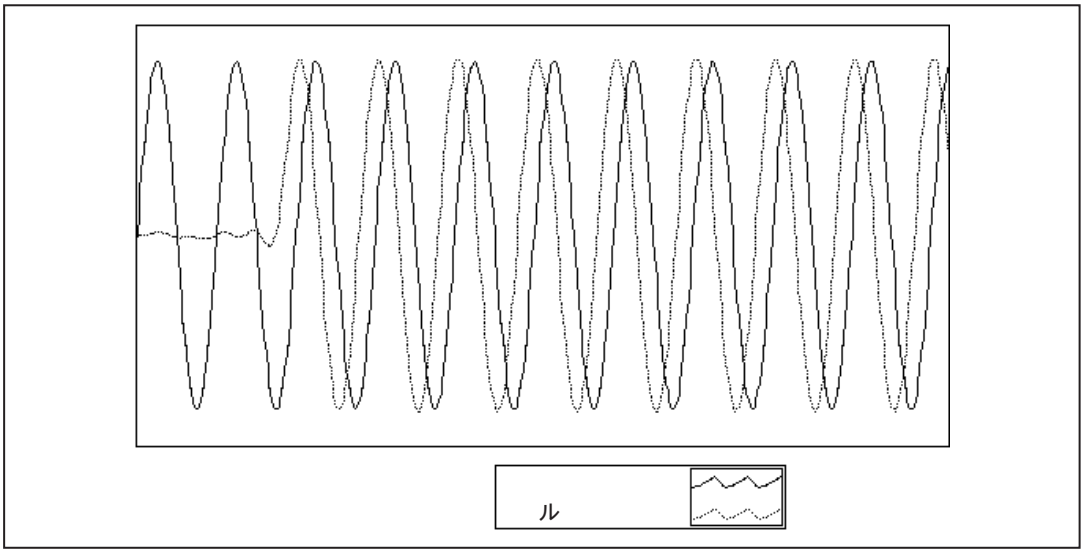
インパルス（$x_0 = 1$でかつ$i \neq 0$のすべての点で$x_i = 0$）に対する一般のIIRの応答を、そのフィルタのインパルス応答と呼びます。式16-3に示すフィルタのインパルス応答の長さは、係数が0でない場合は実際に無限大になりますが、現実のフィルタアプリケーションにおける安定したIIRフィ

ルタのインパルス応答は、有限数のサンプルの中で0に向かって減衰していきます。

LabVIEWに含まれるIIRフィルタには、次のような性質があります。

- 式16-3から得られる負の指標は、最初にVIを呼び出したときには0になることが想定されます。
- フィルタの初期状態は0（負の指標）になることが想定されているため、フィルタが安定状態になる前に、フィルタの次数に比例する過渡現象が発生します。ローパスフィルタとハイパスフィルタの場合の過渡応答の持続時間または遅れは、フィルタの次数に等しくなります。
- 遅延 = 次数
- バンドパスフィルタとバンドストップフィルタの場合の過渡応答の持続時間は、フィルタの次数の2倍になります。
- 遅延 = 2* 次数

連続的に呼び出しを行う場合にこのような過渡応答をなくすには、ステータスメモリを有効にします。ステータスメモリを有効にするには、VIの**初期化／継続**制御器をTRUE（連続フィルタ処理）に設定します。



フィルタ処理されるシーケンスに含まれる要素数は、入力シーケンス内の要素数に等しくなります。

フィルタ処理の完了時、フィルタはフィルタの内部ステータス値を保持します。

デジタルIIRフィルタが有限インパルス応答（FIR）フィルタよりすぐれているのは、同じようなフィルタ処理を行うのに通常IIRフィルタの方が必要な係数が少なくて済む点です。したがって、IIRフィルタの方が実行スピードははるかに速い上、その場で実行されるため余計なメモリが不要です。

IIRフィルタの短所は、位相応答が非線形であることです。単に信号を監視するだけのような位相情報を必要としないアプリケーションの場合はIIRフィルタでも構いませんが、線形の位相応答が必要なアプリケーションの場合はFIRフィルタを使用する必要があります。

カスケードフォームIIRフィルタ処理

式16-4により直接定義されたストラクチャを使用して実装されたフィルタを、ディレクトフォームIIRフィルタと言います。**ディレクトフォーム**で実装した場合は、係数の量子化、演算上の制約、精度的な限界などに起因する誤差の影響を受けやすくなります。また、安定するように設計されたフィルタが、フィルタの次数に比例して係数の長さが大きくなるに従って不安定になることがあります。

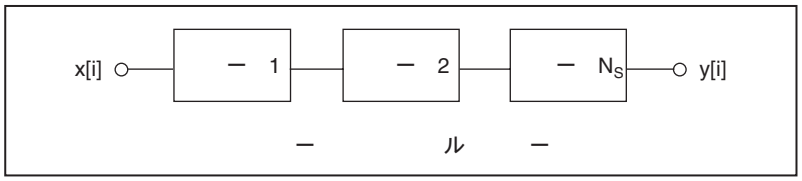
誤差の影響を受けにくいストラクチャを実現するには、ディレクトフォームの変換関数を次数の低い部分、つまりフィルタのステージに分割します。($a_0 = 1$とした場合の)式16-4で得られるフィルタのディレクトフォーム変換関数は、次のように、z変換の比として記述することができます。

$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_{N_b - 1} z^{-(N_b - 1)}}{1 + a_1 z^{-1} + \ldots + a_{N_a - 1} z^{-(N_a - 1)}} \qquad (16\text{-}4)$$

式16-4を2次の項に因数分解すると、フィルタの変換関数は2次のフィルタ関数の積になります。

$$H(z) = \prod_{k=1}^{N_s} \frac{b_{0k} + b_{1k} z^{-1} + b_{2k} z^{-2}}{1 + a_{1k} z^{-1} + a_{2k} z^{-2}} \qquad (16\text{-}5)$$

ただし$N_s = \lfloor N_a/2 \rfloor$は、$\leq N_a/2$と$N_a \geq Nb$の間の最大の整数です（$N_s$は**ステージ数**）。この新しいフィルタストラクチャは、2次フィルタを**カスケード**接続したものとして表現することができます。



それぞれのステージは、算術演算と遅延要素（内部フィルタステータス）を最少数に抑える必要があるため、**ディレクトフォームII**フィルタストラクチャを使用して実行されています。各ステージには、1つの入力、1つの出力、および2つの過去の内部ステータス（$s_k[i\text{-}1]$と$s_k[i\text{-}2]$）があります。

入力シーケンスに含まれるサンプル数がnである場合、フィルタ処理は次の式のように進行します。

$$y_0[i] = x[i],$$

$$s_k[i] = y_{k-1}[i-1] - a_{1k}s_k[i-1] - a_{2k}s_k[i-2],\ k = 1, 2,..., N_s$$

$$y_k[i] = b_{Ok}s_k[i] + b_{1k}s_k[i-1] + b_{2k}s_k[i-2],\ k = 1, 2,..., N_s$$

$$y[i] = y_{Ns}[i]$$

ただし各サンプルについてiの範囲は以下の通りになります。

$$i = 0, 1, 2,...,n-1$$

カットオフ周波数が1つしかないフィルタ（ローパスとハイパス）では、2次フィルタのステージを直接設計することができます。IIR ローパスフィルタまたはハイパスフィルタ全体に、カスケード接続された2次フィルタが含まれています。

カットオフ周波数が2つあるフィルタ（バンドパスとバンドストップ）では、4次フィルタのステージの方が形としてはより自然です。IIRバンドパスフィルタまたはバンドストップフィルタ全体に、カスケード接続された4次フィルタが含まれています。4次ステージの場合のフィルタ処理は次の式のように進行します。

$$y_0[i] = x[i],$$

$$s_k[i] = y_{k-1}[i-1] - a_{1k}s_k[i-1] - a_{2k}s_k[i-2] - a_{3k}s_k[i-3] - a_{4k}s_k[i-4],\quad k = 1, 2,..., N_s$$

$$y_k[i] = b_{0k}s_k[i] + b_{1k}s_k[i-1] + b_{2k}s_k[i-2] + b_{3k}s_k[i-3] + b_{4k}s_k[i-4],\quad k = 1, 2,..., N_s$$

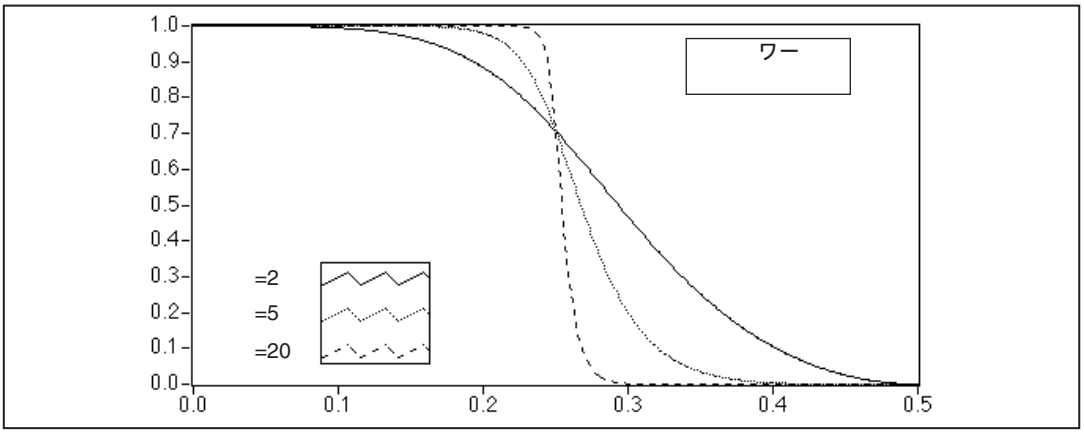
$$y[i] = y_{Ns}[i].$$

4 次フィルタステージの場合は、$N_s = \lfloor (N_a + 1)/4 \rfloor$ になることに注意してください。

バタワースフィルタ

バタワースフィルタの周波数応答の特徴は、すべての周波数における応答が滑らかなことと、指定されたカットオフ周波数からの減少が単調であることです。バタワースフィルタは平坦性が最も高く、パスバンドでの均一性とストップバンドでの 0 が理想的な応答となっています。パワーが 1/2 となる、または 3 dB 低下する周波数は、指定したカットオフ周波数と一致します。

次の図は、ローパスバタワースフィルタの応答を示します。バタワースフィルタの長所は滑らかで単調に減少していく周波数応答です。ユーザがカットオフ周波数を設定すると、LabVIEW はフィルタの次数に比例して遷移領域の**勾配**を設定します。バタワースフィルタの次数を高くしていくと、理想的なローパスフィルタの応答に近付きます。

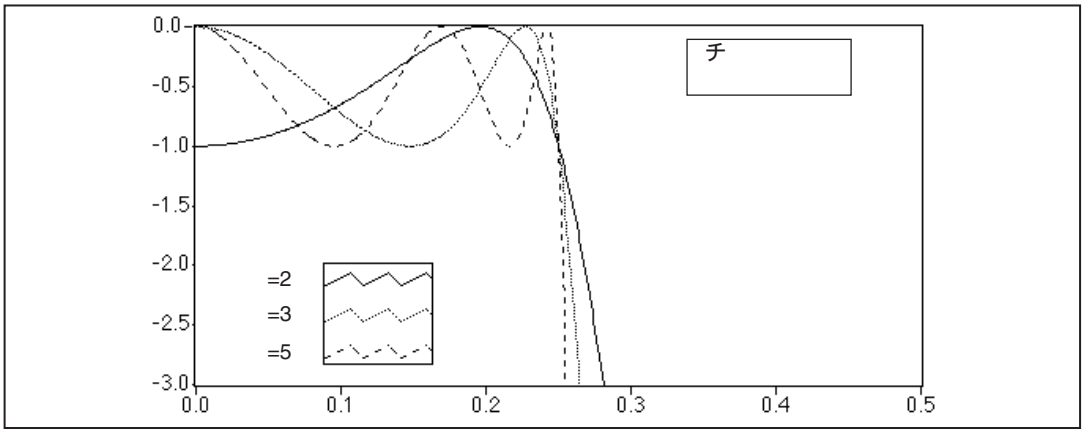


チェビシェフフィルタ

バタワースフィルタでは、パスバンド（スペクトル内の必要な部分）とストップバンド（スペクトル内の不要な部分）の間のロールオフ特性が緩やかなため、理想的なフィルタに近い応答が常に得られるわけではありません。

チェビシェフフィルタは、理想的なフィルタとユーザが必要とするフィルタ応答との間の最大絶対誤差（パスバンド内の最大許容誤差）を考慮することで、パスバンドにおけるピーク誤差を最小限に抑えます。チェビシェフフィルタの周波数応答の特徴は、パスバンド内での振幅応答のリプルが均一であり、ストップバンドでの振幅応答が単調に減少し、バタワースフィルタよりもロールオフがシャープなことです。

次のグラフは、ローパスチェビシェフフィルタの応答を示します。パスバンド内の均一リプル応答は最大許容リプル誤差により抑えられていること、およびストップバンド内にシャープなロールオフが現れていることに注目してください。バタワースフィルタに比べてチェビシェフフィルタが優れている点は、チェビシェフフィルタの方が、次数の低いフィルタでのパスバンドとストップバンドとの間の遷移領域がシャープであることです。これにより、絶対誤差が小さく、実行スピードが速くなります。



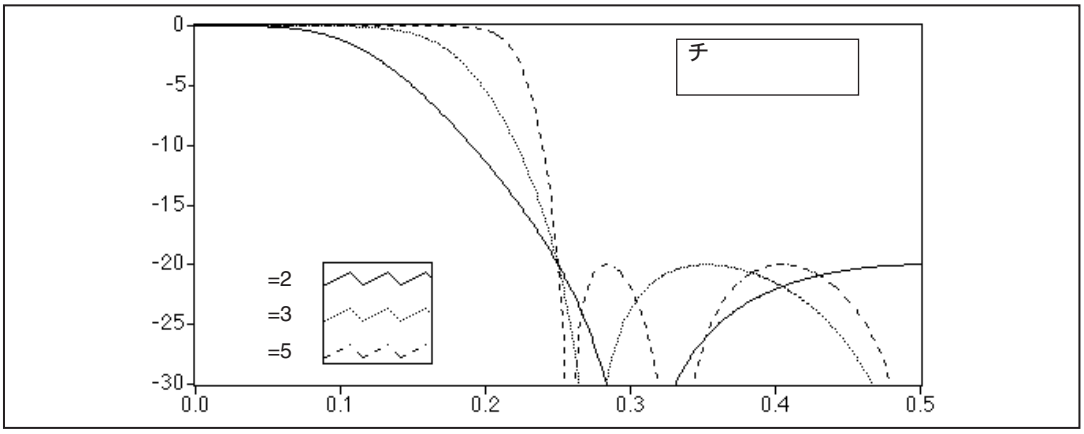
チェビシェフII（逆チェビシェフ）フィルタ

チェビシェフIIフィルタは逆チェビシェフフィルタまたはタイプIIチェビシェフフィルタとも呼ばれ、チェビシェフフィルタに似ていますが、チェビシェフIIフィルタでは（パスバンドとは逆に）ストップバンド上に誤差が分布します。また、チェビシェフIIフィルタは（ストップバンドとは逆に）パスバンドでの均一性が最も高くなっています。

チェビシェフIIフィルタは、理想的なフィルタとユーザが必要とするフィルタ応答との最大絶対誤差を考慮することで、ストップバンドにおけるピーク誤差を最小限に抑えます。チェビシェフIIフィルタの周波数応答の特徴は、ストップバンド内での振幅応答のリプルが均一であり、パスバンドでの振幅応答が単調に減少し、バタワースフィルタよりもロールオフがシャープなことです。

次のグラフは、ローパスチェビシェフフィルタの応答をプロットしたものです。ストップバンド内の均一リプル応答は最大許容リプル誤差により抑えられていること、およびストップバンド内になめらかなロールオフが現れていることに注目してください。バタワースフィルタに比べてチェビシェフIIフィルタが優れている点は、チェビシェフIIフィルタの方が、次数の低いフィルタでのパスバンドとストップバンドとの間の遷移領域がシャープであることです。このような違いにより、絶対誤差が小さく、実

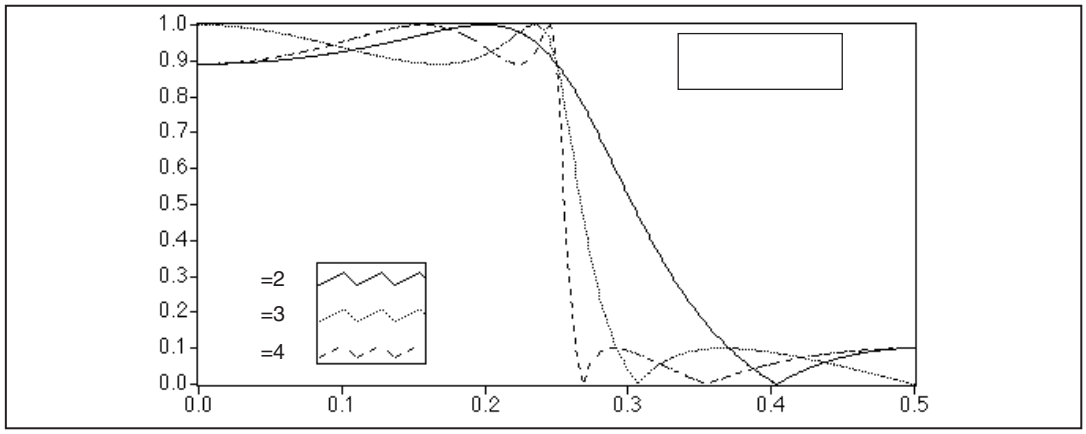
行スピードが速くなります。チェビシェフフィルタに比べてチェビシェフ II フィルタが優れている点の一つは、チェビシェフ II フィルタでは誤差がパスバンドではなくストップバンド上に分布していることです。



エリプティック（カウアー）フィルタ

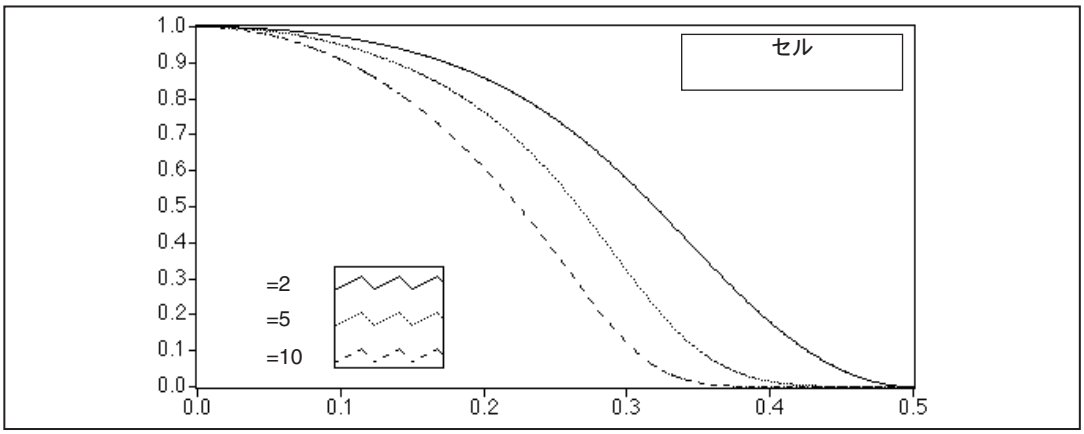
エリプティックフィルタは、パスバンドとストップバンド上に誤差を分布させることによりピーク誤差を最小限に抑えます。エリプティックフィルタの振幅応答の特徴は、パスバンドとストップバンド内のリプルが均一なことです。同じ次数のバタワースフィルタやチェビシェフフィルタと比べ、エリプティック方式ではパスバンドとストップバンドとの間の遷移領域が最もシャープになります。このため、エリプティックフィルタは広く使用されています。

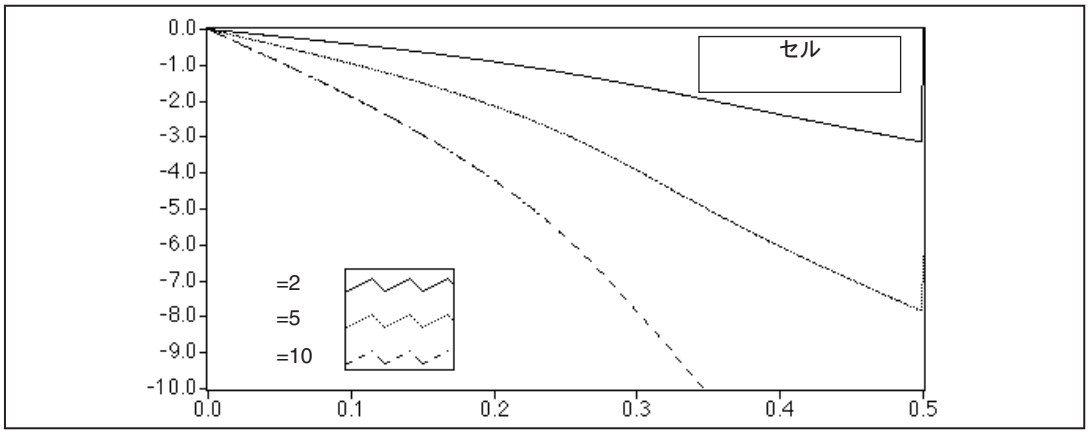
次のグラフは、ローパスエリプティックフィルタの応答をプロットしたものです。パスバンドとストップバンド内のリプルが、同じ最大許容誤差（dB 単位のリプル量で指定される）により抑えられていることに注目してください。また、次数の低いエリプティックフィルタでは遷移エッジがさらにシャープになっていることに注目してください。



ベッセルフィルタ

ベッセルフィルタを使用すると、すべての IIR フィルタに特有な非線形位相歪みを軽減することができます。より次数が高いフィルタやよりロールオフの勾配が急なフィルタでは、特にフィルタ遷移領域において、この条件がいっそう明確になります。ベッセルフィルタも、振幅と位相の両方で応答が最も平坦になります。さらに、重要な領域であるパスバンドにおけるベッセルフィルタの位相応答はほとんど線形です。バタワースフィルタと同様、ベッセルフィルタも、誤差を最小限に抑えるには次数の高いフィルタが必要であるため、あまり広くは使用されていません。また、FIR フィルタ方式を使用して線形な位相応答を実現することもできます。次のグラフは、ローパスベッセルフィルタの応答をプロットしたものです。すべての周波数の応答が滑らかで、振幅と位相の両方で単調に減少していくことに注意してください。また、パスバンド内で位相がほとんど線形であることにも注目してください。





有限インパルス応答フィルタ

有限インパルス応答（FIR）フィルタは、インパルス応答が有限であるデジタルフィルタです。FIR フィルタは、非再帰フィルタ、あるいは移動平均（MA）フィルタとも呼ばれます。これは、FIR フィルタの出力を次のような有限回旋として表現できるからです。

$$y_i = \sum_{k=0}^{n-1} h_k x_{i-k}$$

ただし、*x*はフィルタ処理される入力数列を、*y*は出力フィルタ処理される数列を、*h*はFIR フィルタ係数を示します。

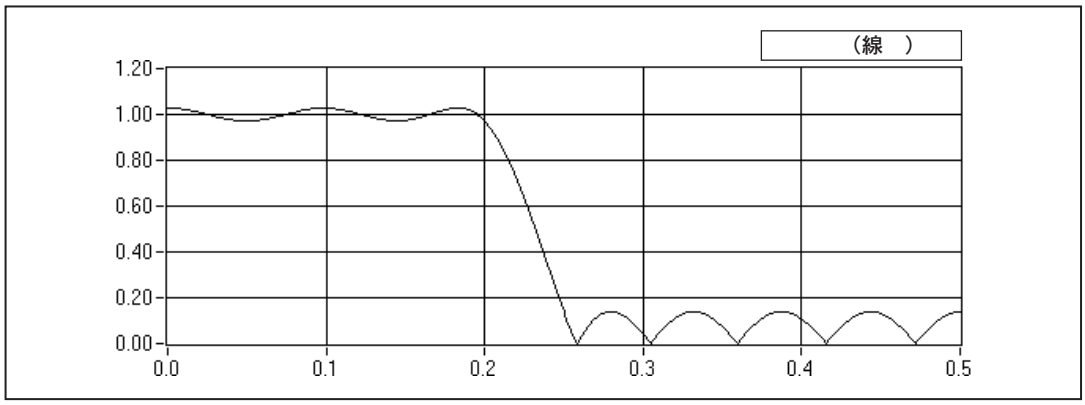
FIR フィルタの最も重要な特徴を以下に示します。

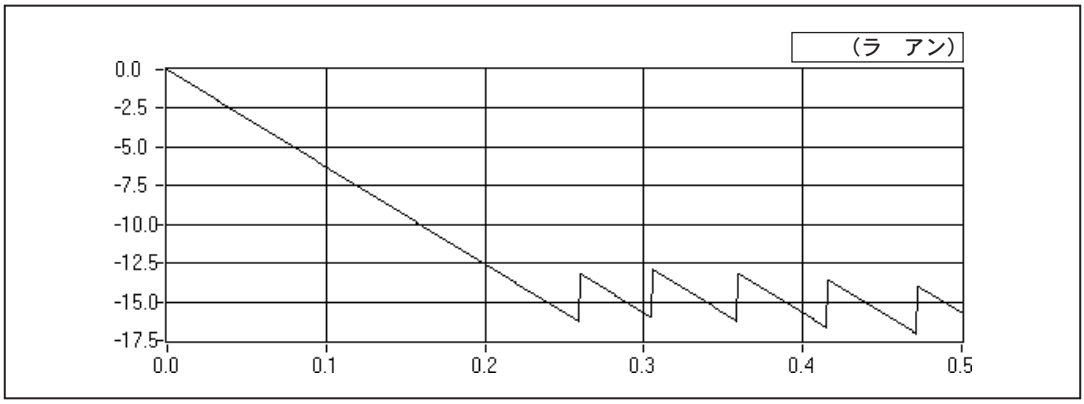
- FIR フィルタは実際のフィルタ係数が対称であるため、線形な位相を実現することができます。
- FIR フィルタは常に安定しています。
- 回旋を使用してフィルタ処理を実行できますので、一般に遅れ (delay) と出力数列は次のように対応付けられます。

$$\text{delay} = \frac{n-1}{2}$$

ただし*n*はFIR フィルタ係数の数です。

次のグラフは、正規化周波数に対するFIR フィルタの典型的な振幅と位相の応答をプロットしたものです。





位相応答内の不連続部は、絶対値を使用して振幅応答を計算する際に生じる不連続性が原因で発生します。位相の不連続部はπのオーダーであることに注意してください。ただし、位相は明らかに線形です。これに関する詳しい説明は、「付録 A　解析に関する参考文献」を参照してください。

FIR フィルタを設計するには、指定された望ましい離散時間系の周波数応答に近付けます。位相応答の線形性を保ちながら、希望する振幅応答に近付けるのが、最も一般的な方法です。

ウィンドウ処理によりFIRフィルタを設計する

線形位相FIR フィルタを設計する最も簡単な方法は、**ウィンドウ設計**法です。ウィンドウ処理によりFIR フィルタを設計するには、理想的な周波数応答から始め、そのインパルス応答を計算し、次にインパルス応答をカットして有限数の係数を求めます。位相を線形にするという制約を満たすには、係数群の中心点について対称になるようにします。理想的なインパルス応答をカットするとGibbs 現象という効果が現れ、FIR フィルタの周波数応答の中に鋭い遷移に近い振動現象（カットオフ周波数）が起こります。

スムージングウィンドウ関数を使用して理想的インパルス応答のカットを滑らかにすることで、Gibbs 現象の影響を軽減することができます。両端のFIR 係数に勾配を付けることで、周波数応答における横の突出部の高さを低くすることができます。ただしこの方法には、メインの突出部の幅が広がってカットオフ周波数での遷移領域が広くなるという欠点があります。この場合のウィンドウ関数の選択は、カットオフ周波数に近い横の突出部のレベルと遷移領域の幅との間で妥協しなければならないという点で、チェビシェフ IIR フィルタとバタワース IIR フィルタのいずれかを選択する場合と似ています。

ウィンドウ処理によるFIR フィルタの設計方法は簡単で、しかも計算が少なくて済みます。したがってこの方法ではFIR フィルタを最も短期間に設

計できます。ただし、この方法が必ずしも最高のFIRフィルタ設計方法であるとは言えません。

パークスマクレランアルゴリズムを使用して最適なFIRフィルタを設計する

パークスマクレランアルゴリズムは、指定された数の係数に対して可能な範囲で最適なフィルタを設計するための、最適なFIRフィルタ設計技術を提供します。このような設計を行うことで、カットオフ周波数における悪影響を軽減することができます。またこのアルゴリズムは、さまざまな周波数帯域における近似誤差をより良好に制御できます。ウィンドウ法ではこのような制御を行えません。

パークスマクレランアルゴリズムを使用してFIRフィルタを設計する方法は、計算は多くなりますが、時間のかかる反復的な手法を用いることにより最適なFIRフィルタを作成できます。

狭帯域FIRフィルタを設計する

従来の技術で特に帯域幅が狭いFIRフィルタを設計する場合、結果として得られるフィルタの長さが非常に長くなることがあります。FIRフィルタが長くなると、設計や実装に要する期間が長くなり、数値的な精度も低下する恐れがあります。場合によっては、パークスマクレランアルゴリズムのような従来のフィルタ設計技術では、設計が完全な失敗となる恐れがあります。

狭帯域FIRフィルタの設計には、補間有限インパルス応答（Interpolated Finite Impulse Response: IFIR）フィルタ設計技術と呼ばれる非常に効率的なアルゴリズムを使用することができます。この技術を使用すると、パークスマクレランアルゴリズムを直接適用して設計したフィルタよりもはるかに少ない数の係数しか必要としない（したがって計算量も少ない）狭帯域フィルタを作成することができます。また、LabVIEWでは、広帯域の（カットオフ周波数がナイキストに近い）ローパスフィルタや（カットオフ周波数が0に近い）ハイパスフィルタの作成にもこの技術を使用します。IFIRフィルタの設計に関する詳細は、P.P.Vaidyanathan著の『Multirate Systems and Filter Banks』や、本書の「付録A　解析に関する参考文献」の一覧に掲載されている、Neuvoらによる補間有限インパルス応答に関する論文を参照してください。

ウィンドウ処理されたFIRフィルタ

ローパス、ハイパス、バンドパス、バンドストップなどウィンドウ処理されたFIRフィルタのタイプを選択するには、FIR VIの**フィルタタイプ**パラメータを使用します。関連する2つのFIR VIを次に示します。

- FIR Windowed Coefficients ― ウィンドウ処理した（またはウィンドウ処理されていない）係数を生成します。
- FIR Windowed Filter ― ウィンドウ処理した（またはウィンドウ処理されていない）係数を使用して入力をフィルタ処理します。

最適なFIRフィルタ

パークスマクレランアルゴリズムを使用すると、与えられた数の係数に対するフィルタ仕様に最適なフィルタが結果として得られるような、最適な線形位相特性のFIRフィルタ係数を設計することができます。Parks-McClellan VIはバンド記述の配列を入力として受け取ります。このバンド記述には、特定のバンドに求められる応答を指定した情報が含まれています。このVIはFIR係数と計算されたリプルを出力します。リプルは、結果として得られるフィルタと理想的なフィルタ仕様の誤差の程度を示します。

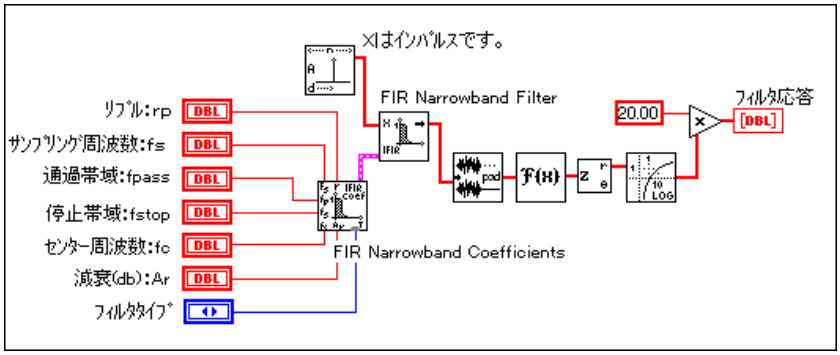
Equi-Ripple LowPass、Equi-Ripple HighPass、Equi-Ripple BandPass、およびEqui-Ripple BandStopの4つのVIは、Parks-McClellan VIを使用して、ストップバンドとパスバンドのリプルレベルが等しいフィルタを実行します。

FIR狭帯域フィルタ

狭帯域FIRフィルタを設計するにはFIR Narrowband Coefficients VIを使用し、さらにFIR Narrowband Filter VIを使用してフィルタ処理機能を実行します。多くの狭帯域フィルタでは長い設計期間を必要とするのに対し、実際のフィルタ処理は非常に高速で効率的であるため、設計と実行は別の作業です。狭帯域フィルタ処理のダイアグラムを作成するときは、このことを忘れないでください。

狭帯域フィルタの仕様に必要なパラメータとしては、フィルタのタイプ、サンプリングレート、パスバンド周波数とストップバンド周波数、パスバンドリプル（線形スケール）、およびストップバンドの減衰量（デシベル）です。バンドパスフィルタとバンドストップフィルタの場合、パスバンド周波数とストップバンド周波数は帯域幅を示しますので、ユーザはこの他に中心周波数パラメータを指定する必要があります。また、狭帯域フィルタVIを使用して、（カットオフ周波数がナイキストに近い）広帯域ローパスフィルタや（カットオフ周波数が0に近い）広帯域ハイパスフィルタを設計することもできます。

次の図は、FIR Narrowband Coefficients VI とFIR Narrowband Filter VI を使用して狭帯域フィルタのインパルスに対する応答を求める方法を示します。



非線形フィルタ

スムージングウィンドウ、IIR フィルタ、FIR フィルタは、次のような重合と比例の原則を満たしますので線形です。

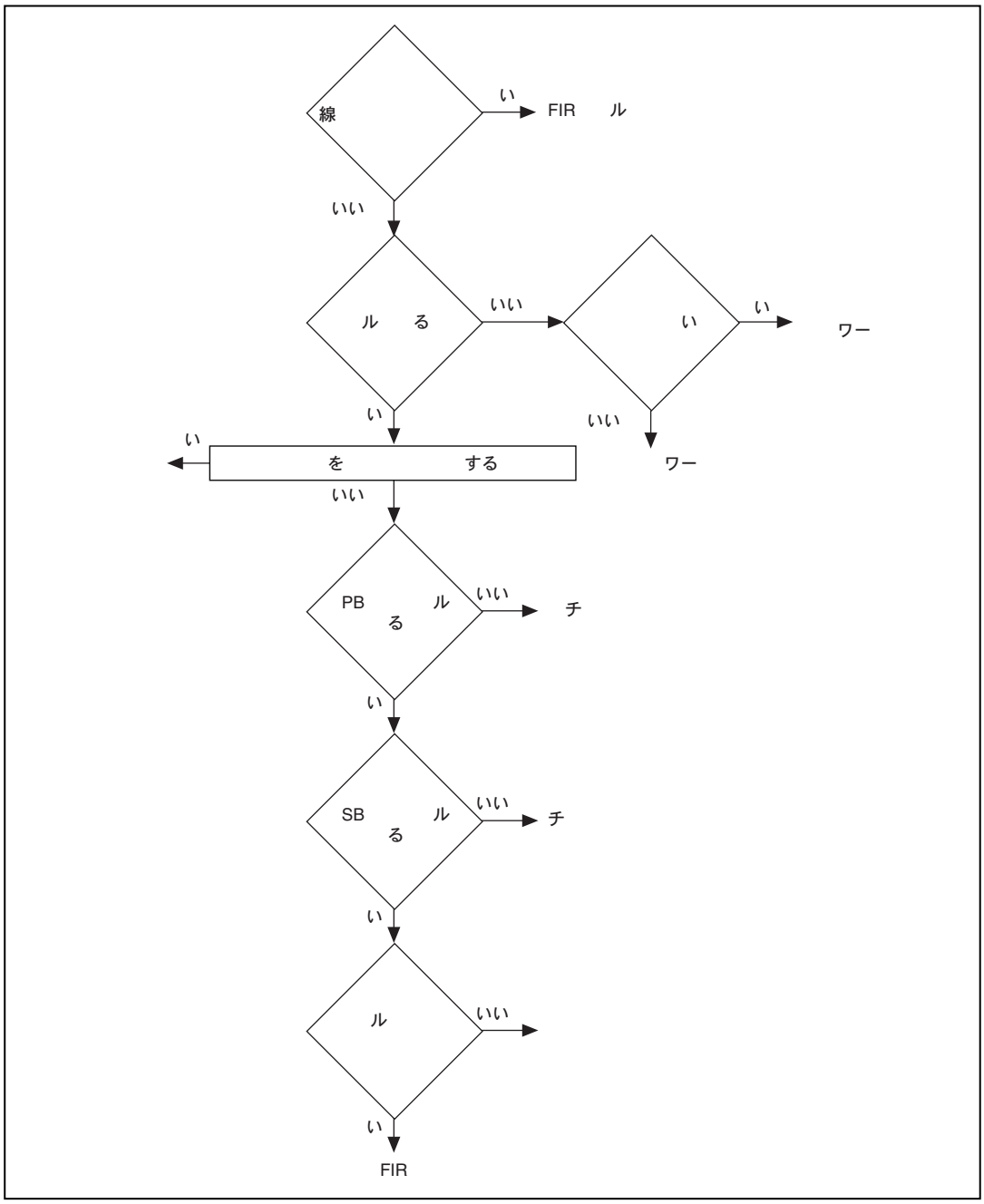
$$L\{ax(t) + by(t)\} = aL\{x(t)\} + bL\{y(t)\}$$

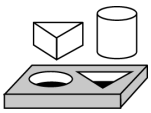
ただし、b は定数、$x(t)$ と $y(t)$ は信号、L{•} は線形フィルタ処理動作であり、これらの入力と出力は回旋操作によって関連付けられます。

非線形フィルタは前述の条件を満たさないため、回旋操作によって出力信号を得ることはできません。これは、一連の係数がフィルタのインパルス応答の特性を決めることができないからです。非線形フィルタでは、線形的な手法では得ることが難しい特殊なフィルタ特性が得られます。中央値フィルタは、ローパスフィルタ特性（高周波数ノイズを除去するため）と高周波数特性（エッジを検出するため）を組み合わせた非線形フィルタです。

使用するフィルタの決定方法

フィルタの種類とその特徴がわかりましたので、次に問題となるのは、各アプリケーションに対してどのフィルタ設計が最適かということです。一般に、適切なフィルタの選択に関わる要素としては、線形の位相特性が必要か、リプルが許されるか、遷移部の帯域幅が狭いことが要求されるか、などがあります。適切なフィルタを選択するためのガイドラインとして、次のフローチャートが役立つはずです。ただし、最終的に適切なものを見つけるためには、さまざまなオプションを実際にいくつか試してみる必要がある、ということを忘れないでください。


線
い
FIR　ル
いい
ル　る
いい
い
い
ワー
い
いい
ワー
い
を　する
いい
PB　ル
る
いい
チ
い
SB　ル
る
いい
チ
い
ル
いい
い
FIR

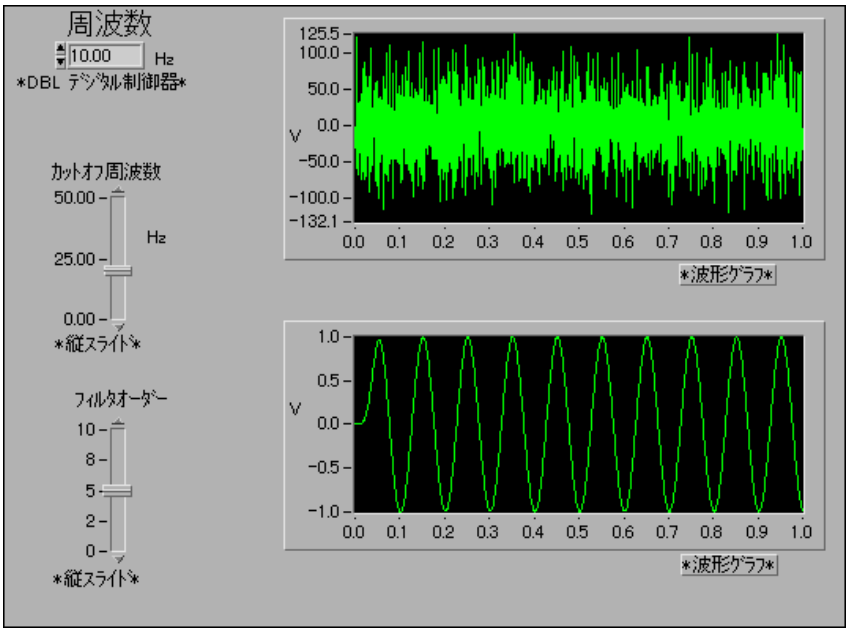


作業 16-1.　正弦波を抽出する

ここでは、高周波ノイズと正弦波信号が含まれているデータサンプルをフィルタ処理することが目的です。

この作業では、Sine Pattern VI で生成された正弦波を高周波ノイズと組み合わせます。(高周波ノイズは、バタワースフィルタを使用して均一なホワイトノイズをハイパスフィルタ処理することにより得られます。) 次に、組み合わせた信号を、別のバタワースフィルタでローパスフィルタ処理して正弦波を抽出します。

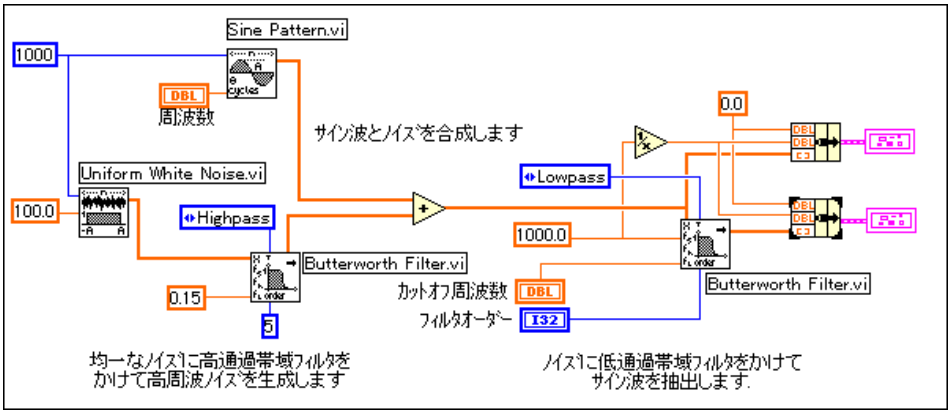
フロントパネル



1. 新しいVIを開き、上図のようなフロントパネルを作成します。
   a. **数値→制御器**パレットからデジタル制御器を選択して**周波数**というラベルを付けます。
   b. **数値→制御器**パレットから垂直スライドを選択して**カットオフ周波数**というラベルを付けます。
   c. **数値→制御器**パレットから垂直スライドをもう一つ選択して**フィルタオーダー**というラベルを付けます。
   d. **数値→グラフ**パレットから波形グラフを選択してノイズ信号を表示し、波形グラフをもう一つ選択して元の信号を表示します。

ブロックダイアグラム

2. 次のようなブロックダイアグラムを作成します。



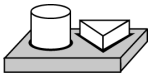
Sine Pattern VI（**関数→解析→信号生成**パレット）は、希望する周波数の正弦波を生成します。

Uniform White Noise VI（**関数→解析→信号生成**パレット）は、正弦波信号に加算される均一なホワイトノイズを生成します。

Butterworth Filter VI（**関数→解析→フィルタ**パレット）は、ノイズに対してハイパスフィルタ処理を行います。

ここでは 10 サイクルの正弦波を生成しようとしており、1000 個のサンプルがあることに注意してください。また、右側の Butterworth Filter VI に対するサンプリング周波数は 1000 Hz に指定されます。このようにして 10 Hz の信号を効果的に生成します。

3. この VI を `Extract the Sine Wave.vi` という名前で `LabVIEW¥Activity` ディレクトリに保存します。
4. フロントパネルに戻ります。**周波数**を 10 Hz、**カットオフ周波数**を 25 Hz、**フィルタオーダー**を 5 と選択し、VI を実行します。
5. **フィルタオーダー**を 4、3、2 と小さくしていき、フィルタ処理された信号の変化を観察します。フィルタの次数を下げていくとどうなるか、説明してください。
6. 作業が終わったら、この VI を、`Extract Sine Wave.vi` という名前で `Dig.filt.llb` ライブラリに保存します。
7. VI を閉じます。



これで作業 16-1 は完了です。

まとめ

周波数応答の特性から、実際のフィルタは理想的フィルタとは異なることがわかりました。実際のフィルタでは、パスバンドのゲインは必ずしも 1 になるわけではなく、ストップバンドの減衰も必ずしも無限大になるわけではありません。また、有限の幅を持つ遷移領域が存在します。遷移領域の幅はフィルタの次数によって決まり、次数が大きくなるほど幅は小さくなります。

また、FIR および IIR デジタルフィルタについても学習しました。FIR フィルタの出力は現在と過去の入力値だけによって決まるのに対し、IIR フィルタの出力は現在と過去の入力値だけでなく過去の出力値によっても決まります。さまざまな設計による IIR フィルタの周波数応答を確認し、パスバンドやストップバンド内でのリプルの有無によってこれらを分類しました。過去の出力によって出力が異なることから、VI を呼び出すたびに IIR フィルタの出力には過渡現象が現れます。この VI を最初に呼び出した後この過渡現象をなくすには、この VI の**初期化／継続**制御器の値を TRUE に設定します。

17

カーブフィット

この章では、データセットから情報を抽出してカーブフィットを行う方法を説明します。回帰 VI の使用方法については、examples¥analysis¥regressn.llb の例を参照してください。

カーブフィットの概要

カーブフィット解析は、データセットから一連の曲線引数や係数を抽出して関数表現を得る手法です。特定のデータセットに合う曲線を求めるアルゴリズムは最小二乗法と呼ばれ、ほとんどの確立統計の入門書で説明されています。誤差は、次のように定義されます。

$$e(a) = [f(x,a) - y(x)]^2 \qquad (17\text{-}1)$$

ただし、$e(a)$ は誤差、$y(x)$ は観測されたデータセット、$f(x,a)$ はこのデータセットの関数表現、および a は曲線を最適近似する一連の曲線係数です。

たとえば、$a = \{a_0,a_1\}$ とすると、直線の関数表現は次のようになります。

$$f(x,a) = a_0 + a_1 x$$

最小二乗アルゴリズムでは、次の式を解くことによって a を求めます。

$$\frac{\partial}{\partial a} e(a) = 0 \qquad (17\text{-}2)$$

この式を解くためには、17-2 の式を展開することによって得られるヤコビアンシステムを設定して解きます。式を a について解くと、関数表現 $f(x,a)$ を使用して、観測されたデータセットの任意の x 値における予測値を求めることができます。

LabVIEW のカーブフィット VI は、自動的にヤコビアンシステムを設定して解き、データセットを最適記述できる一連の係数を返します。ユーザは、17-2 の式を解くことにわずらわされずにデータを関数表現することに集中できます。

2つの入力数列Y値とX値は、データセット$y(x)$を示します。データセットに含まれるサンプルまたは点は、次のようになります。

$$(x_i, y_i),$$

ただし、x_iは数列X値のi番目の要素、y_iは数列Y値のi番目の要素です。

一般的に、特に指定のない限り、あらかじめ定義された各カーブフィットタイプごとに、2つのタイプのVIがあります。一つのタイプは係数だけを返しますので、ユーザはさらにデータを処理することができます。もう一つのタイプは、係数、対応する予測曲線または近似曲線、および二乗平均誤差（MSE）を返します。これは離散的な式であるため、VIは次の公式を使用して、予測曲線値と実際の観測値との間の残差の相対測定値であるMSEを計算します。

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=0}^{n-1}(f_i - y_i)^2 \qquad (17\text{-}3)$$

ただし、fは近似値を表す数列、yは観測値を表す数列、nは観測されたサンプル点の数です。

解析ライブラリには、線形および非線形のカーブフィットアルゴリズムが用意されています。LabVIEWに含まれているさまざまなカーブフィットの概要を以下に示します。

- **線形フィット** — 実験データをy = mx+cの形の直線で近似します。

$$y[i]=a_0+a_1*x[i]$$

- **指数関数フィット** — データをy = *aexp(bx)*の形の指数曲線で近似します。

$$y[i] = a_0*\exp(a_1*x[i])$$

- **一般多項式フィット** — データを次の形の多項式関数で近似します。

$$y = a + bx + cx\text{^}2 + \ldots$$
$$y[i] = a_0+a_1*x[i]+a_2*x[i]^2\ldots$$

- **一般線形フィット** — データを次の関数で近似します。

$$y[i] = a_0+a_1*f_1(x[i])+a_2*f_2(x[i]) + \ldots$$

ただし、y[i] はパラメータ a_0,a_1,a_2... の線形結合です。また、一般線形フィットでは精度と正確さを高めるためにアルゴリズムを選択することもできます。たとえば、$y = a_0+a_1*\sin(x)$ は、y とパラメータ a_0、a_1 の間に線形な関係があるため、線形フィットです。同じ理由から、多項式フィットもすべて線形フィットです。ただし、多項式フィットには、処理を高速化し精度を向上させるための特別なアルゴリズムを設計することができます。

- **非線形 Levenberg-Marquardt フィット** ― データを次の関数で近似します。

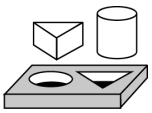
$$y[i] = f(x[i], a_0, a_1, a_2...)$$

ただし、a_0,a_1,a_2... はパラメータです。これは最も一般的な方法であり、a_0,a_1,a_2... との間に線形な関係が存在する必要はありません。この方法は線形または非線形のカーブフィットに使用できますが、ほとんどの場合は非線形カーブフィットに使用されます。これは、線形カーブフィットには一般線形フィット法の方が適しているからです。Levenberg-Marquardt 法では必ず正しい結果が得られるという保証はありませんので、必ず結果を検証する必要があります。

カーブフィットの適用

カーブフィットの実際の適用は多岐にわたります。いくつかの用途を以下に示します。

- 測定時のノイズを除去する。
- 欠けているデータ点を補足する（たとえば 1 つまたは複数の測定値が欠けている場合や正しく記録されていない場合など）。
- 補間（データ点の間のデータを計算する。たとえば測定値間の時間間隔が長すぎる場合）。
- 補外（データ点の範囲外のデータを計算する。たとえば測定が行われた前後のデータ値が欲しい場合）。
- デジタルデータの微分（たとえば、データ点の導関数を求める必要がある場合。離散的なデータを多項式によりモデル化し、結果として得られる多項式を微分することができる。）
- デジタルデータの積分（たとえば、曲線に離散的な点だけしかないときに、その曲線より下の部分の面積を求める場合）。
- 速度（1 階導関数）または加速度（2 階導関数）の離散的な測定値から、物体の軌跡を求める。

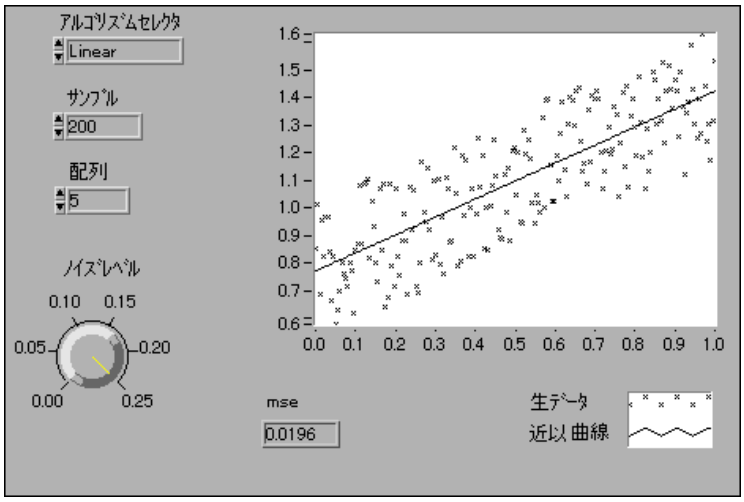


作業 17-1.　カーブフィットVIを使用する

ここでは、Liner Curve Fit VI、Exponential Curve Fit VI、Polynominal Curve Fit VI を比較すること、およびこれらを使用して一連のデータ点を最適近似する一連の最小二乗係数を求めることが目的です。

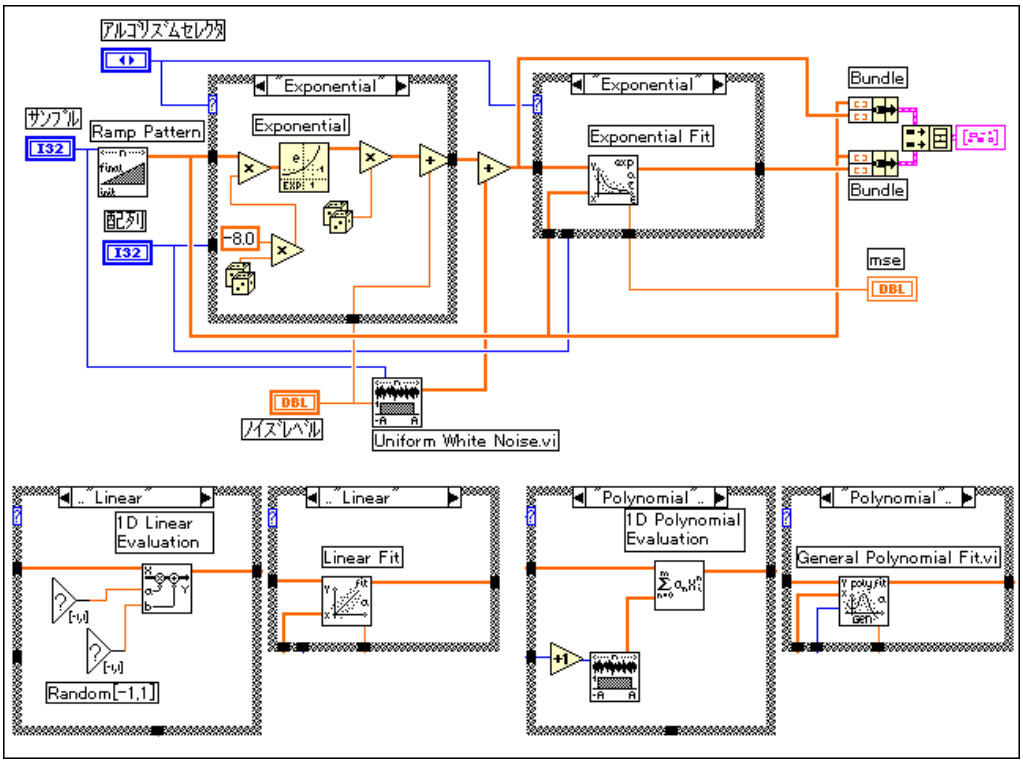
フロントパネル

1. `examples¥analysis¥regressn.llb`のRegressions Demo VIを開きます。フロントパネルとブロックダイアグラムはすでに作成されています。



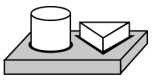
このVIは、線形、指数関数、または多項式に近い、「ノイズの多い」データサンプルを発生します。次にこのVIは対応する解析カーブフィットVIを使用して、これらのデータ点を最適近似する曲線のパラメータを求めます。(この段階では、ノイズの多いデータサンプルがどう発生したのかは考える必要ありません。) フロントパネルの**ノイズレベル**制御器を使用すると、ノイズの振幅を調整できます。

ブロックダイアグラム



2. **アルゴリズムセレクタ**制御器で**Linear**を選択し、**ノイズレベル**を0.1に近い値に設定して、VIを実行します。データ点の広がりと近似された曲線（直線）を確認してください。
3. **階級**と**ノイズレベル**の値を変えてみます。何か気がついたことはありますか？　またmseはどの程度変化しましたか？
4. **アルゴリズムセレクタ**を**Exponential**に変更してVIを実行します。**階級**と**ノイズレベル**の値を変えて実験します。何か気がついたことはありますか？
5. **アルゴリズムセレクタ**を**Polynomial**に変更してVIを実行します。**階級**と**ノイズレベル**の値を変えて実験します。何か気がついたことはありますか？
6. 特に、**アルゴリズムセレクタ**制御器を **Polynomial** に設定した状態で、**階級**を0に設定してVIを実行します。次に階級を1に設定してVIを実行します。観察結果を説明してください。

7. ステップ2、3、4、5の観察結果から、**階級**による制御が最も効果的なのはどのアルゴリズム(Liner,Exponential,Polynominal)の場合ですか？　また、それはなぜですか？
8. VIを閉じて終了します。変更内容は保存しません。



これで作業 17-1 は完了です。

一般LS線形近似理論

一般LS線形近似の問題は、次のように説明できます。

一連の観察データが与えられた場合、線形モデルに最適な一連の係数を求めます。

$$y_i = b_o x_{i0} + \ldots + b_{k-1} x_{ik-1}$$

$$= \sum_{j=0}^{k-1} b_j x_{ij} \quad \text{i=0, 1, ... , n–1} \tag{17-4}$$

ただし、Bは一連の**係数**、nは**Y値**に含まれる要素の数およびHの行数、kは**係数**の数です。

x_{ii}はユーザの観察データであり、Hの中に含まれています。

$$H = \begin{bmatrix} x_{00} & x_{01} \cdots & x_{0k-1} \\ x_{10} & x_{11} \cdots & x_{1k-1} \\ . & & \\ . & & \\ . & & \\ . & & \\ x_{n-10} & x_{n-12} \cdots & x_{n-1k-1} \end{bmatrix}$$

式17-4は、$Y = HB$と表すこともできます。

これは、複数の変数$x_{i0}, x_{i1}, \ldots, x_{ik-1}$を使用して1つの変数$y_i$を予測する複数線形回帰モデルです。これに対し、線形フィット、指数関数フィット、多項式フィットVIはいずれも1つの予測変数を元にしており、1つの変数を使用して別の変数を予測します。

ほとんどの場合は、係数よりも多くの観察データが存在します。17-4の式には解が存在しない場合があります。近似という問題は、最終的には観測データy_iと次の予測値との間の誤差が最も小さくなる係数Bを求める問題ということになります。

$$z_i = \sum_{j=0}^{k-1} b_j x_{ij}$$

このVIは最小カイ二乗平面法を使用して17-4の係数を求めます。すなわち、次の量が最も小さくなる解Bを求めます。

$$\chi^2 = \sum_{i=0}^{n-1} \left(\frac{y_i - z_i}{\sigma_i} \right)^2 = \sum_{i=0}^{n-1} \left(\frac{y_i - \sum_{j=0} b_j x_{ij}}{\sigma_i} \right) = |H_0 B - Y_0|^2 \qquad (17\text{-}5)$$

ただし

$$h_{oij} = \frac{x_{ij}}{\sigma_i}, \; y_{oi} = \frac{y_i}{\sigma_i}, \; i=0, 1, \ldots, n-1; \; j=0, 1, \ldots, k-1$$

この式でσ_iは**標準偏差**です。測定誤差が独立しており一定の標準偏差$\sigma_i = \sigma$で正規分布している場合、前の式も最小二乗計算になります。

χ^2 を最小にするにはさまざまな方法があります。χ^2 を最小にする一つの方法は、χ^2 の $b_0, b_1, \ldots, b_{k-1}$ に関する偏微分が0になるようにすることです。

$$\begin{cases} \dfrac{\partial \chi^2}{\partial b_0} = 0 \\ \dfrac{\partial \chi^2}{\partial b_1} = 0 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \dfrac{\partial \chi^2}{\partial b_{k-1}} = 0 \end{cases}$$

前の式は次のようにすることができます。

$$H_0^T H_0 B = H_0^T Y \tag{17-6}$$

ただし、H_0^T は H_0 の移項です。

17-6 の式も、最小二乗の問題の正規方程式と呼ばれます。LU または Choleskyの因数分解アルゴリズムを使用してこれらを解くこともできますが、正規方程式から得られる解には四捨五入による誤差の恐れがあります。

χ^2 を最小にするための、これに代わる望ましい方法として、次の式の最小二乗解を求める方法があります。

$$H_0 B = Y_0$$

QR またはSVD因数分解を使用すると、解 B を求めることができます。QR因数分解の場合は、Householder、Givens、およびGivens2（高速Givensとも呼ばれます）を選択することができます。

アルゴリズムによって精度が異なる場合があるほか、別のアルゴリズムでは解けると思われる式を、あるアルゴリズムでは解けない場合もあります。さまざまなアルゴリズムを試して、観測データに基づき最適なアルゴリズムを見つけてください。

共分散行列Cは、次のように計算されます。

$$C = (H_0^T H_0)^{-1}$$

最適なフィットZは、次の式で与えられます。

$$z_i = \sum_{j=0}^{k-1} b_j x_{ij}$$

mseは、次の公式を使用して求められます。

$$mse = \frac{1}{n} \sum_{i=0}^{n-1} \left(\frac{y_i - z_i}{\sigma_i} \right)^2$$

1つの予測変数を持つ多項式フィットは、複数回帰の特別な場合とみなすことができます。観測データセットが$\{x_i, y_i\}$（ただし$i = 0, 1, ..., n\text{-}1$）の場合、多項式フィットモデルは次のようになります。

$$y_i = \sum_{j=0}^{k-1} b_j x_i^j = b_0 + b_1 x_i + b_2 x_i^2 + \ldots + b_{k-1} x_i^{k-1} \qquad (17\text{-}7)$$

$$i = 0, 1, 2, \ldots, n-1$$

式17-4と17-7を比較すると、$x_{ij} = x_i^j$ であり、したがって次のようになることがわかります。

$$x_{i0} = x_i^0, \quad x_{i1} = x_i, \quad x_{i2} = x_i^2, \ldots \quad x_{ik-1} = x_i^{k-1}$$
$$= 1$$

この場合は、Hを次のような構成にすることができます。

$$H = \begin{Bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{k-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{k-1} \\ . & & & & \\ . & & & & \\ . & & & & \\ . & & & & \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{k-1} \end{Bmatrix}$$

$x_{ij} = x_j^i$ を使用せずに、データセット$\{x_i,y_i\}$を近似する別の関数公式を選択することもできます。通常は $x_{ij} = f_j(x_i)$ を選択できます。ただしここでは、$f_j(x_i)$ は観測データを近似するために選択する関数モデルです。多項式フィットの場合は $f_j(x_i) = x_i^j$ となります。

一般に、Hは次のような構成にすることができます。

$$H = \begin{Bmatrix} f_0(x_0) & f_1(x_0) & f_2(x_0) & \dots & f_{k-1}(x_0) \\ f_0(x_1) & f_1(x_1) & f_2(x_1) & \dots & f_{k-1}(x_1) \\ . & & & & \\ . & & & & \\ . & & & & \\ . & & & & \\ f_0(x_{n-1}) & f_1(x_{n-1}) & f_2(x_{n-1}) & \dots & f_{k-1}(x_{n-1}) \end{Bmatrix}$$

近似モデルは次のようになります。

$$y_i = b_0 f_0(x) + b_1 f_1(x) + \dots + b_{k-1} f_{k-1}(x)$$

General LS Linear Fit VIの使用方法

Liner Fit VIは、次の式で与えられる直線モデルが実験データ（x[i]とy[i]）を最適近似できるような係数a_0とa_1を計算します。

$$y[i] = a_0 + a_1*x[i]$$

ただし、y[i]は係数a_0とa_1の線形結合です。a1にかける要素が、たとえば、

$$y[i] = a_0 + a_1*\sin(\omega x[i])$$

または、

$$y[i] = a_0 + a_1*x[i]^2$$

または、

$$y[i] = a_0 + a_1*\cos(\omega x[i]^2)$$

などのような*x*の何らかの関数になるように、この概念をさらに拡張することができます。ただしωは角周波数です。上記のいずれの場合もy[i]は係数a_0とa_1の線形結合です。これは、General LS Linear Fit VIの背景にある基本的な考え方であり、y[i]はいくつかの係数の線形結合で、そのそれぞれの要素に対してx[i]の何らの関数がかけ合わされます。したがって、このVIを使用すると、

$$y = a_0 + a_1*\sin(\omega x)$$

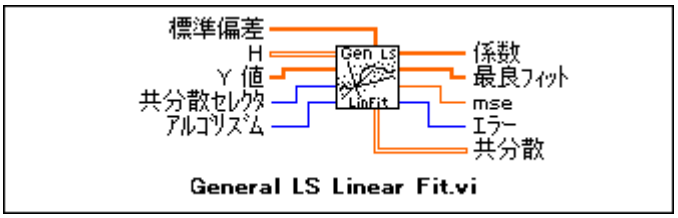
または、

$$y = a_0 + a_1*x^2 + a2*\cos(\omega x^2)$$

$$y = a_0 + a_1*(3\sin(\omega x)) + a_2*x^3 + a_3 / x + ...$$

のような、係数の線形結合として表現可能な関数モデルの係数を計算できます。いずれの場合も*y*は係数の**線形**関数であることに注意してください（*x*については非線形関数の場合もあります）。

次に、General LS Linear Fit VIを使用して、一連のデータ点に対する最適直線フィットを求める方法を説明します。General LS Linear Fit VIの入出力を、次の図に示します。



ユーザが収集するデータ（x[i]とy[i]）は、入力**H**と**Y値**に与えられます。**共分散**出力は係数*ak*の間の共分散行列であり、*cij*は*ai*と*aj*の間の共分散、c_{kk}はa_kの分散です。この段階では、入力**標準偏差**、**共分散セレクタ**、**アルゴリズム**について考える必要はありません。ここでは、これらのデフォルト値をそのまま使用します。これらの入力に関する詳細は、『オンラインリファレンス』を参照してください。

行列**H**を**観測値行列**と呼び、これについては後で詳しく説明します。**Y値**は観測された一連のデータ点y[i]です。たとえばトランスデューサからサンプル（**Y値**）を収集し、次に示すモデルの係数を求める場合を考えてみましょう。

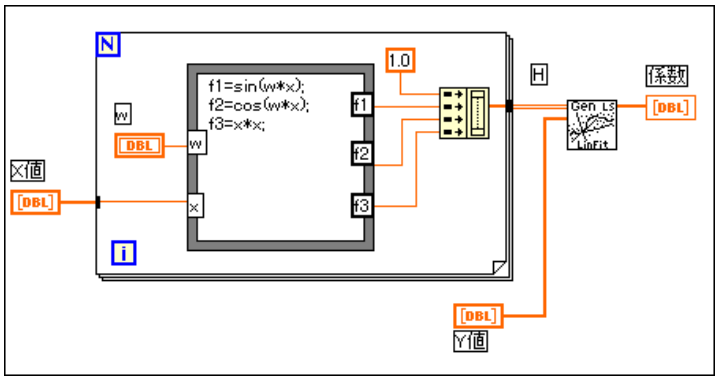
$$y = a_o + a_1 \sin(\omega x) + a_2 \cos(\omega x) + a_3 x^2$$

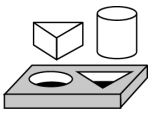
各*aj* $(0 \le j \le 3)$に対する乗数はそれぞれ異なる関数であることがわかります。たとえば、a_0には1、a_1には$\sin(\omega x)$、a_2には$\cos(\omega x)$、...がかけられます。**H**を作成するには、**H**の各列を、各*x*値x[i]で求められた独立した関数に設定します。100個の"*x*"値があると仮定すると、**H**は次のようになります。

$$H = \begin{bmatrix} 1 & \sin(\omega x_0) & \cos(\omega x_0) & x_0^2 \\ 1 & \sin(\omega x_1) & \cos(\omega x_1) & x_1^2 \\ 1 & \sin(\omega x_2) & \cos(\omega x_2) & x_2^2 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & \sin(\omega x_{99}) & \cos(\omega x_{99}) & x_{99}^2 \end{bmatrix}$$

N 個のデータ点と求めるべき *k* 個の係数（a_0、a_1、....a_{k-1}）がある場合、H は *N* 個の行と *k* 個の列からなる *N* 行 *k* 列の行列になります。このように、H の行数は**Y 値**に含まれる要素の数と等しくなり、H の列数は求めようとしている係数の数と等しくなります。

実際に、H は用意されているものではありませんので、作成しなければなりません。*N* 個の独立した**X 値**と観測された**Y 値**がある場合に H を作成して General LS Linear Fit VI を使用する方法を、次のブロックダイアグラムに示します。





作業 17-2.　General LS Linear Fit VI を使用する

ここでは、General LS Linear Fit VI の入力引数をセットアップして VI を使用する方法を学習するのが目的です。

この作業では、General LS Linear Fit VI を使用して一連の最小二乗係数 *a* と近似値を求める方法、および VI に対する入力引数をセットアップする方法を示します。

目的は、一連のデータ点（x[i]、y[i]）を最適近似する一連の最小二乗係数 a を求めることにあります。例として、次の関係を使用してデータを発生させる実際の処理がある場合を考えて見ましょう。

$$y = 2h_0(x) + 3h_1(x) + 4h_2(x) + \text{noise} \qquad (17\text{-}8)$$

ただし

$$h_0(x) = \sin(x^2),$$
$$h_1(x) = \cos(x),$$
$$h_2(x) = \frac{1}{x+1},$$

上記の式で **noise** は乱数値です。また、*x* と *y* の関係の一般形式については大体のことはわかっているが、係数の値が確定できていないものとします。したがって、*x* と *y* には次のような形の関係があると想定できます。

$$y = a_0f_0(x) + a_1f_1(x) + a_2f_2(x) + a_3f_3(x) + a_4f_4(x) \qquad (17\text{-}9)$$

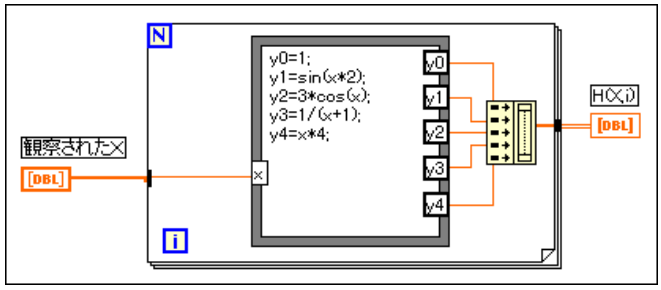
ただし

$$f_o(x) = 1.0,$$
$$f_1(x) = \sin(x^2),$$
$$f_2(x) = 3\cos(x),$$
$$f_3(x) = \frac{1}{x+1},$$
$$f_4(x) = x^4.$$

式17-8と17-9はそれぞれ、実際の物理的な処理と、この処理に対するユーザの推測に対応します。ユーザが推測する際に選択する係数は、実際の値に近い場合もありますが、大きな差がある場合もあります。したがって次には、正確な係数*a*を求めることが目的になります。

観測値行列を作成する

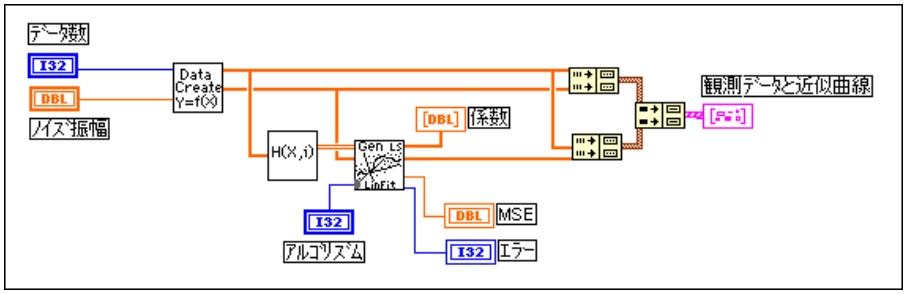
係数*a*を求めるには、配列**H**と**Y値**（行列**H**は2次元配列）内の一連の点（x[i]、y[i]）をGeneral LS Linear Fit VIに対する提供する必要があります。x[i]とy[i]の点は、実験での観測値です。行列**H**を簡単に作成するには、次のブロックダイアグラムに示すフォーミュラノードを使用します。



フォーミュラノードを編集することで、関数を変更、追加、削除できます。この時点では、General LS Linear Fit VIを使用して*a*を求めるために必要な入力はすべて揃っています。式(2)から式(1)を得るには、f0(*x*)に0.0を、$f_1(x)$に2.0を、$f_2(x)$に1.0を、$f_3(x)$に4.0をかける必要があります。したがって、式(1)と(2)から、一連の係数が次のようになることが予測されます。

$$a = \{0.0, 2.0, 1.0, 4.0, 0.0\}$$

次のブロックダイアグラムは、係数と新しい一連の y 値を求めるために General LS Linear Fit VI を設定する方法を示します。



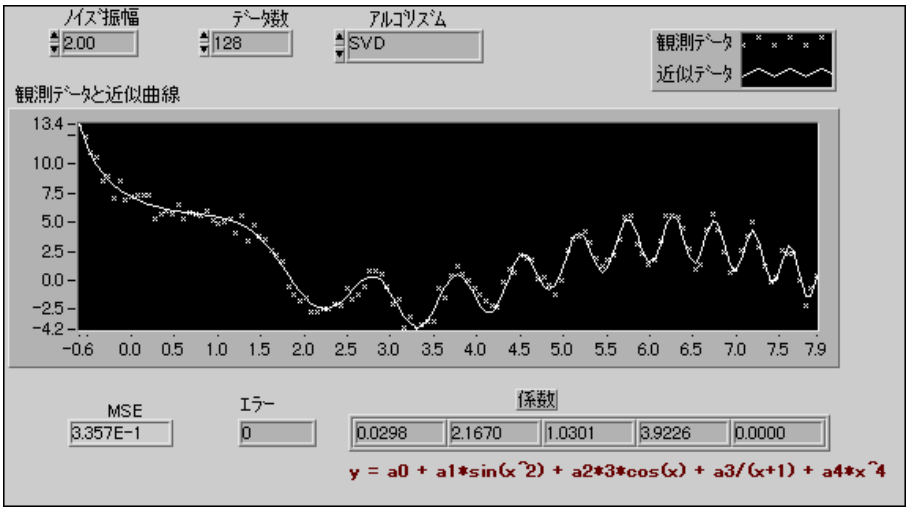
日付を作成というラベルの付いたサブ VI は、配列 X と Y を生成します。このアイコンを、実験中に実際にデータを収集するアイコンと交換することができます。H(X,i) というラベルの付いたアイコンは 2D 行列 H を生成します。

ブロックダイアグラムの最後の部分は、元のデータ点および計算されたデータ点と重ね合わせて General LS Linear Fit の表示用レコードを作成します。X、Y、H の値を使用して General LS Linear Fit VI を実行すると、次の一連の係数が返されます。

したがって、結果として得られる式は次のようになります。

$$y = 0.0298(1) + 2.1670\sin(x^2) + 1.0301(3\cos(x)) + 3.9226/(x+1) + 0.00(x^4)$$
$$= 0.0298 + 2.1670\sin(x^2) + 1.0301(3\cos(x)) + 3.9226/(x+1)$$

結果は次のようなグラフになります。



今度は、この例が組み込まれたVIが表示されます。

1. ライブラリ `examples¥analysis¥regressn.llb` の General LS Fit Example VIを開きます。
2. ブロックダイアグラムをチェックし、よく理解できたか確認してください。
3. フロントパネルをチェックします。

   ノイズ振幅：データ点に加算するノイズの振幅を変更できます。この値が大きいほど、データ点が広がります。

   データ数：発生させるデータ点の数。

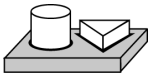
   アルゴリズム:一連の係数と近似値を求めるための6つの異なるアルゴリズムの選択肢を提供します。この例では、どのアルゴリズムでも大きな違いはありません。フロントパネルから異なるアルゴリズムを選択して結果を確認できます。観測するデータセットによっては、アルゴリズムが異なると大きな違いが生ずる場合もあります。

   MSE：二乗平均誤差を与えます。MSEが小さいほど近似結果が良くなります。

   エラー：エラーがあった場合にエラーコードを与えます。エラーコード = 0 の場合はエラーがないことを示します。エラーコードの一覧は、『オンラインリファレンス』を参照してください。

   係数：モデルの係数の計算値（a_0、a_1、a_2、a_3、a_4）です。
4. **ノイズ振幅**の値が徐々に大きくなるようにしてVIを実行します。グラフ上にプロットされる観測データはどうなりますか？　MSEはどうなりますか？

5. **ノイズ振幅**の値を一定に固定したまま、**アルゴリズム**制御器で異なるアルゴリズムを選択してVIを実行します。他のアルゴリズムより良い結果が得られるアルゴリズムがありますか？　MSEが最も小さくなるのは、どのアルゴリズムですか？
6. 作業が終わったらVIを閉じます。変更内容は保存しません。



これで作業 17-2 は完了です。

非線形 Lev-Mar 近似理論

このVIは、カイ二乗の数値量が最小になるような一連の係数を求めます。

$$\chi^2 = \sum_{i=0}^{N-1} \left(\frac{y_i - f(x_i; a_1 \ldots a_M)}{\sigma_i} \right)^2 \qquad (17\text{-}10)$$

この式で、(x_i,y_i)は入力データ点、$f(x_i;a_1...a_M) = f(X, A)$は非線形関数、$a_1...a_M$は係数です。測定誤差の間に依存性がなく、一定の標準偏差$\sigma_i = \sigma$で正規分布している場合は、これも最小二乗推定になります。

Nonlinear Lev-Mar Fit VIのサブVIであるTarget Fnc & Deriv NonLin VIのブロックダイアグラムのフォーミュラノードでは、非線形関数$f = f(X, A)$を指定しなければなりません。Target Fnc & Deriv NonLin VIにアクセスするには、**プロジェクト→このVIのサブVI**を選択し、表示されるメニューでこのVIを選択します。

このVIには、アルゴリズムに必要なヤコビアン（係数についての偏微分係数）を計算する方法が2つ用意されています。この2つの方法を、次に示します。

- 数値的な計算 ― 数値的な近似によりヤコビアンを計算します。
- 公式計算 ― 公式によりヤコビアンを計算します。非線形関数$\partial f/\partial A$だけでなく、Target Fnc & Deriv NonLin VIのブロックダイアグラムのフォーミュラノードのヤコビアン関数$f = f(X, A)$も指定しなければなりません。この計算方法では、ヤコビアンの数値的な近似が不要なため、数値的な計算方法よりも効率的です。

入力配列XとYは、一連の入力データ点を定義します。このVIでは、x座標とy座標の間の関係が非線形であることがあらかじめわかっているものと仮定しています。したがって、一連の係数を*A*としたとき、$f = f(X, A)$はLevenberg-Marquardtアルゴリズムにより求められます。

場合によっては、この関数をうまく使用できるかどうかは、ユーザが最初に予測した係数が解にどれだけ近いかによって決まります。したがって、関数を使用する前に、十分な労力と時間を割いて、使用可能なすべてのリソースから解の係数をうまく予測するようにしてください。

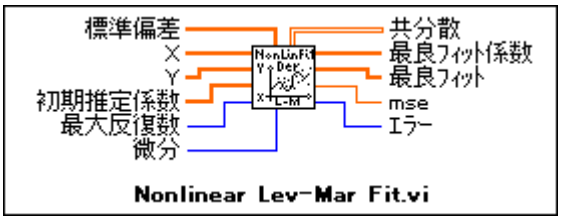
Nonlinear Lev-Mar Fit VIを使用する

ここまで、yと係数$a_0,a_1,a_2,...$の関係が線形である場合に使用されるVI群を説明してきましたが、非線形な関係が存在する場合は、Nonlinear Lev-Mar Fit VIを使用して係数を求めることができます。このVIは、非常に強力なLevenberg-Marquardt法を使用して、Aとy[i]の間の非線形な関係を示す係数A = $\{a_0,a_1,a_2,...,a_k\}$を求めます。このVIでは、x座標とy座標の間に非線形な関係があることがあらかじめわかっているものと仮定しています。

まず準備段階としてNonlinear Lev-Mar Fit VIのサブVIの一つのブロックダイアグラムのフォーミュラノードで非線形関数をあらかじめ指定しておく必要があります。このサブVIは、Target Fnc & Deriv NonLin VIです。Target Fnc & Deriv NonLin VIにアクセスするには、**プロジェクト→このVIのサブVI**を選択し、表示されるメニューでこのVIを選択します。

☞ **注**　**Nonlinear Lev-Mar Fit VIを使用する場合は、Target Fnc & Deriv NonLin VIのブロックダイアグラムのフォーミュラノードで非線形関数も指定しなければなりません。**

Nonlinear Lev-Mar Fit VIへの接続は、次のようになります。



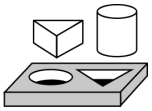
XとYは、入力データ点x[i]とy[i]です。

初期推定係数は最初に予測した係数値です。係数は、Target Fnc & Deriv NonLin VIの**フォーミュラノード**で入力したフォーミュラで使用されているものです。場合によっては、Nonlinear Lev-Mar Fit VIをうまく使用できるかどうかは、ユーザが最初に予測した係数が実際の解にどれだけ近い

かによって決まります。したがって、関数を使用する前に、十分な労力と時間を割いて、使用可能なすべてのリソースから解の係数をうまく予測するようにしてください。

ここでは、その他の入力はデフォルト値のままにしておきます。これらの入力に関する詳細は、『オンラインリファレンス』を参照してください。

最良フィット係数：実験データのモデルを最適近似する係数値（a_0 や a_S など）です。



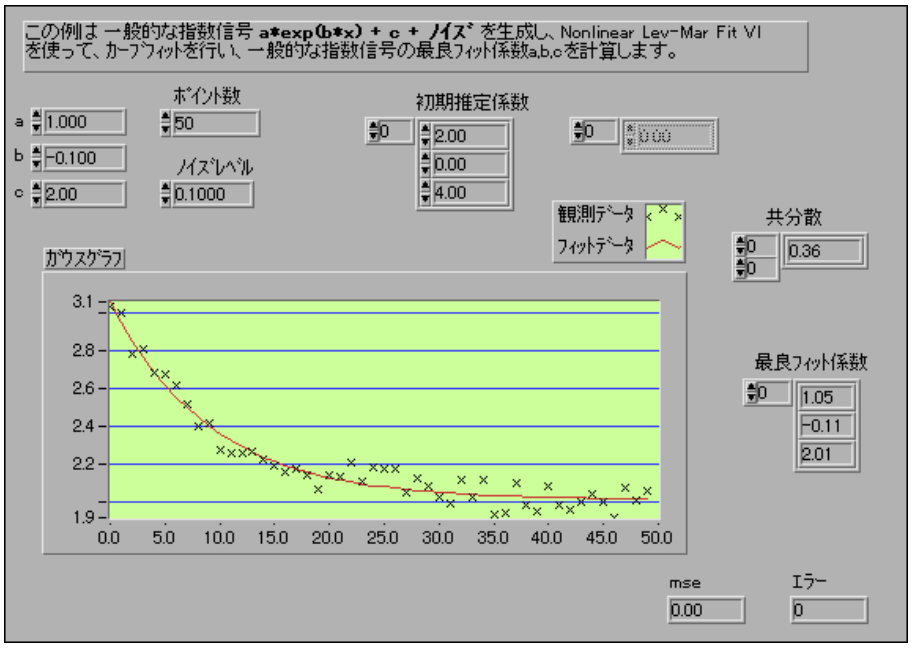
作業 17-3.　Nonlinear Lev-Mar Fit VI を使用する

この作業の目的は、一般的な指数関数信号 a*exp(b*x)+c+noise を作成し、Nonlinear Lev-Mar Fit VI を使用してデータを近似し、一般的な指数関数信号の最適推定係数値 a、b、c を求めることにあります。

この作業では、Nonlinear Lev-Mar Fit VI を使用して、a*exp(b*x)+c で与えられる非線形関数の係数 *a*、*b*、*c* を求める方法を説明します。

フロントパネル

1. ライブラリ `examples¥analysis¥regressn.llb` の Nonlinear Lev-Mar Exponential Fit VI を開きます。フロントパネルを次の図に示します。



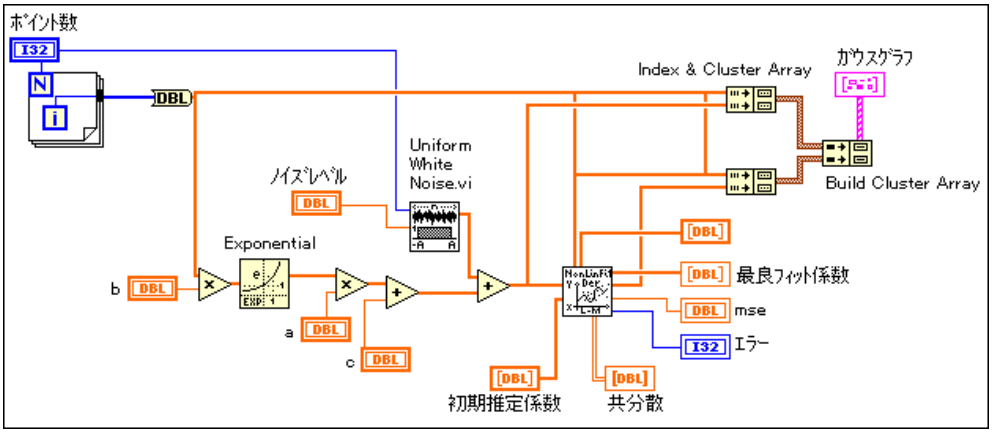
制御器 a、b、c は、係数 *a*、*b*、*c* の実際の値を決定します。**初期推定係数値**制御器は、*a*、*b*、*c* の実際の値に関する学習による推測値です。また、**最良フィット係数**表示器には、Nonlinear Lev-Mar Fit VI により計算された *a*、*b*、*c* の値が表示されます。より実際的な例をシミュレーションするため、この式にノイズも追加して、次のような形にします。

$$a*\exp(b*x) + c + \text{noise}$$

ノイズレベル制御器はノイズレベルを制御します。ただし、選択される *a*、*b*、*c* の実際の値が +1.0、–0.1、2.0 になっていることに注意してください。**初期推定係数**制御器のこれらに対するデフォルトの推測値は、*a* = 2.0、*b* = 0、*c* = 4.0 です。

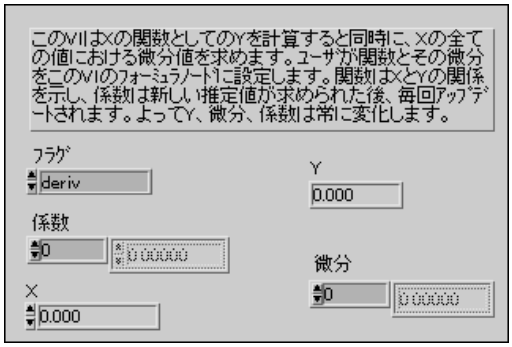
ブロックダイアグラム

2. ブロックダイアグラムをチェックします。

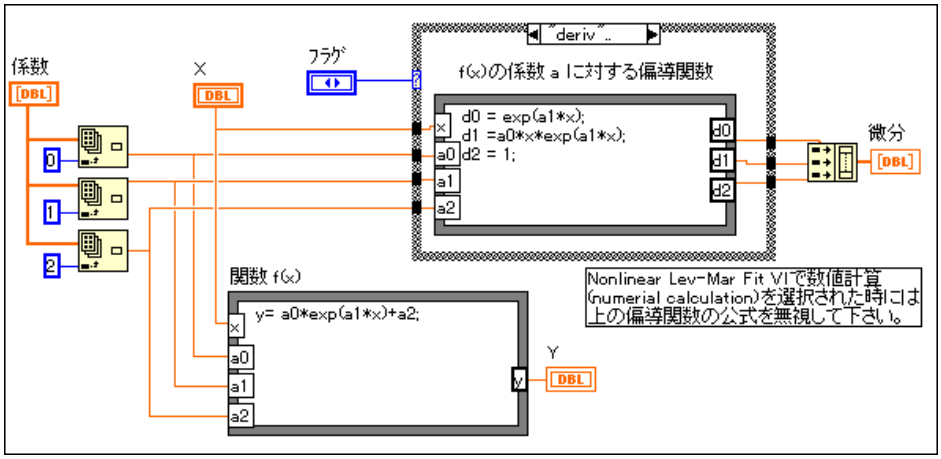


指数関数のデータサンプルを、Exponential VI（**数値→対数関数**サブパレット）を使用してシミュレーションし、Uniform White Noise VI（**解析→信号生成**サブパレット）を使用して均一なホワイトノイズをサンプルに追加します。

3. **プロジェクト**メニューから**開かれていないサブVI→**Target Fnc & Deriv NonLin VI を選択します。次の図のように、Target Fnc & Deriv NonLin VIのフロントパネルが開きます。

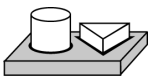


4. ブロックダイアグラムに切り換えます。



下部のフォーミュラノードをよく見てください。フォーミュラノードの形は、計算しようとしているパラメータ（a0、a1、a2）を含む関数の形になっています。

5. Target Fnc & Deriv NonLin VI のフロントパネルとブロックダイアグラムを閉じます。
6. NonLinear Lev-Mar Exponential Fit VI を実行します。**最良フィット係数**の中に返された係数値が、**初期推定係数**制御器に入力された実際の値に非常に近いことに注目してください。また、mse の値も確認してください。
7. **ノイズレベル**を 0.1 から 0.5 に上げてください。mse や**最良フィット係数の係**数値はどうなりますか？
8. **ノイズレベル**を 0.1 に戻し、**初期推定係数**を 5.0、–2.0、10.0 に変更して VI を実行します。**最良フィット係数**表示器と mse 表示器に返される値を確認してください。
9. **ノイズレベル**を 0.1 にしたまま、**初期推定係数**の推測値を 5.0、8.0、10.0 に変更して VI を実行します。今度は、ステップ 4 で選択した値よりも推測値が離れた値になります。誤差を確認してください。この図は、学習により妥当な係数を推測することがどれほど重要であるかを示しています。
10. 作業が終わったら VI を閉じます。変更内容は保存しません。



これで作業 17-3 は完了です。

18

線形代数

この章では、線形代数VI群を使用して行列演算と解析を実行する方法を説明します。線形代数VIの使用方法については、examples¥analysis¥linxmpl.llbの例を参照してください。

連立一次方程式と行列解析

連立一次代数方程式は、信号処理、流体力学計算などの科学計算を使用する多くのアプリケーションに見られます。このような連立方程式は、自然にできることもありますし、微分方程式を代数式で近似した結果できることもあります。

行列のタイプ

どのようなアプリケーションでも必ず、連立方程式の正確な解を非常に効率的な方法で求める必要があります。このような連立一次代数方程式は、行列ベクトル表記では次のような形になります。

$$Ax = b$$

ただしAは$n \times n$の行列、bはn個の要素を含む与えられたベクトル、xは求めるべき未知の解ベクトルです。行列は、要素の2次元配列により表現されます。これらの要素となり得るのは、実数、複素数、関数、または演算子です。次に示す行列Aは、$m \times n$個の要素を含むm行n列の配列です。

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m-1,0} & a_{m-1,1} & \cdots & a_{m-1,n-1} \end{bmatrix}$$

この式でa_{ij}は、第i行第j列に位置する(i,j)番目の要素を示します。一般に、このような行列を**方形行列**と言います。$m = n$の場合、すなわち行の数と列の数が等しい場合の方形行列を**正方行列**と言います。$m \times 1$（m行1列）の行列を**列ベクトル**と言います。**行ベクトル**は、$1 \times n$（1行n列）の行列です。対角要素以外のすべての要素が0（すなわち$i \neq j$のとき$a_{ij} = 0$）であるような行列を**対角行列**と言います。たとえば、

$$A = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 9 \end{bmatrix}$$

は対角行列です。すべての対角要素が 1 に等しい対角行列は**単位行列**と言います。主対角線より下の要素がすべて 0 である行列を**上三角行列**と言い、逆に主対角線より上の要素がすべて 0 である行列を**下三角行列**と言います。すべての要素が実数である行列を**実行列**と言い、逆に複素数の要素が 1 つでもある行列を**複素行列**と言います。わかりやすくするため、ここでは主に実行列について説明しますが、興味のある方のために、複素数行列に関連する作業もいくつか用意しました。

行列式

行列の最も重要な属性のひとつが**行列式**です。次のような最も簡単な 2 × 2 の行列

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

の行列式は *ad–bc* により求められます。正方行列の行列式は、その要素の行列式を使用した形で表すことができます。たとえば、

$$A = \begin{bmatrix} 2 & 5 & 3 \\ 6 & 1 & 7 \\ 1 & 6 & 9 \end{bmatrix}$$

の場合、この行列 A の行列式 |A| は次のようになります。

$$|A| = \left|\begin{bmatrix} 2 & 5 & 3 \\ 6 & 1 & 7 \\ 1 & 6 & 9 \end{bmatrix}\right| = \left(2\left|\begin{bmatrix} 1 & 7 \\ 6 & 9 \end{bmatrix}\right| - 5\left|\begin{bmatrix} 6 & 7 \\ 1 & 9 \end{bmatrix}\right| + 3\left|\begin{bmatrix} 6 & 1 \\ 1 & 6 \end{bmatrix}\right|\right) =$$

$$2(-33) - 5(47) + 3(35) = -196$$

行列式は、その行列についての多くの重要な性質を示しています。たとえば、行列の行列式が 0 である場合、この行列は**特異**であると言います。言い換えると、（行列式が 0 でない）上記の行列は**正則**です。特異性の概念については、後述の「逆行列と連立一次方程式の解法」の項で、逆行列と連立一次方程式の解法を説明する際にもう一度説明します。

行列の転置行列

実行列の**転置行列**は、行と列を入れ替えることにより得られます。行列Bが Aの転置行列A^Tである場合は、$b_{j,i} = a_{i,j}$になります。上記の行列Aの場合、以下のようになります。

$$B = A^T = dd\begin{bmatrix} 2 & 6 & 1 \\ 5 & 1 & 6 \\ 3 & 7 & 9 \end{bmatrix}$$

複素行列の場合の転置行列の複素共役は次のように定義されます。すなわち、行列Dが複素数行列Cの**転置行列の複素共役**（$a = x+iy$のとき共役複素数$a^* = x–iy$）である場合は、次のようになります。

$$D = C^H \Rightarrow d_{i,j} = c^*_{j,i}$$

すなわち、行列Dは、Cの各要素を共役複素数で置き換えてできた行列の行と列を入れ換えることにより得られます。

実行列の転置行列が元の行列と等しい場合、その実数行列は**対称行列**と呼ばれます。この例の行列Aは対称行列ではありません。複素行列Cが関係$C = C^H$を満たすとき、Cを**エルミット行列**と言います。

他のベクトルの線形結合により1つのベクトルが得られるでしょうか？（線形従属）

次に示すようにすべてが0でないスカラ$x_1, x_2, ..., x_n$が存在する場合のみ、1組のベクトル$\alpha_1, \alpha_2, ..., \alpha_n$, は**線形従属**であると言います。

$$\alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n = 0$$

簡単に言うと、他のベクトルの線形結合によりあるベクトルを記述できる場合、この2つのベクトルは線形従属であると言います。

上記の式が成立するα_iの組が$\alpha_1 = 0, \alpha_2 = 0,..., \alpha_n = 0$のみである場合、ベクトルの組$x_1, x_2,..., x_n$は**線形独立**であると言います。したがってこの場合は、いずれのベクトルも他のベクトルの線形結合で記述することはできません。どのベクトルの組が与えられても、$\alpha_1 = 0, \alpha_2 = 0,..., \alpha_n = 0$の場合は常に上記の式が成立します。したがって、ある組の線形独立性を証明するには、上記の式が成立するα_iの組が$\alpha_1 = 0, \alpha_2 = 0,..., \alpha_n = 0$だけであることを証明する必要があります。

例として、まず次のようなベクトルを考えてみます。

$$x = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad y = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

$\alpha_{1x} + \alpha_{2y} = 0$ の関係を満足する値は α_1 と α_2 だけであることに注目してください。したがって、この2つのベクトルは互いに線形独立です。次のようなベクトルの場合はどうでしょうか？

$$x = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad y = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

$\alpha_1 = -2$ かつ $\alpha_2 = 1$ の場合は $\alpha_{1x} + \alpha_{2y} = 0$ であることに注目してください。したがって、この2つのベクトルは互いに線形従属です。次に説明する行列の**階数**という概念を完全に理解するには、このベクトルの線形独立の定義を完全に理解する必要があります。

線形独立性を判断する方法（行列の階数）

行列 A の**階数** $\rho(A)$ は、A に含まれる線形独立な列の最大個数です。例にあげた行列 A を見ると、A のすべての列が互いに線形独立であることがわかります。すなわち、いずれの列も、他の列の線形結合では得られません。したがって、行列の階数は3になります。もう一つ、次の行列 B の例を考えてみます。

$$B = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 2 & 3 \\ 2 & 0 & 2 \end{bmatrix}$$

この行列 B では、3番目の列が最初の2つの列と線形従属であるため、線形独立の列は2つしかありません。したがって、この行列の階数は2となります。また行列の線形独立の列の数が独立の行の数と等しいことを証明することができます。したがって、行列の次数を小さくした場合に行列の階数が大きくなることは絶対にありません。つまり、A が $n \times m$ の行列である場合は、次のようになります。

$$\rho(A) \leq min(n, m)$$

ただし、*min* は 2 つの数の小さい方を示します。行列理論では、正方行列の階数は、その正方行列から形成される最高次数の正則行列の階数になります。前述のように、行列式が 0 である場合その行列は特異行列であることを思い出してください。したがって、階数は、行列式が 0 でない最高次数の行列の階数になります。たとえば4 × 4 の行列を考えてみます。

$$B = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & -1 & 0 \\ 1 & 0 & 1 & 2 \\ 1 & 1 & 0 & 2 \end{bmatrix}$$

この行列の場合は$det(B) = 0$ ですが、

$$\left| \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & -1 \\ 1 & 0 & 1 \end{bmatrix} \right| = -1$$

であるため、*B* の階数は 3 になります。正方行列は、行列式が 0 でない場合に限り、最大の階数になります。行列 *B* は最大階数の行列ではありません。

行列の「大きさ」（ノルム）

連立一次方程式を解く際の誤差と感度を判断するには、ベクトルと行列の「大きさ」についての概念を展開する必要があります。このような連立一次方程式は、たとえば制御システムや流体力学計算などのアプリケーションにおいて得られます。2 次元では、たとえば 2 つのベクトル $\boldsymbol{x} = \begin{bmatrix} x1 & x2 \end{bmatrix}$ と $y = \begin{bmatrix} y1 & y2 \end{bmatrix}$ を比較することができません。これは、$x1 > y1$ だが $x2 < y2$ である場合があるからです。ベクトルノルムは、ベクトルを互いに比較できるように、スカラ量をこれらのベクトルに割り当てる方法です。ノルムは、スカラ数の大きさ、対数の母数、絶対値の概念に似ています。

行列のノルムを計算する方法はいくつかあります。このような方法としては、**2- ノルム**（ユークリッドノルム）、**1- ノルム**、**フロベニウスノルム**（F- ノルム）、**無限ノルム**（inf- ノルム）があります。各ノルムの実際の解釈はそれぞれ異なります。原点を含む単位球を考えてください。ベクトルのユークリッドノルムは、与えられたベクトルが正確に含まれるように円を拡大または縮小するのに必要な単なる倍率です。この様子を、次の図に示します。

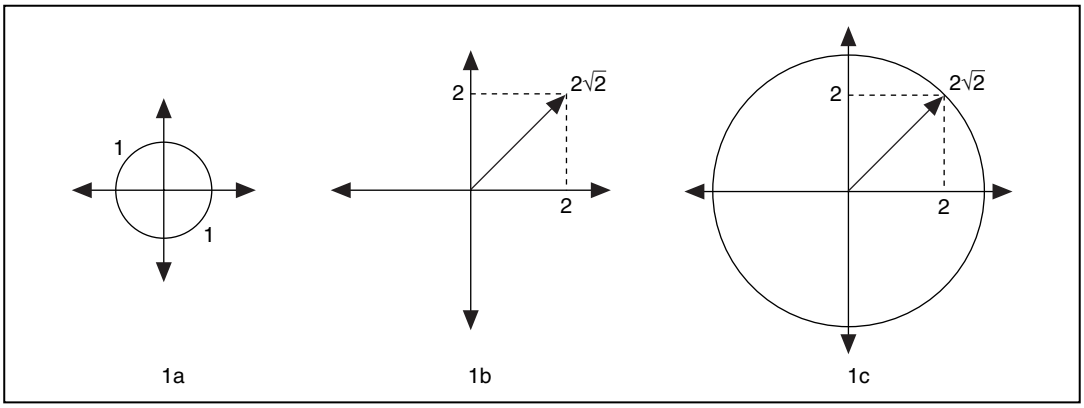


図1aは半径が1単位の単位球を示します。図1bは、長さ$\sqrt{2^2 + 2^2} = \sqrt{8} = 2\sqrt{2}$のベクトルを示します。図1cに示すように、与えられたベクトルが正確に含まれるようにするには、倍率$2\sqrt{2}$で単位球を拡大する必要があります。したがって、このベクトルのユークリッドノルムは$2\sqrt{2}$になります。

行列のノルムは根底にあるベクトルノルムを使用して定義されます。これは、行列がどのベクトルに対しても行う最大相対拡大率です。ベクトル**2-ノルム**を使用した場合は、ノルムに等しい倍率で単位球が拡大されます。一方、行列**2-ノルム**を使用した場合は、いずれかの軸が他の軸よりも長い楕円体（3Dの楕円）になる場合があります。行列のノルムは最も長い軸により決まります。

行列ノルムの中には、他の行列ノルムよりもはるかに容易に計算できるものがあります。**1-ノルム**は、行列の各列に含まれるすべての要素の絶対値の和を求めることにより得られます。これらの和のうち最大のものを1-ノルムと言います。数学用語では、1-ノルムは単に行列の列の和の最大絶対値のことを言います。

$$\|\boldsymbol{A}\|_1 = max_j \sum_{i=0}^{n-1} |a_{i,j}|$$

たとえば

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

の場合は、次のようになります。

$$\|A\|_1 = max(3, 7) = 7$$

行列の**inf-ノルム**は、行列の行の和の最大絶対値です。

$$\|A\|_{\infty} = max_i \sum_{j=0}^{n-1} |a_{i,j}|$$

この場合は、行列の各行のすべての要素の大きさを加算します。こうして得られる最大値を inf- ノルムと言います。上記の例の行列の場合は、次のようになります。

$$\|A\|_{\infty} = max(4, 6) = 6$$

計算が最も難しいのは行列の最大特異値で求められる**2-ノルム**です。特異値については、「行列の因数分解」の項で説明します。

特異性（条件数）を判断する

行列のノルムを使用すると行列の大きさを判断できるのに対し、行列の**条件数**を使用すると行列がどの程度特異に近いかを判断することができます。正則正方行列の条件数は、次のように定義されます。

$$cond(A) = \|A\|_p \cdot \|A^{-1}\|_p$$

ただし p は、前述の 4 種類のノルムのいずれかにすることができます。たとえば、行列 A の条件数を求めるには、A の 2- ノルムと、行列 A の逆行列 A^{-1} の 2- ノルムを求め、両者をかけ合わせます（正方行列 A の逆行列とは、I を単位行列としたときに $AB = I$ となるような正方行列 B のことです）。前述のように、2- ノルムを紙上で計算するのは困難です。LabVIEW 解析ライブラリの Matrix Norm VI を使用すると、次のように 2- ノルムを計算することができます。

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, A^{-1} = \begin{bmatrix} -2 & 1 \\ 1.5 & -0.5 \end{bmatrix}, \|A\|_2 = 5.4650, \|A^{-1}\|_2$$

$$= 2.7325, cond(A) = 14.9331$$

条件数は 1 と無限大の間で変化します。条件数が大きい行列ほど特異であり、条件数が 1 に近い行列ほど特異でなくなります。上記の行列 A は正則ですが、次の行列を考えてください。

$$B = \begin{bmatrix} 1 & 0.99 \\ 1.99 & 2 \end{bmatrix}$$

この行列の条件数は47168ですので、この行列は特異性が大きくなります。覚えていると思いますが、行列式が0に等しい場合、その行列は特異になります。ただし、行列式は行列がどの程度特異に近いかを判断する良い規準とは言えません。上記の行列Bの場合、行列式（0.0299）は0ではありませんが、条件数が大きいということは、行列が特異に近いことを示しています。行列の条件数は常に1以上になることを忘れないでください。また、単位行列と置換行列の場合は行列の条件数が1になります（**置換行列**とは、単位行列のいくつかの行と列を入れ換えたもののことです）。条件数は、連立一次方程式の解の精度を判断する上で非常に役に立つ量です。

この項では、行列に関するいくつかの基本的な表記法と、行列の行列式や階数などの基本概念に慣れていただきました。次の作業は、後の章で頻繁に出てくるこれらの用語の理解を深めるのに役立つはずです。

基本的な行列演算と固有値－固有ベクトルに関する問題

この項では、非常に基本的な行列演算について考察します。行と列の数が同じであり、対応するすべての要素が等しいとき、2つの行列AとBは等しくなります。行列Aとスカラαの積は、すべての要素にそのスカラ値をかけたものに等しくなります。すなわち、

$$\boldsymbol{C} = \alpha\boldsymbol{A} \Rightarrow c_{i,j} = \alpha a_{i,j}$$

です。たとえば、

$$2\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} = \begin{bmatrix}2 & 4\\ 6 & 8\end{bmatrix}$$

2つ（あるいはそれ以上）の行列の加算や減算を行えるのは、両者の行と列の数が同じ場合だけです。行列A、Bの両方がm個の行とn個の列を持つ場合、両者の和Cは$C = A \pm B$と定義される$m \times n$の行列となり、$c_{i,j} = a_{i,j} \pm b_{i,j}$になります。たとえば、

$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} + \begin{bmatrix}2 & 4\\ 5 & 1\end{bmatrix} = \begin{bmatrix}3 & 6\\ 8 & 5\end{bmatrix}$$

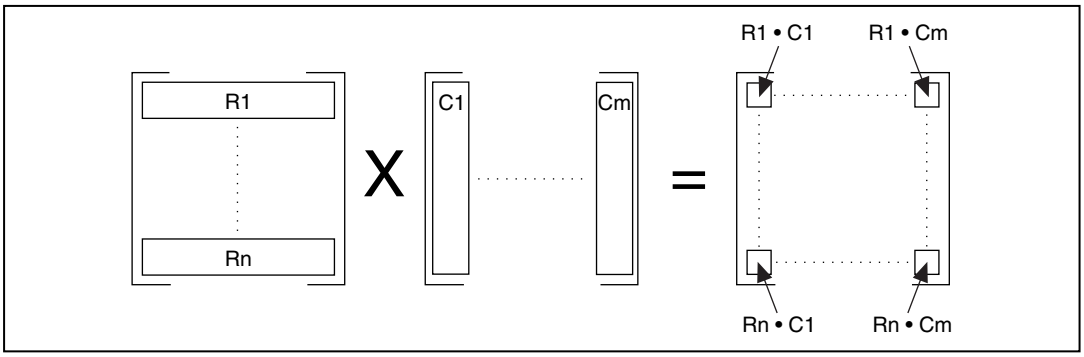
2つの行列の積の場合は、最初の行列の列の数が、2番目の行列の行の数に等しくなければなりません。行列Aがm行n列であり、行列Bがn行p列である場合、両者の積Cは$C = AB$で定義される$m \times p$の行列になり、次のようになります。

$$c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{k,j}$$

たとえば、以下のようになります。

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 2 & 4 \\ 5 & 1 \end{bmatrix} = \begin{bmatrix} 12 & 6 \\ 26 & 16 \end{bmatrix}$$

したがって、*A* の第1行の要素に *B* の第1列の対応する要素をかけ、すべての結果を加算すると *C* の第1行と第1列の要素が得られます。同様に、*C* の第 *i* 行第 *j* 列の要素を計算するには、*A* の第 *i* 行の要素に *C* の第 *j* 列の対応する要素をかけてすべてを加算します。この操作を図にすると、次のようになります。



行列の乗算については一般に交換法則は成り立たず、$AB \neq BA$ になります。また、行列と単位行列の積は元の行列になることも覚えておいてください。

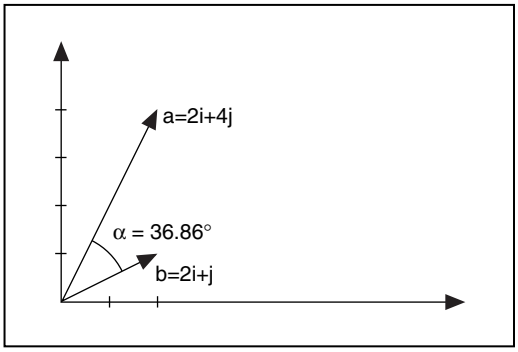
ドット積と外積

X があるベクトルで、*Y* が別のベクトルである場合、各ベクトルの対応する要素をかけて結果を加算すると、この2つのベクトルの**ドット積**が得られます。この操作は次のように表されます。

$$X \bullet Y = \sum_{i=0}^{n-1} x_i y_i$$

ただしnは、*X* と *Y* の要素数です。2つのベクトルの要素数は同じでなければならないことに注意してください。ドット積はスカラ量であり、多くの実用的な用途があります。

たとえば、二次元直交座標系のベクトル $a = 2i + 4j$ と $b = 2i + j$ を考えてください。



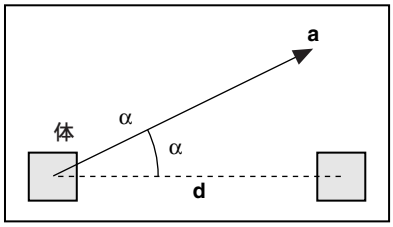
この2つのベクトルのドット積は、次の式で求められます。

$$d = \begin{bmatrix} 2 \\ 4 \end{bmatrix} \bullet \begin{bmatrix} 2 \\ 1 \end{bmatrix} = (2 \times 2) + (4 \times 1) = 8$$

この2つのベクトルの間の角度 α は、次の式で求められます。

$$\alpha = inv\cos\left(\frac{a \bullet b}{|a||b|}\right) = inv\cos\left(\frac{8}{10}\right) = 36.86^{o}$$

ただし|a|は*a* の大きさを示します。



第2の用途として、物体に一定の力*a*が作用する場合を考えてください。*a*が物体を移動させるときの仕事Wは、次のように、|*d*|と*a*の変位*d*の方向の成分との積で定義されます。

$$W = |a||d|\cos\alpha = a \bullet d$$

一方、この2つのベクトルの**外積**は行列になります。この行列の(*i*, *j*)番目の要素は次の公式で得られ、

$$a_{i,j} = x_i \times y_j$$

たとえば次のようになります。

$$\begin{bmatrix}1\\2\end{bmatrix} \times \begin{bmatrix}3\\4\end{bmatrix} = \begin{bmatrix}3 & 4\\6 & 8\end{bmatrix}$$

固有値と固有ベクトル

固有値と固有ベクトルを理解するため、まず古典的な定義を見てみましょう。$n \times n$の行列*A*が与えられたとき、次の式を満足するスカラλと0でないベクトル*x*を求めることが問題です。

$$Ax = \lambda x$$

このようなスカラλを**固有値**、*x*をこれに対応する**固有ベクトル**と言います。

固有値と固有ベクトルの計算は、線形代数学の基本的な法則であり、この法則が示す内容を良く理解していれば、これを使用して連立微分方程式など多くの問題を解くことができます。*x*に*A*をかけたとき回転せず（正確に逆方向を向く場合を除く）0でないベクトルである行列*A*の固有ベクトルを考えてみます。*x*の長さが変わったり方向が逆になることはあっても、方向が横に変わることはありません。言い換えると、上記の式が成り立つ何らかのスカラ定数λが存在します。この値λが*A*の**固有値**です。

次の例を考えてください。次に示す行列A

$$A = \begin{bmatrix} 2 & 3 \\ 3 & 5 \end{bmatrix}$$

の固有ベクトルの一つは、次のようになります。

$$x = \begin{bmatrix} 0.62 \\ 1.00 \end{bmatrix}$$

行列Aとベクトルxをかけるとベクトルxが単に6.85の倍率で拡大されます。したがって、6.85はベクトルxの固有値の一つです。αを任意の定数とすると、次の関係が成り立つため、ベクトルαxも固有値λの固有ベクトルとなります。

$$A(\alpha x) = \alpha Ax = \lambda \alpha x$$

言い換えると、スカラをかけたときに、行列が任意のベクトルを拡大縮小する方向は行列の固有ベクトルにより決まり、拡大縮小の倍率は対応する固有値により与えられます。固有値についての問題を**一般化**すると、次の式を満足するスカラλと0でないベクトルxを求める問題になります。

$$Ax = \lambda Bx$$

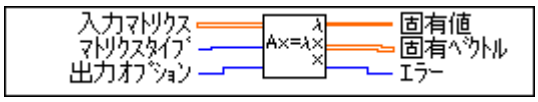
ただしBは別の$n \times n$の行列です。

固有値と固有ベクトルの重要な性質のいくつかを、次に示します。

- 行列の固有値はすべてはっきり決まっているわけではありません。言い換えると、1つの行列に複数の固有値が存在する場合があります。
- 実行列の固有値はすべてが実数である必要はありませんが、実行列の複素数固有値は必ず一対の共役複素数となります。
- 対角行列の固有値はその行列の対角線要素となり、固有ベクトルは同じ次数の単位行列の対応する列となります。
- 実数対称行列の固有値と固有ベクトルは常に実数です。
- 前述のように、固有ベクトルは自由に拡大縮小できます。

固有値の問題は、実際に科学や工学の分野でさまざまな用途に使用されます。たとえば、構造体の安定性、振動の固有モードと周波数は、対応する行列の固有値と固有ベクトルによって決まります。また固有値は、連立代数方程式を解くための反復的な方法の収束性に関する解析、連立微分方程式の解法の安定性に関する解析など、数量的な方法の解析にも非常に役立ちます。

EigenValues and Vectors VIを次に示します。**入力マトリクス**はN×Nの実数正方行列です。**マトリクスタイプ**は入力行列のタイプを決定します。**マトリクスタイプ**の取り得る値は0か1であり、*0*の場合は**一般的な行列**を、1の場合は**対称行列**を示します。対称行列の固有値と固有ベクトルは必ず実数になります。一般的な行列には、対称構造や三角形構造などの特殊な性質がありません。



出力オプションは、計算の必要があるものを規定します。出力オプション = 0 の場合は、計算が必要なのは固有値だけであることを示し、出力オプション = 1 の場合は、固有値と固有ベクトルの両方を計算しなければならないことを示します。固有値と固有ベクトルの両方を計算すると、非常に多くの計算が必要になりますので、EigenValues and Vectors VIで出力オプション制御器を使用する場合は注意が必要です。固有値だけを計算する必要があるか、固有値と固有ベクトルの両方を計算する必要があるかは、個々のユーザアプリケーションにより異なります。また、対称行列の場合は非対称行列より計算が少なくて済みますので、マトリクスタイプ制御器での選択にも注意が必要です。

ここでは、いくつかの基本的な行列演算、および固有値と固有ベクトルについての問題を学習しました。次の例では、解析ライブラリの中にある、このような演算を行ういくつかのVIについて概要を説明します。

逆行列と連立一次方程式の解法

正方行列*A*に対し次のような関係がある正方行列を**逆行列**と言い、A^{-1}で表されます。

$$A^{-1}A = AA^{-1} = I$$

ただし*I*は単位行列です。逆行列が存在するのは、行列の行列式が0でない（正則である）場合のみです。一般には、正方行列の逆行列しか求めることはできませんが、方形行列の**疑似逆行列**は計算することができます。これについては、後述の「行列の因数分解」の項で説明します。

連立一次方程式の解

行列ーベクトル表記では、$n \times n$ の行列を A、与えられた n 次のベクトルを b としたとき、連立一次方程式は $Ax = b$ という形になります。目的は、未知の解となる n 次ベクトル x を求めることにあります。このような解が存在するかどうかに関して問題となる重要な点が 2 つあります。すなわち、そのような解が存在するのか、もし存在するなら、それがただ 1 つの解なのかということです。この 2 つの問に対する答えはいずれも、行列 A が特異か正則かを判断することによって得られます。

前述のように、次に示す性質のいずれかが行列にある場合、その行列は特異であると言います。

- その行列の逆行列が存在しない。
- その行列の行列式が 0 である。
- A の行（または列）が一次従属である。
- $z \neq 0$ であるいくつかのベクトルに対して $Az = 0$ である。

そうでない場合、この行列は正則です。行列が正則である場合はその逆行列 A^{-1} が存在し、b の値にかかわらず連立方程式 $Ax = b$ にはただ 1 つの解 $x = A^{-1}b$ が存在します。逆に、その行列が特異である場合は、右辺のベクトル b によって解の数が決まります。A が特異であり $Ax = b$ である場合は、上記の最後の定義にあるようなベクトルを z としたとき、任意のスカラ Υ に対して $A(x + \Upsilon z) = b$ となります。このように、特異な連立方程式に解が存在する場合は、それがただ 1 つの解ではありません。

逆行列をそのまま計算するのは、値が不正確になりやすいため好しくありません。したがって、A の逆行列に右辺の既知のベクトルをかけて連立一次方程式を解くのは良い方法ではありません。このような連立方程式を解く方法として一般的なのは、元の連立方程式を、元の連立方程式と同じ解を持ち計算が容易な別の連立方程式に変換する方法です。そのような方法の一つにガウス消去法があります。行列演算に関する詳細は、「付録 A　解析に関する参考文献」を参照してください。ガウス消去法に含まれる 3 つの基本ステップを次に示します。まず、行列 A を次のように積の形で表現します。

$$A = LU$$

ただし、L は単位下半三角形行列、U は上半三角行列です。このような因数分解を LU 因数分解と言います。このように表した場合、連立一次方程式 $Ax = b$ は $LUx = b$ と表すことができます。このような連立方程式を解くには、まず下半三角連立方程式形 $Ly = b$ を、**前進代入**により y について解きます。これが、ガウス消去法の第 2 ステップです。たとえば、

$$l = \begin{bmatrix} a & 0 \\ b & c \end{bmatrix} \qquad y = \begin{bmatrix} p \\ q \end{bmatrix} \qquad b = \begin{bmatrix} r \\ s \end{bmatrix}$$

である場合は、次のようになります。

$$p = \frac{r}{a}, q = \frac{(s - bp)}{c}$$

yの第1要素は、行列Lが下半三角行列であるという性質から、容易に決定することができます。次に、この値を使用して未知のベクトルの残りの要素を順番に計算できます。これが、前進代入という名前の理由です。最後のステップでは、**後退代入**により上半三角連立方程式$Ux = y$を解きます。たとえば、

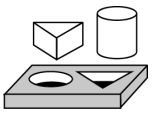
$$U = \begin{bmatrix} a & b \\ 0 & c \end{bmatrix} \qquad x = \begin{bmatrix} m \\ n \end{bmatrix} \qquad y = \begin{bmatrix} p \\ q \end{bmatrix}$$

の場合は、次のようになります。

$$n = \frac{q}{c}, m = \frac{(p - bn)}{a}$$

この場合、x のこの最後の要素は容易に決定でき、これを使用して他の要素を順番に決定できます。これが、後退代入という名前の理由です。この章のここまでの部分では、正方行列の場合について考察してきました。非正方行列は必ずしも特異ではないため、連立方程式の解が存在しない場合や、ただ1つの解でない場合があります。このような場合は通常、近似的な意味で連立一次方程式を満たすただ1つの解xを求めます。

解析ライブラリには、逆行列の計算、行列のLU分解計算、および連立一次方程式を解く計算を行うための VI 群が用意されています。不要な計算をしなくても済むように、適切な入力行列を指定することが重要です。また、こうすることで値の精度も高くすることができます。可能な行列のタイプとしては、一般の行列、正の有限行列、下半三角行列、上半三角行列の4つがあります。実行列が正の有限行列となるのは、これが対称行列であり、0でないすべてのベクトルに対する二次形式がXである場合だけです。入力行列が正方行列であるが最大階数でない（**階数が足りない行列**である）場合、VIは最小二乗解xを求めます。**最小二乗解**は、$Ax - b$のノルムが最小になるような解のことです。非正方行列の場合も同じことが成り立ちます。



作業 18-1.　逆行列を計算する

ここでは、逆行列を計算することが目的です。

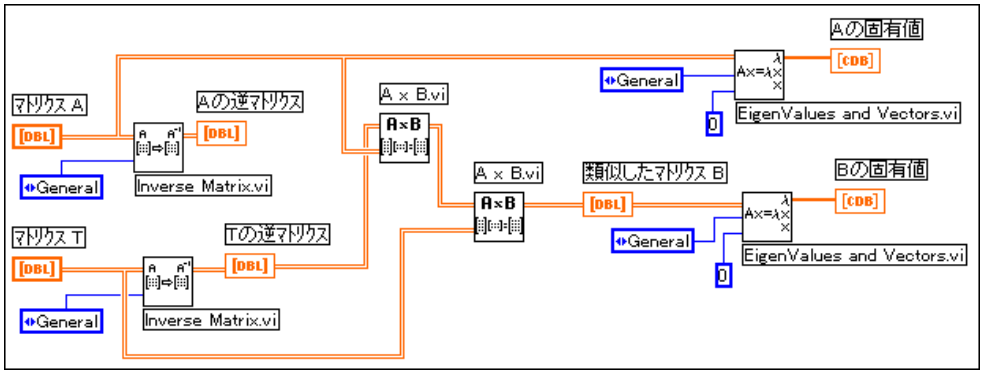
行列*A*の逆行列を計算するVIを作成します。さらに、行列*A*と類似した行列*B*を計算します。$B = T^{-1}AT$となるような正則行列Tが存在し*A*と*B*の固有値が等しい場合、行列*B*と行列*A*は**相似**であると言います。ここでは、相似な行列に関するこの定義を確認します。

フロントパネル

1. 次のようなフロントパネルを作成します。行列*A*は2 × 2 の実行列です。行列*T*は2 × 2 の正則行列であり、類似した行列*B*の作成に使用されます。

ブロックダイアグラム

2. ブロックダイアグラムを開き、次の図のように修正します。

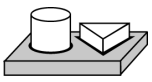


Inverse Matrix 関数（**解析→線形代数**サブパレット）。この作業で、この関数は入力行列*A* の逆行列を計算します。

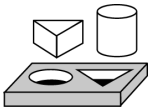
AxB 関数（**解析→線形代数**サブパレット）。この作業で、この関数は 2 つの二次元入力行列の積を計算します。

EigenValues and Vectors 関数（**解析→線形代数**サブパレット）。この作業で、この VI は入力行列の固有値と固有ベクトルを計算します。

3. この VI を、`Mtrix Inverse.vi` という名前で `LabVIEW¥Activity` ディレクトリに保存します。
4. フロントパネルに戻って VI を実行します。*A* の固有値と、相似な行列 *B* の固有値が同じになっているか確認してください。



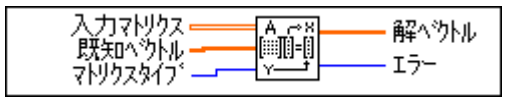
これで作業 18-1 は完了です。



作業 18-2.　連立一次方程式を解く

ここでは、連立一次方程式を解くことが目的です。

実際の応用では多くの場合、連立一次方程式を解くことが必要になります。非常に重要な応用分野として、防衛関連の分野があります。この分野には、大きい標的からの電磁気の散乱や放射の解析、大きいレードームの性能解析、レーダー断面の小さい航空宇宙船の設計（ステルス技術）などが含まれます。第二の応用部分は、携帯電話などの無線通信システムの設計とモデリングです。ほかにも多数の応用分野がありますので、解析ライブラリに含まれる VI を使用して連立一次方程式を解く方法を正しく理解することが非常に重要です。

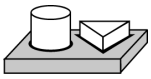


1. **解析→線形代数**サブパレットの中にある`Solve Linear Equations.vi`を使用して、次のような入力行列*A*、既知のベクトル*b*がある場合の連立方程式*Ax* = *b*を解きます。

$$A = \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -1 & 7 \end{bmatrix}, b = \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix}$$

   行列タイプとして一般を選択します。

2. `A x Vector.vi`を使用して行列*A*とベクトル*x*の積（上記の演算の出力）を計算し、結果が上記のベクトル*b*と等しいかチェックします。
3. この VI を、`Linear System.vi`という名前で`LabVIEW¥Activity`ディレクトリに保存します。



これで作業 18-2 は完了です。

行列の因数分解

前の項では、連立一次方程式を、より計算が容易な連立一次方程式に変換する方法を説明しました。この方法では、入力行列を、より単純な複数の行列の乗算に因数分解することが基本的な考え方となっていました。このような手法の一つであるLU分解法を前に説明しましたが、この**LU分解法**では入力行列を上半三角行列と下半三角行列の積に因数分解しました。よく使用される他の因数分解法としては、**Cholesky**、**QR**、**特異値分解**（**Singular Value Decomposition**: SVD）があります。これらの因数分解法を使用すると、連立一次方程式を解く、逆行列を求める、行列の行列式を求めるなど、行列に関する多くの問題を解くことができます。

入力行列Aが対称行列でありかつ正の有限行列である場合は、上半三角行列をUとしたときに$A = U^T U$となるようなLU因数分解を計算することができます。これを**Choleskyの因数分解**と呼びます。この方法では、ガウス消去法による一般行列のLU因数分解に比べて、作業量と記憶容量が約半分で済みます。解析ライブラリの中にあるTest Positive Definite VIを使用すると、行列が正の有限行列であるか容易に判断することができます。

行列Qの列が**正規直交**である場合、この行列は**直交行列**です。すなわち、$Q^T Q = I$の場合、単位行列になります。**QR因数分解法**は、行列を直交行列Qと上半三角行列Rの積に因数分解します。すなわち$A = QR$になります。QR因数分解は、正方行列と方形行列のどちらにも使用できます。QR因数分解には、**Householder変換**、**Givens変換**、**高速Givens変換**など、多くのアルゴリズムを使用できます。

特異値分解（SVD）法は、行列を$A = USV^T$のように3つの行列の積に分解します。UとVは直交行列です。Sは対角行列であり、対角線上の値をAの**特異値**と呼びます。Aの特異値は$A^T A$の負でない平方根であり、UとVの列はそれぞれ左特異ベクトル、右特異ベクトルと呼ばれ、AA^Tと$A^T A$の正規直交固有ベクトルとなります。SVDは、行列の階数、ノルム、条件数、疑似逆行列の計算などの解析上の問題を解くのに役立ちます。次の項では、この最後の応用について説明します。

疑似逆行列

スカラσの疑似逆行列は、1/σの場合は$\sigma \neq 0$、そうでない場合は0と定義されます。スカラの場合、疑似逆行列は逆行列と同じになります。また、対角行列の疑似逆行列は、行列の転置行列を求め各要素のスカラの疑似逆行列を求めることで定義できます。一般の$m \times n$の実数行列Aの逆行列$A^{\dagger}$は、次の式で与求められます。

$$A^{\dagger} = VS^{\dagger}U^{T}$$

行列が正方でも方形でも、逆行列が存在することに注意してください。Aが正方で正則な場合、疑似逆行列は通常の逆行列と同じになります。解析ライブラリには、実行列と複素行列の疑似逆行列を計算するためのVIが含まれています。

まとめ

- 行列は、m個の行とn個の列からなる二次元配列とみなすことができます。行列の重要な属性としては、行列式、階数、条件数などがあります。
- 行列の条件数は、最終的な解の精度に影響を与えます。
- 対角行列、上半三角行列、下半三角行列の行列式は、対角要素の積になります。
- 2つの行列の積を計算できるのは、第1行列の列の数が第2行列の行の数に等しい場合だけです。
- 行列の固有ベクトルとは、行列を適用しても回転しない0でないベクトルのことです。相似な行列の固有値は等しくなります。
- 連立方程式にただ1つの解が存在するかどうかは、その行列が特異であるか正則であるかによって決まります。

19

確率と統計

この章では、確率と統計に関するいくつかの基本概念を説明し、これらの概念を使用して実際の問題を解く方法を示します。確率 VI と統計 VI の使用方法については、examples¥analysis¥statxmpl.llb の中の例を参照してください。

確率と統計

我々は情報時代に生きており、事実や数字が生活の重要な部分を形成しています。「雷雨の確率は 60% である」、「ジョーはクラスの上位 5 番以内に入る」、「マイケル・ジョーダンの今シーズンの 1 試合当たりの平均得点は 30 点である」などの表現が、日常的に使用されています。このような表現は多くの情報を与えてくれますが、このような情報がどのようにして得られるのかについては、ほとんど考えません。この情報を得るためには多くのデータが関連しているのでしょうか？　その場合、どのようにして、**確率60%** とか**平均30点**とか**上位5番以内**などのような1つの数にまとめたのでしょうか？　このようなさまざまな問いに対する答えとして、統計という非常に興味深い分野が出てきます。

まず、情報（データ）がどのように生成されるかを考えてみましょう。バスケットボールの 1997 年のシーズンを考えてみます。シカゴブルズのマイケル・ジョーダンは、51 試合に出場して合計で 1568 点を入れました。この中には、シャーロットホーネッツに対して最後のスリーポイントシュートを決めて 103 対 100 で勝った時の 45 点、ポートランドトレイルブレーザーズに 88 対 84 で勝った時の 36 点、ニューヨークニックスに対して 88 対 87 で勝った時のシーズン最高得点となった 51 点、クリーブランドキャバリエズに 102 対 97 で勝った時の 45 点、7 リバウンド、5 アシスト、3 スチール、ミルウォーキーバックスに 107 対 104 で勝った時の 40 点、6 リバウンド、6 アシストなどが含まれています。ここで言いたいのは、ジョーダンがすばらしいプレーヤーであるということではなく、たった 1 人のプレーヤーでも 1 シーズンでこれだけ多くのデータが発生するということです。問題は、このようなデータをすべてまとめて、特に重要なすべての情報を覚えやすい形で抜き出す方法です。**統計**という言葉が重要になってくるのはこのような場合です。

すべてのデータをまとめるには、1 つの数字でよりわかりやすく有効な内容を推測できるようにする必要があります。たとえば、さまざまな試合でのジョーダンの得点数を考えてみます。各試合での彼の得点を覚えておく

のは大変ですが、合計得点数（1568点）を出場試合数（51回）で割ると、1試合当たりの**平均**得点数30.7という1つの数になります。

次に、ジョーダンのフリースローシュートの技能をランク付けしたい場合を考えてみましょう。各試合で彼のパフォーマンスを見るだけで得点を付けるのは困難ですが、全試合で彼が得点したフリースローの数を、彼が獲得したフリースローの合計回数で割ると、彼のフリースローの**確率**84.4%が得られます。NBAのすべてのプレーヤーについてこの値を計算すれば全員をランク付けできます。このように、すべてのプレーヤーについての情報をまとめて、フリースローの確率、1試合当たりの得点、スリーポイントの平均値などを示す数を得ることができます。この情報を基にすれば、さまざまな部門でプレーヤーをランク付けすることができます。また、これらのさまざまな値の重みを決めて、各プレーヤーごとに1つの数にまとめることもできます。このような数があれば、シーズンの最優秀選手（MVP）を決めるのに役立ちます。このように、一般の概念での統計という言葉は、データをまとめて役立つ重要情報を抽出するためのさまざまな方法を意味します。

次に問題となるのは、確率とは何かということです。大量のデータをまとめて1つの数を抽出する方法はすでに説明しました。これらの値は、現在の状態に関する結論を引き出すのに役立ちます。たとえば、1996年のシーズンにおけるジョーダンの統計データを見ると、彼がこのシーズンのMVPに選ばれたのがうなずけます。では、将来については何かわかるでしょうか？　推測の精度を調べて将来の事に関する判断に使用できないでしょうか？　これに答えるのが確率理論です。一般的には、ジョーダンは今後数年は引き続き最高選手と**なりそうだ**、という言い方をしますが、後述のような確率分野のさまざまな概念を使用すれば、もっと数量的に表現することができます。

話は変わりますが、結果は予測できないがある特定の結果になる可能性が高い、というような場合があります。この場合もやはり確率についての概念に結びつきます。たとえば、偏りのないコインを空中に投げた場合、表が上になる可能性はどのくらいでしょうか？　可能性または確率は、50%です。これは、コインを繰り返し何回も投げた場合に、そのうちの半分は表が上になることを意味します。これは、10回投げたら必ず5回は表が上になったり、100回投げたら必ず50回は表が上になるという意味でしょうか？　おそらくそうはならないでしょうが、長い目で見れば確率は0.5になります。

つまり、統計を使用すると、データをまとめて現在のことに関する結論を引き出すことができるのに対し、確率を使用すると、これらの結論の正確さを調べて将来のことに関して使用できます。

統計

この項では、統計でよく使用されるさまざまな概念と用語を紹介し、さまざまな用途にG Analysis VIを使用する方法を説明します。

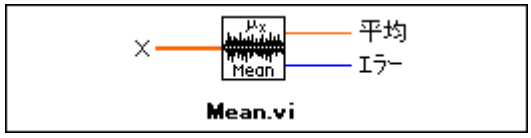
平均値

n個のサンプル $x_0, x_1, x_2, x_3, \ldots, x_{n-1}$ で構成されるデータセットXを考えます。平均値は$\bar{x}$で表され、次の公式で定義されます。

$$\bar{x} = \frac{1}{n}(x_0 + x_1 + x_2 + x_3 + \ldots + x_{n-1})$$

言い換えると、平均値は、すべてのサンプル値の総和をサンプル数で割った値です。マイケル・ジョーダンの例では、データセットは51個のサンプルから構成されています。各サンプルは、ジョーダンの各試合での得点です。これらの得点の合計は1568でありサンプル数（51）で割ることで、平均値30.7が得られました。

Mean VIに対する入出力接続を、次に示します。



中央値（Median）

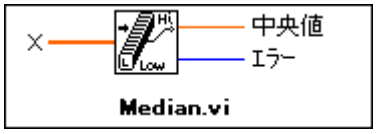
データセットXを並べ換えた数列を$S = \{s_0, s_1, s_2, \ldots, s_{n-1}\}$ とします。数列は、昇順にも降順にも並べ換えることができます。数列の中央値はx_{median}で表され、次の公式で得られます。

$$x_{median} = \begin{cases} s_i & \text{nは奇数} \\ 0.5(s_{k-1} + s_k) & \text{nは偶数} \end{cases} \qquad i = \frac{n-1}{2} \text{ と } k = \frac{n}{2}.$$

要するに、データ列の中央値とは、その数列を並べ換えたものの**中央に位置する**値のことです。たとえば、5つの（奇数の）サンプルからなる数列{5,4, 3, 2, 1}を考えてみます。この数列は既に降順に並べ換えられています。この場合、中央値は中央の値3です。また、4つの（隅数の）サンプルからなる数列{1, 2, 3, 4}を考えてみます。この数列は既に昇順に並べ換えられています。この場合は、中央値が2と3の2つあります。上記の公式から、中央値は$0.5 \times (2 + 3) = 2.5$となります。ある学生Xがテストで4.5点とり、別の学生Yの点数が1点である場合、「Xはクラスの上位半分

に属する」、「Y はクラスの下位半分に属する」というような定性的な表現を行う上で、中央値は非常に役に立ちます。

Median VI に対する入出力接続を、次に示します。



サンプル分散

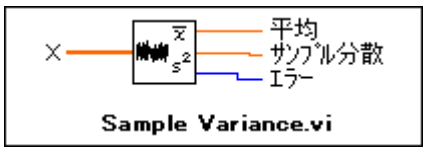
n 個のサンプルから構成されるデータセット X のサンプル分散は s^2 で表され、次の公式で定義されます。

$$s^2 = \frac{1}{n-1}[(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + \ldots + (x_n - \bar{x})^2]$$

ただし $\bar{x}$ はデータセットの平均値を示します。したがって、サンプル分散は、サンプル値の平均値からの二乗和を *n–1* で割った値に等しくなります。

☞ **注**　***n = 1*の場合には上記の公式を適用できませんが、データセット内に１つのサンプルしかない場合は、分散を計算する意味がありません。**

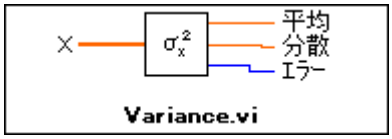
Sample Variance VI に対する入出力接続を、次に示します。



言い換えると、サンプル分散は、サンプル値の広がりまたは分散の程度を示します。データセットがさまざまな試合におけるプレーヤーの得点から構成されている場合、サンプル分散は、プレーヤーの一貫性を示す尺度として使用することができます。すべてのサンプル値が互いに等しく平均値に等しくない限り、サンプル分散は常に正の値になります。

分散にはこの他に、集合分散というタイプがあります。集合分散の計算公式は上記のサンプル分散の計算公式と似ていますが、分母の (*n–1*) が *n* になる点が異なります。

Variance VIに対する入出力接続を、次に示します。



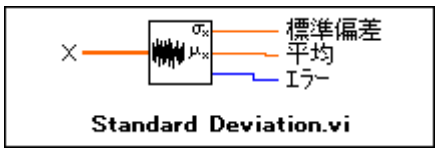
Sample Variance VIはサンプル分散を計算するのに対し、Variance VIは集合分散を計算します。統計学者や数学者は後者を、エンジニアは前者をよく使用します。*n*がたとえば30以上の大きな数になれば、どちらでも大きいな違いはありません。

☞ **注**　　用途に適した正しいタイプのVIを使用してください。

標準偏差

サンプル分散S^2の正の平方根はsで表され、サンプルの標準偏差と呼ばれます。

標準偏差 VIに対する入出力接続を、次に示します。



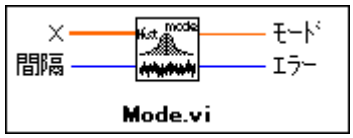
モード

サンプルのモードとは、サンプルの中に最も頻繁に現れる値のことです。たとえば、次のような入力数列*X*がある場合、

$$X = \{0, 1, 3, 3, 4, 4, 4, 5, 5, 7\}$$

*X*の中で最も頻繁に現れるのは4ですので、*X*のモードは4になります。

Mode VIに対する入出力接続を、次に示します。



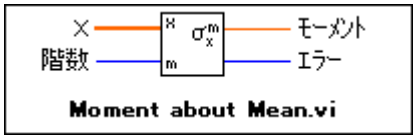
平均値に関するモーメント

n個の要素を含む数列をXとし、この数列の平均値を$\bar{x}$とした場合、m次のモーメントは次の公式により計算されます。

$$\sigma_x^{\,m} = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - \bar{x})^m$$

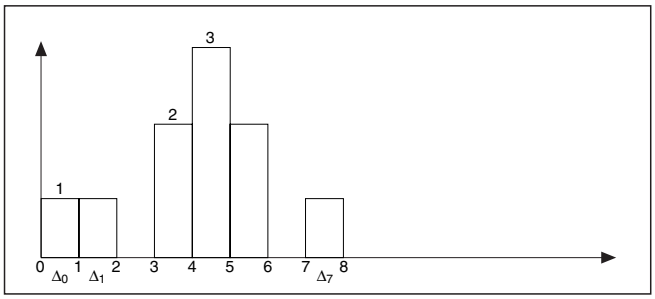
言い換えると、平均値に関するモーメントとは、数列内の要素の、平均値からの偏差の程度の尺度です。$m = 2$の場合の平均値に関するモーメントは集団分散に等しくなることに注意してください。

Moment About Mean VIに対する入出力接続を、次に示します。



ヒストグラム

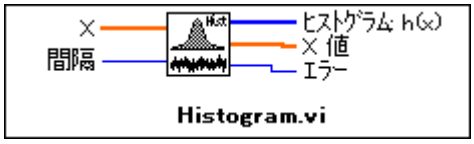
この章のここまでの部分では、データセットの重要な特徴を抽出するさまざまな方法を説明してきました。データは通常、表形式で格納されますが、このままでは多くの人にはよく理解できません。データを視覚的に表示すると、データを洞察することができます。データを図式的に表示して有用な情報を要約するこのような方法の一つが、ヒストグラムです。データ数列$X = \{0, 1, 3, 3, 4, 4, 4, 5, 5, 8\}$を考えてみましょう。値の全範囲を、0～1、1～2、2～3、... 7～8という8つの区間に分けます。数列Xをヒストグラムにすると、この区間に含まれるデータサンプルの数がプロットされます。上限値は区間に含まれません。



前図は、0～1、1～2の範囲には1つずつのデータサンプルがありますが、2～3の範囲にはサンプルがないことを示します。同様に、3～4の範囲に

は2つのサンプルがあり、4～5の範囲には3つのサンプルがあります。上記の数列*X*の例を学習し、この概念を理解してください。

ヒストグラム用にデータを計算するには、さまざまな方法があります。次に、Histogram VIで数列*X*を使用してこの計算を行う方法を示します。



前図のように、このVIに対する入力は、入力数列*X*と間隔*m*です。VIは、次のようにしてHistogram: h(x)を求めます。Xをスキャンして*X*に含まれる値の範囲を判断します。続いてVIは、指定された値*m*に従って次のように間隔幅Δxを設定します。

$$\Delta x = \frac{max - min}{m}$$

ただし*max*は*X*内の最大値、*min*は*X*内の最小値、*m*は指定された間隔です。

たとえば、

$$m = 8$$

である場合、間隔幅は次のようになります。

$$\Delta x = \frac{8 - 0}{8} = 1$$

出力数列Xの値をχとします。ヒストグラムは*X*の関数です。このVIは、次の式を使用してχの要素を求めます。

$$\chi_i = min + 0.5\Delta x + i\Delta x \qquad \text{ただし} \qquad i = 0, 1, 2, \ldots, m - 1$$

この例の場合は次のようになります。

$$\chi_0 = 0.5, \chi_1 = 1.5, \ldots, \chi_7 = 7.5$$

続いてVIは、$\chi_i - 0.5\Delta x$以上$\chi_i + 0.5\Delta x$未満の範囲内に入るように*i*番目の間隔を次のように定義します。

$$\Delta_i = [(\chi_i - 0.5\Delta x),\ (\chi_i + 0.5\Delta x)] \qquad \text{ただし} \qquad i = 0, 1, 2, \ldots, m - 1$$

さらに、xがΔ_iに属する場合は$y_i(x) = 1$、そうでない場合は$y_i(x) = 0$となるような関数yi(x)を定義します。指定した間隔内に*x*がある場合、この関数

の値は1になり、そうでない場合は0になります。ただし、間隔の境界値は間隔に含まれません。間隔はχ_iを中心に位置決めされ、幅がΔ_xになることに注意してください。最大値に等しい値がある場合、その値は最後の間隔に属するものとみなされます。

我々の例では次のようになり、

$$\Delta_0 = [0, 1], \Delta_1 = [1, 2], \ldots, \Delta_7 = [7, 8]$$

また、例と同様、

$$y_0(0) = 1$$

および

$$y_0(1) = y_0(3) = y_0(4) = y_0(5) = y_0(8) = 0$$

となります。

最後に、VIは次の式を使用してヒストグラムの数列Hを求めます。

$$h_i = \sum_{j=0}^{n-1} y_i(x_j) \qquad \text{ただし} \qquad i = 0, 1, 2, \ldots, m-1$$

ただしh_iは出力数列**Histogram:h(X)**の要素を示し、nは入力数列Xに含まれる要素の数を示します。この例の場合は、$h_0 = 1, h_4 = 3, \ldots, h_7 = 1$となります。

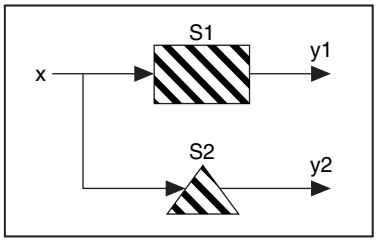
G解析ライブラリには、Histogram VIより高度なGeneral Histogram VIも用意されています。詳しい説明は、『オンラインリファレンス』を参照してください。

二乗平均エラー（MSE）

2つの入力数列をXおよびYとするとき、二乗平均エラーは、2つの入力数列の対応する要素の差を二乗した値の和の平均値です。mse は、次の公式により求められます。

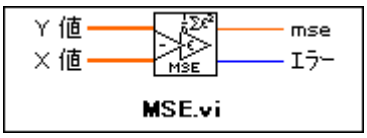
$$mse = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - y_i)^2$$

n はデータ点の数です。



システムS1にデジタル信号xが加えられる場合を考えてみましょう。このシステムの出力をy1とします。ここで、理論的には同じ結果を出力し応答速度が2倍の新しいシステムS2を入手します。旧システムを交換する前に、両方のシステムの出力応答が同じであることを完全に確認する必要があります。数列y1とy2が非常に大きい場合は、数列内の各要素を比較するのは困難です。このような場合は、MSE VIを使用して2つの数列y1とy2の二乗平均エラー（mse）を計算することができます。mse が許容誤差より小さければ、システムS1を安心して新しいシステムS2に交換することができます。

MSE VIに対する入出力接続を、次に示します。



二乗平均平方根（RMS）

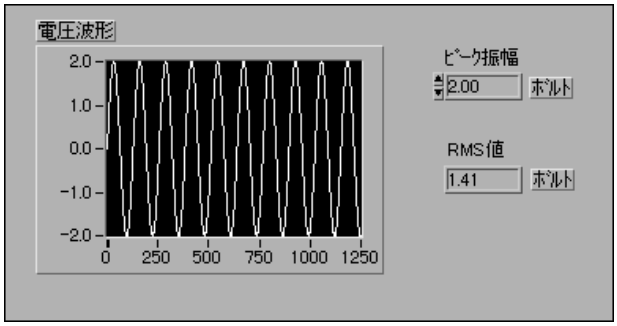
数列 X の二乗平均平方根の値Ψ_xは、入力数列を二乗した値の平均値の正の平方根です。言い換えると、入力数列を二乗し、新しく得られた二乗値の数列の平均値を計算し、その平方根を求めます。rms 値の計算には、次の公式が使用されます。

$$\Psi_x = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1} x_i^2}$$

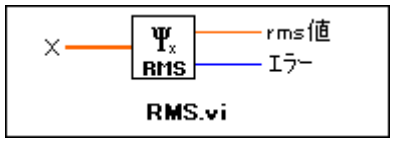
nはXに含まれる要素の数です。

RMS は、アナログ信号の場合に広く使用されている量です。電圧波形が正弦波である場合、信号のピーク振幅を V_p とすると二乗平均電圧 V_{rms} は $\frac{V_p}{\sqrt{2}}$ で得られます。

次の図に、ピーク振幅 = 2V の平方根電圧波形と、解析ライブラリを使用して計算された RMS 値 $\sqrt{2} \approx 1.41$ を示します。



RMS VI に対する入出力接続を、次に示します。



確率

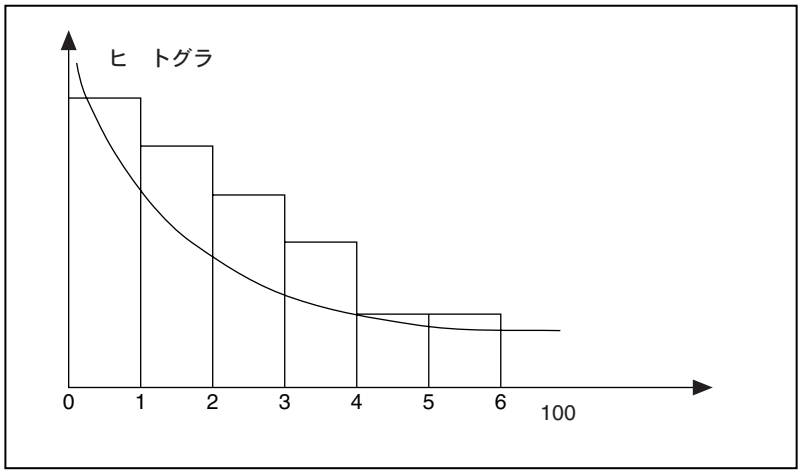
どのような無作為の実験でも必ず、特定の事象が起こる可能性と起こらない可能性があります。特定の事象が起こる可能性の程度つまり確率を示すため、0～1の範囲の値を割り当てます。ある事象が絶対に起こる場合、その確率は100%または1.0になり、絶対に起こらない場合は0になります。

簡単な例を考えてみましょう。偏りのないさいころが1つある場合、考えられる事象は6種類あり、1、2、3、4、5、または6のいずれかの目が出ます。2が出る確率はどのくらいでしょうか？　この確率は1/6、または0.16666になります。確率は、簡単な用語で定義することができます。すなわち、事象Aが起こる確率とは、同等の可能性がある結果の合計数に対する、Aとなる結果の数の比率です。

確率変数

多くの実験では、実数として解釈可能な結果が得られます。このような例としては、赤信号を通過する1日当たりの車の台数、候補者Aを支持している投票者の数、特定の交差点における事故の数などがあります。この実験の結果として得られる数値は、実験によって異なる可能性があるため、確率変数と呼ばれます。確率変数には、離散的なもの（取り得る値の数が有限なもの）と連続的なものがあります。連続的なものの例として、病院に来る患者の体重は、たとえば35キロから110キロの間の任意の値になり得ます。このような確率変数は、実数値範囲内のどのような数にもなり得ます。このような状況で、体重が正確に78.19キロである患者に遭遇する確率を求めたいと仮定します。次に、このような確率を計算する方法を、ある例を使用して説明します。

特定のタイプの50個の電池の寿命xを調べる実験を考えてみましょう。これらの電池は、同じ電池のより大きな集合から選択されたものです。観測されたデータのヒストグラムを、次に示します。



この図は、ほとんどの寿命は0～100時間の範囲内にあり、寿命が長くなるに従って度数の値が滑らかに減少していくことを示しています。

上記のヒストグラムは、指数関数的に減衰していく曲線で近似することができます。この関数を、データサンプルの動きに関する数学的モデルとして解釈することもできます。無作為に抽出された電池の寿命が400時間より長い確率を知りたい場合、この値は、値4より右側の曲線より下の部分の面積で近似することができます。確率変数のヒストグラムがモデル化された関数を、**確率密度関数**と呼びます。

上記のすべての内容を定義としてまとめると次のようになります。すなわち、実数区間でXが取り得る値の数が無限である場合、その確率変数Xは**連続形**であると言い、確率密度関数と呼ばれる次のような関数*f*(*x*)が存在します。

1. すべてのxについて $f(x) \geq 0$ となります。

2. $\int_{-\infty}^{\infty} f(x)dx = 1$

3. $P(a \leq X \leq b) = \int_{a}^{b} f(x)dx$

上記の式(3)から、特定の連続形な確率変数値については、次のようになることに注意してください。

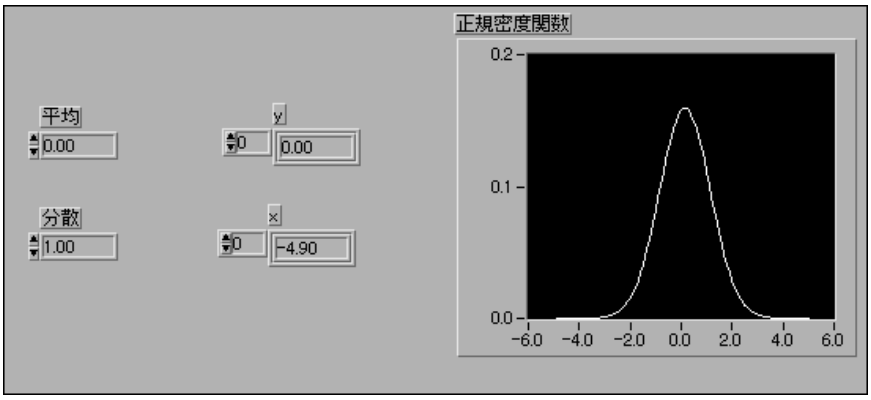
$$P(X = a) = \int_{a}^{a} f(x)dx = 0$$

どの特定の値に対しても確率0を割り当てることは意外なことではありません。理由は、確率変数が取り得る値の数が無限であるからです。したがって、これが特定の値$X = a$になる確率はきわめて小さくなります。

前の例では、確率密度関数として指数関数モデルを使用しました。確率密度関数にはさまざまな関数を選択できます。このような関数の一つである正規分布について、次に説明します。

正規分布

正規分布は、最も広く使用されている連続形確率分布の一つです。この分布関数は、次の図のような左右対称の釣鐘形になります。



この曲線の中心は、平均値$\bar{x} = 0$の位置にあり、広がりは分散$s^2 = 1$により示されます。この2つのパラメータにより、正規密度関数の形状と位置は完全に決まり、関数式の形は次のようになります。

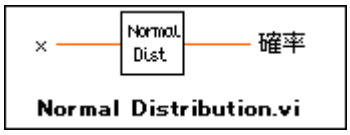
$$f(x) = \frac{1}{\sqrt{2\pi}s} e^{-(x-\bar{x})^2/(2s^2)}$$

確率変数Zが、平均値0、分散1で正規分布している場合、この確率変数の分布は**標準正規分布**であると言います。

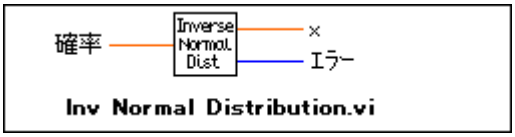
G Analysis Normal Distribution VIは、正規分布している確率変数xの片側確率pを計算します。

$$p = Prob(X \leq x)$$

ただし、*X*は平均値が0で分散が1である標準正規分布、*p*は確率、*x*は値です。



成人男性の身長を測定する調査を行う場合を考えます。無作為に選ばれた1,000人の男性に対してこの調査を行い、データセットSが得られました。ヒストグラムの分布を見ると、多くの測定値は平均身長の近くに集まっており、身長が非常に低い男性や非常に高い男性は集合の中には比較的少数しか存在しません。したがって、このヒストグラムは正規分布にかなり近似しています。次に、無作為抽出された別の1,000人の男性の中で、ある男性の身長が170cm以上である確率を求める場合を考えます。この確率を求めるには、Normal Distribution VIを使用します。入力を*x* = 170と設定します。このように、確率密度関数を選択することは、正しい確率値を求めるための基本になります。



Inverse Normal Distribution VIは、正反対の関数を実行します。このVIは、確率*p*が与えられたとき、正規分布しているサンプルの中にその確率で存在するような値xを求めます。たとえば、無作為に選ばれたデータセット内に存在する確率が60%であるような身長を求める場合です。

前述の通り、さまざまな確率密度関数を選択することができます。良く知られた、広く使用されている確率密度関数としては、カイ二乗分布、F分布、T分布があります。これらの分布に関する詳細は、「付録A　解析に関する参考文献」を参照してください。G解析ライブラリには、上記の各種分布の片側確率を計算するVI群が用意されています。また、その逆の演算を行うVI群もあります。

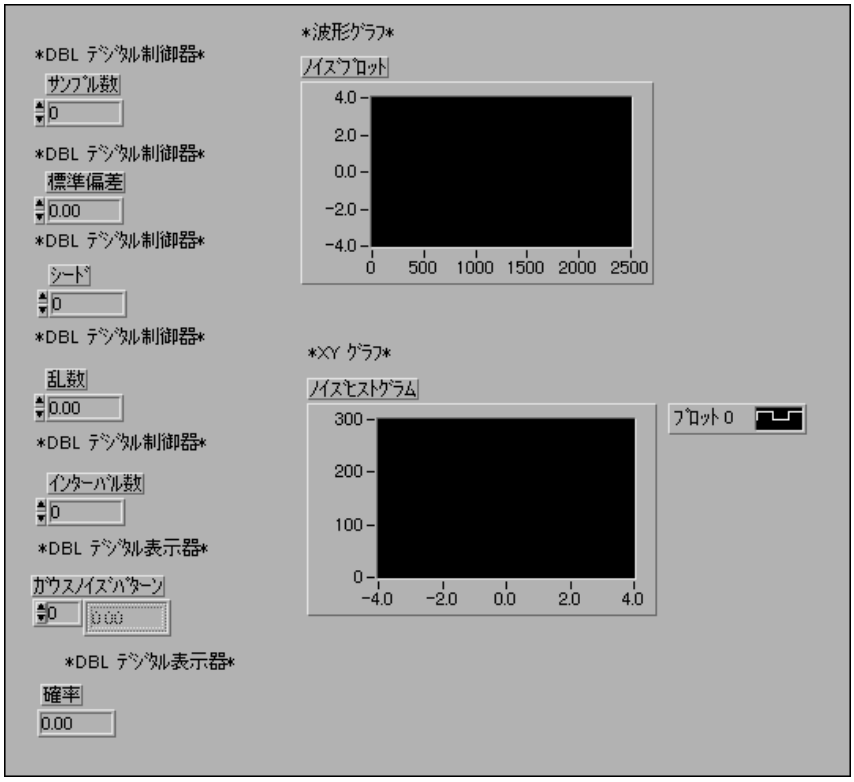
作業 19-1.　Normal Distribution VI を使用する

ここでは、確率についての中核となる概念を理解することが目的です。

この作業では、最初に標準正規分布するデータサンプルを生成し、次に Normal Distribution VI を使用して確率変数 x の確率を調べます。

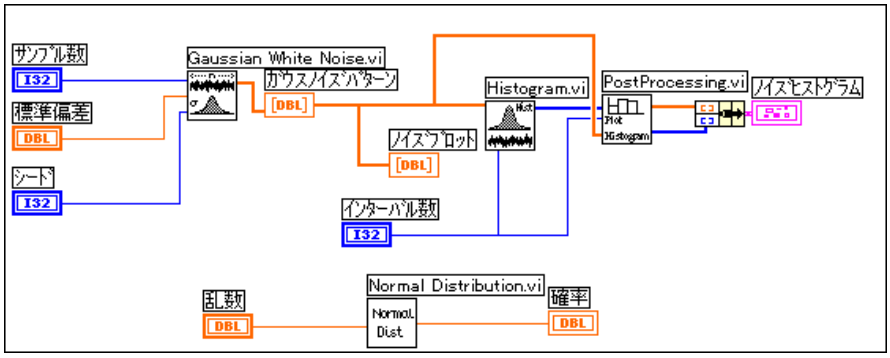
フロントパネル

1. 次の図のようなフロントパネルを作成します。ノイズプロットは波形グラフであるのに対し、ノイズヒストグラムは XY グラフです。



ブロックダイアグラム

2. 次の図のようなブロックダイアグラムを作成します。Gaussian White Noiseは、平均値が0で、ユーザが入力標準偏差で設定した標準偏差を持つ、ガウス分布パターンを発生します。サンプルは、Gaussian Noise Pattern のサンプル数です。シードは、乱数ノイズの発生に使用されるシード値です。Gaussian Noise Pattern を波形グラフノイズプロットに接続してください。



Gaussian White Noise 関数（**解析→信号生成**サブパレット）。この作業で、この関数はGaussian White Noiseパターンを発生します。

Histogram関数（**解析→確率と統計**サブパレット）。この作業で、この関数はGaussian Noise Patternのヒストグラムを計算します。

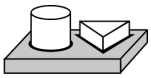
Normal Distribution 関数（**解析→確率と統計**サブパレット）。この作業で、この関数は正規分布する確率変数**乱数**の片側確率を計算します。

3. 前の作業で使用した Histogram VI を使用して、Gaussian Noise Pattern のヒストグラムを計算します。
4. 前述のように、別の方法でヒストグラムをプロットするために多少の後処理を行います。`LabVIEW¥Activity` ディレクトリの PostProcessing VI を選択します。
5. このVIの出力をまとめてノイズヒストグラムに接続します。
6. Normal Distribution VI を選択します。乱数制御器を入力端子に、出力を確率表示器に接続します。

7. フロントパネルに戻ります。サンプル数を 2,048 に、標準偏差を 1 に、シードを2に、インターバル数を10にセットして、VIを実行します。
8. ノイズプロットグラフ上にガウスのホワイトノイズが表示されます。このプロットから多くのことを理解するのは困難ですが、同じノイズパターンのヒストグラムプロットからは、多くの情報が得られます。このヒストグラムでは、大部分のサンプルが平均値0を中心に集まっていることがわかります。このヒストグラムから、このノイズパターンはNormal Distribution関数（ガウス分布）で近似できます。平均値が0であることと、標準偏差を1に設定してあることから、確率密度関数は実際には標準正規分布となります。

☞ **注**　**データを近似するのに適した正しい分布タイプを慎重に選択することが、非常に重要です。この例では、この決定を行うために実際にヒストグラムをプロットしました。多くの場合は、現象やデータサンプルの特性に関して事前に知識があれば、妥当な結論を出すことができます。**

9. フロントパネルに戻り乱数に値を入力します。このVIは、正規分布する確率変数の片側確率を計算します。ここでは、ヒストグラムの外観から変数が正規分布するものと仮定していることに注意してください。
10. このVIを、`Probability.vi` という名前で `LabVIEW¥Activity` ディレクトリに保存します。



これで作業 19-1 は完了です。

まとめ

- 情報とデータを読みとって妥当な結論を得るには、統計と確率に関するさまざまな概念が役に立ちます。
- サンプルから集合を推定するのに役立つ統計技法として、平均値、中央値、サンプル分散、モードなどがあります。
- ヒストグラムは、簡単で有効なデータ表示方法として広く使用されています。
- 確率理論を使用することで、サンプルから集合を推定して、その推定の精度を判断することができます。

第IV部

ネットワークとアプリケーション間通信

第IV部では、ネットワークとアプリケーション間通信に関する基本的な事項を説明します。

「第IV部 ネットワークとアプリケーション間通信」は、以下の章から構成されています。

- 「第20章 通信の概要」では、LabVIEWにおけるネットワークとアプリケーション間通信の扱いについて紹介します。
- 「第21章 TCPとUDP」では、インターネットプロトコル（Internet Protocol: IP）、ユーザデータグラムプロトコル（User Datagram Protocol: UDP）、転送制御プロトコル（Transmission Control Protocol: TCP）、インターネットアドレスについて説明し、TCPクライアント／サーバアプリケーションの例を示します。
- 「第22章 ActiveXのサポート」では、LabVIEWをActiveXサーバやクライアントとして使用する方法を示します。ActiveXは、OLE自動通信と同じです。
- 「第23章 DDEを使用する」では、Windows 3.1、Windows 95、Windows NTで行う動的データ交換（Dynamic Data Exchange: DDE）用のLabVIEW VIについて説明します。これらのVIは、DDE関数を実行することで、DDE接続に対応する他のアプリケーションとの間でデータを共有します。
- 「第24章 AppleEvent」では、複数のMacintoshアプリケーション間でのデータ交換を可能にする、Macintosh専用のアプリケーション間通信（inter application communication: IAC）の一形態であるAppleEventsについて説明します。
- 「第25章 プログラム間通信」では、プログラム間通信（Program-to-Program Communication: PPC）、Macintoshアプリケーションがデータブロックを送受信するためのAppleの低レベルアプリケーション間通信（inter application communication: IAC）の形態について説明します。

20

通信の概要

この章では、LabVIEW でのネットワーク動作とアプリケーション間通信の処理方法について概要を説明します。

LabVIEW での通信の概要

この部分の説明における**ネットワーク動作**という語は、複数の処理間の通信を意味し、場合によっては複数のコンピュータ間での通信も意味します。通常このような通信は、イーサネットやローカルトークなどのハードウェアネットワークを介して行われます。

ソフトウェアアプリケーションにおいてネットワーク動作が使用されるのは、基本的に 1 つまたは複数のアプリケーションで別のアプリケーションのサービスを使用する場合です。たとえばサービスを提供するアプリケーション（サーバ）は、専用のコンピュータ上で動作するデータ集録アプリケーションになったり、別のアプリケーションに情報を提供するデータベースプログラムになることができます。

ここでは、ネットワーク動作、通信に関する用語、ネットワーク接続されるアプリケーションのプログラミングについて、概要を説明します。

通信プロトコルの概要

複数のプロセス間で通信を行う場合は、各プロセスに共通の**プロトコル**という通信用言語を使用しなければなりません。

通信プロトコルを使用することにより、データがどのように転送されるかを知らなくても、送受信するデータや、送信元、送信先のロケーションを指定することができます。プロトコルは、ユーザコマンドを、ネットワークドライバが受け取れるデータに変換します。続いてネットワークドライバが必要に応じてネットワーク経由でのデータ転送処理を行います。

現在まで、通信に使用可能な規格としていくつかのネットワークプロトコルが出てきていますが、一般に、これらのプロトコルの間には互換性がありません。このため、通信アプリケーションにおいて最初にしなければならない仕事の一つが、使用するプロトコルを決定する作業です。既製品のアプリケーションで通信を行いたい場合は、そのアプリケーションでサポートされているプロトコルの範囲内で動作させる必要があります。

実際にアプリケーションを作成する場合は、もっと柔軟にプロトコルを選択できます。プロトコルの選択に影響を及ぼす要素としては、処理を実行するマシンのタイプ、使用可能なハードウェアネットワークの種類、アプリケーションに必要な通信の複雑性などがあります。

LabVIEW にはいくつかのプロトコルが組み込まれており、これらの中には、特定のコンピュータ専用のものもあります。LabVIEW でコンピュータ間の通信に使用されるプロトコルを、次に示します。

- TCP ― すべてのコンピュータで使用可能
- UDP ― すべてのコンピュータで使用可能
- DDE ― PC での Windows 上のアプリケーション間の通信に使用可能
- ActiveX ― Windows 95 と Windows NT で使用可能
- AppleEvents ― Macintosh 上の Macintosh アプリケーション間のメッセージ送信に使用可能
- PPC ― Macintosh 上の Macintosh アプリケーション間のデータの送受信に使用可能

各プロトコルはそれぞれ異なり、特にリモートアプリケーションのネットワークロケーションの参照方法に違いがあります。これらのプロトコルには互換性がないため、Macintosh と PC の間で通信を行いたい場合は、TCP のように両方に対応可能なプロトコルを使用する必要があります。

LabVIEW には、その他の通信オプションとして次のものがあります。

- System Exec VI ― この VI を使用すると、システムレベルのコマンドを実行できます。実際には 2 つの System Exec VI があり、1 つはすべてのバージョンの Windows で、もう 1 つは Sun と HP-UX で使用できます。
- Named Pipes ― UNIX でのみ使用可能
- HiQ ― Macintosh と PC でのみ使用可能

ファイルの共有と通信プロトコル

通信プロトコルの詳細を考える前に、自分のアプリケーションには別の方法の方が適切ではないか、検討してみてください。たとえば、専用システムでデータを集録し、別のコンピュータ上にデータを記録するようなアプリケーションの場合を考えてみます。

ネットワーク処理プロトコルを使用して集録用コンピュータからデータ記録用コンピュータにデータを送るアプリケーションを作成することもできます。記録用コンピュータでは、別のアプリケーションでデータをディスク上に格納します。

しかし、ネットワーク対応のほとんどのコンピュータに用意されているファイルの共有機能を使用すれば、もっと簡単になります。ファイルを共有する場合、オペレーティングシステムに組み込まれているドライバを使用して別のマシンに接続することができます。リモートマシンのディスクの記憶スペースは、自分のディスクの記憶スペースの延長として扱われます。2つのシステムを接続すると、ファイル共有機能では、ローカル接続されているのと全く同様にアプリケーションからリモートディスクに書き込むことができます。マシン間でデータ転送する方法としては、多くの場合、ファイル共有が最も簡単です。

クライアント／サーバモデル

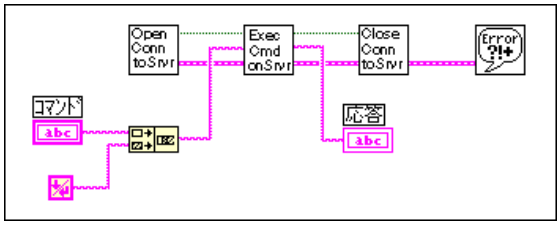
クライアント／サーバモデルは、ネットワーク対応アプリケーションにおいて最も一般的なモデルです。クライアント／サーバモデルでは、一方のプロセス（クライアント）が、もう一方のプロセス（サーバ）にサービスを要求します。

たとえば、アプリケーションの中で実世界から測定値を集録するように専用のコンピュータをセットアップすることができます。このコンピュータは、要求に応じて他のコンピュータにデータを提供するときはサーバとして機能します。また、このコンピュータが集録したデータを記録するよう、データベースなどの別のアプリケーションに要求する場合、このコンピュータはクライアントとして機能します。

LabVIEWでは、MacintoshのAppleEventsを除くどのプロトコルでも、クライアントサーバアプリケーションを使用することができます。AppleEventsは他のアプリケーションにコマンドを送るときに使用できます。LabVIEWでAppleEventsを使用する場合はコマンドサーバをセットアップすることはできません。Macintosh上にサーバ機能が必要な場合は、TCP、UDP、PPCのいずれかを使用してください。

クライアント用の一般モデル

次のブロックダイアグラムは、LabVIEWでの単純化されたクライアント用モデルを示します。



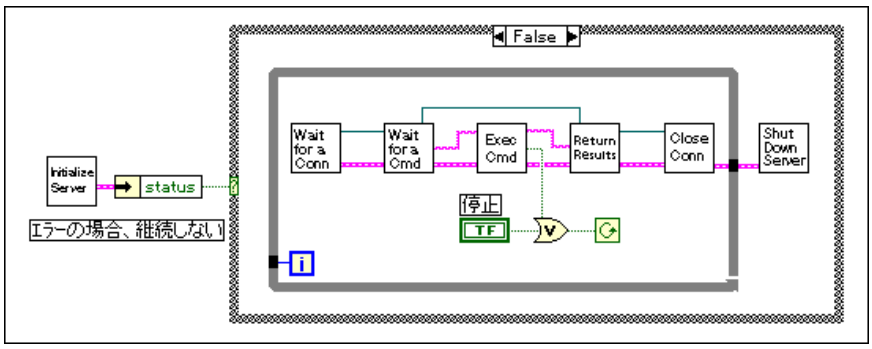
前のダイアグラムで、LabVIEW は最初にサーバへの接続を開きます。次にサーバにコマンドを送り、応答を読み込んでサーバへの接続を閉じます。最後に、通信処理中に発生したエラーを報告します。

高い性能を実現するためには、一度接続を開いた後、複数のコマンドを処理することもできます。コマンドを実行したら接続を閉じます。

ブロックダイアグラムのこの基本構造は、特定のプロトコルを LabVIEW で実装する方法を示すモデルとして、このマニュアルでも使用されています。

サーバ用の一般モデル

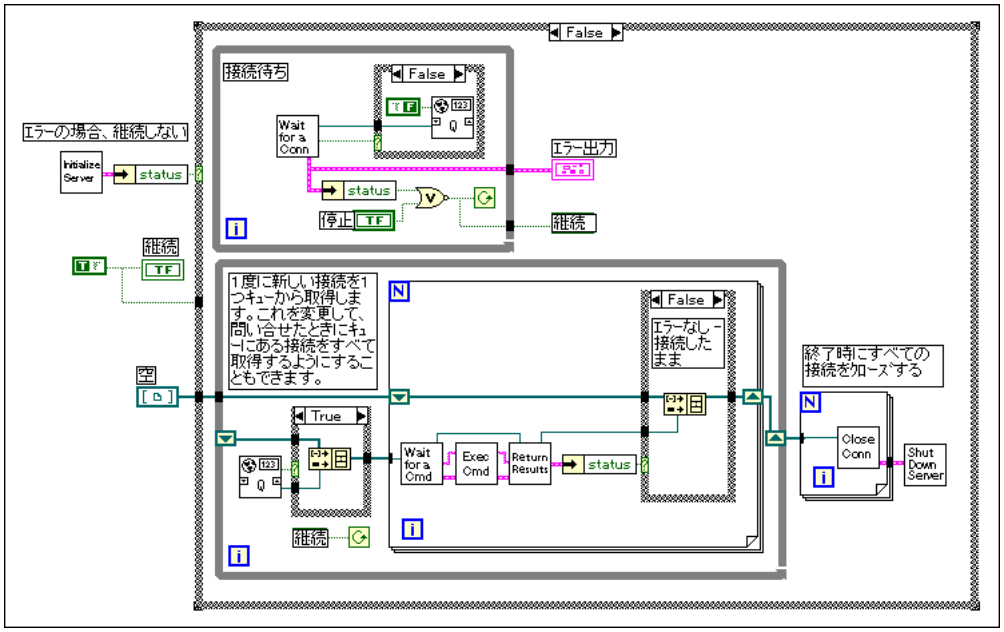
次のブロックダイアグラムは、LabVIEW での単純化されたサーバ用モデルを示します。



上記のダイアグラムで、LabVIEW は最初にサーバを初期化します。初期化が正常に行われた場合、LabVIEW はループに入り接続を待ちます。接続が行われると、コマンドの受信を待ちます。そしてコマンドを実行して結果を返します。この後、接続が閉じられます。ユーザがフロントパネルで停止ボタンを押すか、VI を停止するためのコマンドがリモートマシンから受信されて LabVIEW が停止するまで、この処理全体を繰り返します。

この VI ではエラーは報告されません。この VI では、コマンドが無効であることを示す応答が返される場合はありますが、エラーが発生してもダイアログは表示されません。サーバは無人運転される場合があるため、サーバでのエラー処理には注意が必要です。ダイアログボックスを表示するとユーザによる介入が必要となる（誰かが OK ボタンを押さなければならない）ので、サーバでのダイアログボックスの表示はしないと思われますが、LabVIEW で処理とエラーの内容をファイルや文字列に記録したい場合があるかもしれません。

接続を開いたままにして複数のコマンドを受信できるようにすると、性能を向上させることはできますが、このような動作を行うと、現在のクライアントが接続を解除するまで他のクライアントは接続できなくなってしまいます。プロトコルで複数の同時接続がサポートされている場合は、次の図のように複数のクライアントを同時に処理するよう LabVIEW の構成を変更することができます。



上記のダイアグラムでは、LabVIEW のマルチタスク機能を使用して 2 つのループを同時に動作させています。一方のループは連続的に接続を待ち、接続が受け付けられると、待ち行列に追加されます。もう一方のループは開いているそれぞれの接続をチェックし、受信したコマンドを実行します。いずれかの接続でエラーが発生した場合は、その接続が解除されます。ユーザがサーバを途中停止した場合は、開いているすべての接続が閉じられます。ブロックダイアグラムのこの基本構造は、特定のプロトコルを LabVIEW で実装する方法を示すモデルとして、このマニュアルでも使用されています。

21

TCP と UDP

この章では、インターネットプロトコル（Internet Protocol: IP）、ユーザデータグラムプロトコル（User Datagram Protocol: UDP）、転送制御プロトコル（Transmission Control Protocol: TCP）、インターネットアドレス、TCP クライアント／サーバアプリケーションの例について説明します。

概要

TCP/IP は当初、米国防衛高度研究企画庁（Defense Advanced Research Projects Agency: DARPA）向けに開発された一連の通信プロトコルです。TCP/IP は、開発以来広く認められるようになってきており、多くのコンピュータシステムで使用できます。

TCP/IP という名前は、その中で最も有名なプロトコルである転送制御プロトコル（TCP）とインターネットプロトコル（IP）に由来しています。TCP、IP、ユーザデータグラムプロトコル（UDP）は、基本的なネットワーク通信ツールです。

TCP/IP は、1 つのネットワークまたは相互に接続された複数のネットワーク（インターネット）上での通信を可能にします。個々のネットワークは、地理的に遠い距離にあっても構いません。TCP/IP は、1 つのネットワークまたはインターネット上のコンピュータから他のネットワークまたはインターネット上のコンピュータまでデータを送ります。TCP/IP はほとんどのコンピュータで使用できるため、さまざまなシステム間で情報を転送することができます。

インターネットプロトコル（IP）はネットワーク経由でデータを送信します。この下位プロトコルは、サイズの制限されたデータを受け取り、そのデータを**データグラム**としてネットワーク経由で送ります。IP ではデータが相手に届くことが保証されていないため、アプリケーションで IP が直接使用されることはほとんどありません。また、複数のデータグラムを送る場合は、ネットワークでの転送状況によって、到着するデータグラムの順序が違ったり、データグラムが複数回送られる場合があります。IP を基にして構築された UDP でも同様の問題があります。

TCP は、IP を使用してデータを転送する上位プロトコルです。TCP は、IP が管理可能な要素にデータを分割します。また、TCP にはエラー検出機能がある上、データが重複なしに正しい順序で届くことが保証されています。このような理由から通常、TCP はネットワークアプリケーション用としては最も良いとされています。

LabVIEWとTCP/IP

一連のTCP/IPプロトコルは、すべてのプラットフォーム上でLabVIEWに使用できます。LabVIEWには、クライアントVIやサーバVIの作成に使用するための一連のTCP VIとUDP VIが用意されています。

インターネットアドレス

IPネットワーク上の各ホストには、単一の32ビットのインターネットアドレスがあります。このアドレスは、ホストが接続されるインターネット上のネットワークと、そのネットワーク上の特定のコンピュータを認識します。このアドレスを使用することで、データの送信者や受信者を特定します。IPは、各データグラムが正しく送られるように、このアドレスをヘッダに付けます。

この32ビットアドレスを記述する一つの方法として、IPではドットを含む10進数表記を使用し、32ビットのアドレスを4つの8ビット数に分割しています。アドレスは、ドットで区切られた4つの整数として記述されます。以下の32ビットアドレスは、ドット付き10進数表記では`132.13.2.30`と表記されます。

```
10000100     00001101     00000010    00011110
```

32ビットアドレスを使用するには、この他に、IPアドレスに対応付けられた名前を使用する方法があります。ネットワークドライバは通常、この対応付け処理を実行するため、アドレスの対応情報を含むローカル**ホスト**ファイルを調べたり、ドメインネームシステムを使用して別のコンピュータシステムに問合せを行って、より大規模なデータベースを調べます。ネットワーク構成には、この処理を行うための**ホスト名分析**と呼ばれる正確なメカニズムが記述されています。

インターネットプロトコル（IP）

インターネットプロトコル（IP）は、マシン間でデータ転送を行う低レベルサービスを実行します。IPはデータを**データグラム**という要素にまとめます。データグラムに含まれるさまざまな情報の中には、データ、および送信元と送信先のアドレスを示すヘッダが含まれています。IPは、ネットワークやインターネットを経由してデータグラムを送るための正しいパスを判断し、指定された送信先にデータを送ります。

データが送られる際の完全なパスを、元のホストが知っている必要はありません。ネットワーク上のどのホストでも、ヘッダを使用して直接または別のホストに転送することで、送り先にデータを送ります。システムによって転送機能が異なるため、IPは必要に応じてデータグラムをより小さいセ

グメントに分割することができます。データが送信先に届くと、IPはデータを自動的に元の形に組み立てます。

IPはデータを届けるための最大の努力は行いますが、相手に届くというは保証はありません。また、IPはそれぞれのデータグラムの経路を個別に決めるため、データグラムの届く順序が狂うことがあります。実際に、転送時にパケットが複製されると、IPは同じパケットを複数回送ってしまうことがあります。IPはパケットの順序を判断しませんが、代わりにIPより上層に位置する上位プロトコルが、パケットの順序を整え、信頼性の高い送信を実現します。プログラムではTCPやUDPが使用されるため、IPが直接使用されることはほとんどありません。

ユーザデータグラムプロトコル（User Datagram Protocol: UDP）

UDPは、コンピュータ上のプロセス間での簡単な低レベル通信を提供します。プロセス間の通信は、送信先のマシンまたはポートにデータグラムを送ることで行われます。IPはマシン間の送信プロセスを行います。データがマシンにとどまりしだいUDPは、送信先ポートにデータグラムを転送します。送信先ポートに受信処理機能がない場合、そのデータグラムは破棄されます。IPでの送信エラーはすべて、UDPでもエラーとなります。

通常UDPは、信頼性がそれほど重要でないアプリケーションに使用されます。たとえば、アプリケーションによっては、有用なデータを送信先に頻繁に送るため、データのセグメントのうち多少の部分が失われても問題とならない場合があります。

UDPを使用する

UDPは、TCPのような接続を基本としたプロトコルではありません。したがって、送信先との接続が確立していなくても、データの送受信が行われます。各データグラムが送られるときに、データの送信先が指定されます。このシステムでは送信エラーは報告されません。

ポートを開くにはUDP Open VIを使用します。ポートとは、データが送信されるロケーションです。同時に開くUDPポートの数はシステムによって異なります。UDP Openはネットワーク接続refnumを返します。これは、このポートに関連する以降のすべての動作で使用される透過のデータです。

送信先にデータを送るにはUDP Write VIを、データを読み込むにはUDP Read VIを使用します。各書き込みごとに、送信先のアドレスとポートが必要です。また、各読み込みごとに送信元のアドレスとポートが必要です。パケットの境界は維持されますので、2つの個別の書き込み動作で送

られたデータが、1回に読み込まれるデータに含まれることは絶対にありません。

理論的には、どのようなサイズのデータパケットでも送ることができます。必要に応じ、パケットはより小さい断片に分割されて送られます。送り先では各断片が組み立て直され、要求したプロセスにパケットが渡されます。実際には、システムはある程度のメモリしかパケットの再構築用に割り当てません。再構築できないパケットは破棄されます。分割せずに送信可能なパケットの最大サイズは、ネットワークハードウェアによって異なります。

LabVIEWがすべての通信を終了したら、UDP Close VIを呼び出してシステムリソースを解放します。

転送制御プロトコル（TCP）

TCPではネットワーク経由の信頼性の高い転送を実現でき、データはエラーや喪失、重複を起こすことなく正しい順序で届けられます。TCPにデータを渡すと、TCPはこのデータに付加情報を付けてIPに渡します。IPはこのデータをデータグラムにして送信します。受信側ではこの逆の処理が行われ、TCPはデータにエラーがないかチェックし、データを正しい順序に並べ、正常に送信されたことを通知します。送信側のTCPが通知を受信しなかった場合は、そのデータセグメントを再送信します。

TCPを使用する

TCPは接続を基本にしたプロトコルであるため、データを転送する前に双方のサイトで接続を確立しなければなりません。TCPでは、同時に複数の接続が可能です。

接続を開始するには、入って来る接続を待つか、指定されたアドレスとの間の接続をアクティブに要求します。TCP接続を確立するには、アドレスと、そのアドレスにあるポートの両方を指定する必要があります。ポートは、0から65535の間の数で指定します。UNIXシステムでは、1024より小さいポート番号は、特権アプリケーション用に予約されています。同じアドレスでもポートが異なれば、そのアドレスにおける別のサービスとなりますので、複数の同時接続を容易に管理できます。

特定のアドレスとポートに対して能動的に接続を確立するには、TCP Open Connection関数を使用します。この関数を使用して、通信相手のアドレスとポートを指定します。正常に接続できた場合、この関数はその接続を固有に識別できる接続IDを返します。これ以後の関数呼び出しでこの接続を参照するには、この接続IDを使用します。

入って来る接続を待つ方法は、2つあります。

- 第1の方法では、TCP Listen VIを使用してリスナを作成し、指定したポートでTCP接続が受け付けられるのを待ちます。正常に接続された場合、このVIは接続IDとリモートTCPのアドレスとポートを返します。
- 第2の方法では、TCP Create Listener関数を使用してリスナを作成し、Wait on Listener関数を使用して新しい接続を待機し、受け付けます。Wait on Listenerは、関数に渡されたのと同じリスナIDと、その接続の接続IDを返します。新しい接続を待つのを終了する場合は、TCP Closeを使用してリスナを閉じます。リスナに対して読み込みや書き込みを行うことはできません。

第2の方法を使用するメリットは、TCP Closeを呼び出してリスン操作を取り消せることにあります。この機能は、タイムアウトを使用せずにリスン操作を行い、他の条件が生じたとき（ユーザがボタンを押したときなど）にリスン操作を取り消したい場合に役立ちます。

接続が確立されると、TCP Read関数やTCP Write関数を使用してリモートアプリケーションとの間でデータの読み込みや書き込みを行えます。

最後に、TCP Close Connection関数を使用してリモートアプリケーションへの接続を閉じます。まだ読み込まれていないデータがあるのに接続が閉じられた場合はそのデータは失われる可能性があります。接続されている場合は、上位プロトコルを使用して、接続を閉じるタイミングを決定する必要があります。一度接続を閉じると、それ以後読み込みや書き込みはできません。

UDPとTCPの比較

クライアントとサーバの両方を自分で作成し、システムでTCP/IPを使用できる場合、使用するプロトコルとしてはおそらくTCPが最適です。これは、TCPは信頼性が高く、接続を基本としたプロトコルであるからです。UDPは高性能な非接続プロトコルですが、データ転送の信頼性が保証されません。

TCPクライアントの例

TCPプロトコルを含むクライアントブロックダイアグラムモデルの構成要素の一般的な使用方法について説明します。

サーバへの接続を開くにはTCP Open Connection関数を使用します。ユーザはサーバのインターネットアドレスとサーバ用の**ポート**を指定する必要があります。このアドレスにより、ネットワーク上のコンピュータが特定されます。ポートとは、サーバが通信要求のリスン操作のために使用するコンピュータ上の通信チャネルを指定するための付加的な番号です。TCPサーバを作成する場合は、サーバでの通信に使用するポートを指定します。

サーバ上でコマンドを実行するには、TCP Write 関数を使用してサーバにコマンドを送ります。次に、TCP Read関数を使用してサーバから結果を読み返します。TCP Read関数を使用する場合は、読み込みたい文字数を指定する必要があります。応答の長さが変動する可能性があるため、この指定がうまくいかない場合があります。サーバでも、コマンドの長さが異なる可能性があるためにコマンドに関して同様のことが問題になる場合があります。

コマンドのサイズが異なる問題には、以下の方法で対応することができます。

- コマンドや結果の前に、コマンドや結果のサイズを指定する固定したサイズパラメータを指定する。この場合は、サイズパラメータを読み込んでから、サイズで指定された数の文字を読み込みます。このオプションは効果的で柔軟性もあります。
- 各コマンドや結果のサイズを固定する。このサイズよりコマンドが小さい場合は、固定サイズに合うようコマンドを拡大します。
- 各コマンドや結果の後に、特定の終了文字を付ける。この場合は、終了文字を読み込むまで小さいかたまりでデータを読み込む必要があります。

サーバへの接続を閉じるにはTCP Close Connection関数を使用します。

タイムアウトとエラー

前の項では、サーバ用の通信プロトコルについて説明しました。ネットワークアプリケーションを設計する場合は、サーバがクラッシュしたときの各クライアントVIの処理など、何らかのエラーがあった場合の対処をよく考えておく必要があります。

対処法の一つとして、各VIにタイムアウトを設定する方法があります。何らかのエラーがあって結果が得られなかった場合、クライアントは一定時間後に実行を再開します。処理を継続する中で、クライアントは実行の再開を試みることもできますし、エラーを報告することもできます。必要ならば、無理をせずにクライアントアプリケーションを停止することもできます。

TCPサーバの例

次に、TCPを使用して一般的なサーバモデルの各構成要素を実現する方法を説明します。

TCPでは初期化が必要ありませんので、このステップは除外できます。

接続を待つにはTCP Listen VIを使用します。ユーザは通信に使用するポートを指定する必要があります。このポートは、クライアントが接続しようとしているのと同じポートでなければなりません。詳しくは、本章の「TCPクライアントの例」の項を参照してください。

接続が確立されたら、そのポートから読み込みを行ってコマンドを検索します。TCPクライアントの例の項で説明したように、コマンドの形式を決定する必要があります。コマンドの前に長さのフィールドがある場合は、最初にそれを読み込んでから、指定された長さのデータを読み込みます。

コマンドはローカルコンピュータ上で実行されますので、コマンドの実行はプロトコルからは独立していなければなりません。実行が完了したら、結果を次のステージに渡し、そこで結果をクライアントに送信します。

結果を返すにはTCP Write関数を使用します。「TCPクライアントの例」の項で説明したように、データはクライアントが受信可能な形になっていなければなりません。

接続を閉じるには、TCP Close Connection関数を使用します。

接続を閉じるとすべてが完了しますので、TCPではこのステップを省略することができます。

複数接続を含むTCPサーバ

TCPは複数の接続を容易に処理できます。前項で説明した方法を使用すると、複数の接続を扱うサーバの構成要素を実装することができます。

セットアップ

使用するコンピュータによってセットアップが異なりますので、TCP/IPを使用するには、セットアップが正しく行われているか確認する必要があります。

UNIX

TCP/IPのサポート機能が組み込まれています。ネットワークが正しく構成されていれば、LabVIEWで別のセットアップを行う必要はありません。

Macintosh

Macintosh のオペレーティングシステムバージョン 7.5 以降には TCP/IP が組み込まれています。旧バージョンのシステムで TCP/IP を使用するには、Apple Programmer Developer Association（APDA）が提供している MacTCP ドライバソフトウェアをインストールする必要があります。MacTCP ドライバのライセンスに関する詳細は、米国 (800)282-2732 で APDA にお問い合せください。また、LabVIEW は Open Transport でも動作します。

Windows 3.*x*

TCP/IP を使用するには、イーサネットボードと、それに対応する低水準ドライバをインストールする必要があります。さらに、規格 1.1 に適合する Windows Sockets（WinSock）DLL が組み込まれた TCP/IP ソフトウェアを購入してインストールする必要があります。WinSock はさまざまなネットワークドライバを使用してアプリケーション通信を可能にする標準インタフェースです。数社のメーカが、WinSock DLL が組み込まれたネットワークソフトウェアを提供しています。メーカの指示に従って、イーサネットボード、ボードドライバ、WinSock DLL をインストールしてください。

各種のボードで動作するWinSockドライバはいくつかのメーカが提供しています。ボードで使用可能な WinSock DLL が提供されているかどうかを調べるには、各ボードのメーカに問い合わせの上、メーカの指示に従って WinSock DLL をインストールしてください。

ナショナルインスツルメンツは、正常に動作するかに関しさまざまな WinSock DLL についてテストしてきました。テストの結果、多くの DLL は規格に完全に適合していないことが判明しました。DLL をご購入の際にはデモバージョンを試してから実際のバージョンを購入されることをお勧めします。通常、デモバージョンはメーカから入手できます。デモバージョンの多くは完全な機能を備えていますが、一定期間後に使用期限が切れるようになっています。

Windows 95 と Windows NT

Windows 95 と NT には TCP のサポート機能が組み込まれていますので、TCP/IP 用の DLL を使用しなくても TCP による通信を行えます。

22

ActiveXのサポート

この章では、LabVIEWをActiveXサーバやクライアントとして使用する方法を説明します。

ActiveXオートメーションを使用すると、あるWindowsアプリケーションの、通常はオブジェクトに分類されているプロパティやメソッドにアクセスして、これらを他のWindowsアプリケーションに使用することができます。Windows 3.x用のLabVIEWでは、ActiveXオートメーションがサポートされておりません。ActiveXオートメーションは、Windows 95およびNTでのみサポートされています。

1つのアプリケーションは、サーバまたはクライアントのいずれかのオートメーションをサポートします。オブジェクトを表示しそのオブジェクト上で操作する方法を提供するアプリケーションはActiveXオートメーションサーバであり、他のアプリケーションにより表示された方法を使用するアプリケーションはActiveXオートメーションクライアントです。

LabVIEWは、ActiveXサーバとしてもActiveXクライアントとしても機能できます。またLabVIEWは、ActiveXコンテナを使用してActiveXオブジェクトをフロントパネルに表示することもできます。ActiveXに関する詳細は、『Gプログラミングリファレンスマニュアル』の「第15章 ActiveXの制御器」を参照してください。

☞ **注** **本書では、ActiveXという用語はマイクロソフト社のActiveX技術とOLE技術を意味します。**

ActiveXに関する一般的な説明は、Kraig Brockschmidt著の『Microsoft Developers Network documentation and Inside OLE, 2nd edition』を参照してください。

ActiveXオートメーションサーバの機能

LabVIEW を ActiveX オートメーションサーバとして使用できるため、ActiveX を使用できる別のアプリケーション（Microsoft Excel など）は、LabVIEW や個々の VI 群からのプロパティやメソッドを要求することができます。

LabVIEW を ActiveX サーバとして起動するには、**編集→環境設定→サーバ：構成**を選択します。次のようなダイアログボックスが表示されます:

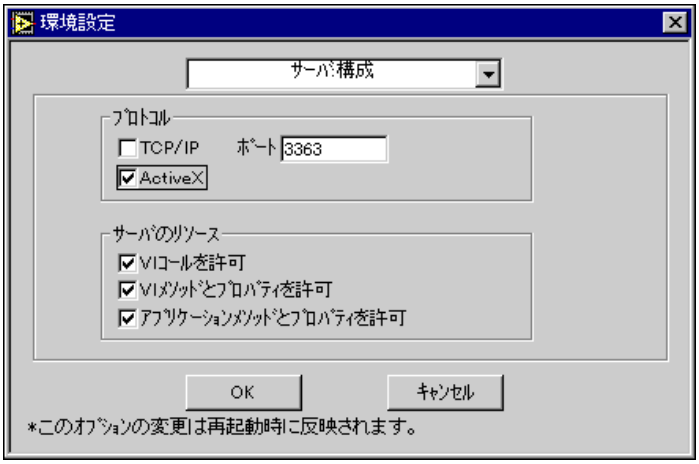


図 22-1 環境設定ダイアログボックス、サーバ構成

プロトコルとして ActiveX を選択します。LabVIEW は作成可能なクラスである Application とディスパッチクラスである VirtualInstrument を ActiveX にエクスポートします。progId の `LabVIEW.Application` または `LabVIEW.Application.5` は、アプリケーションオブジェクトを生成します。Application method GetVIReference は、Virtual Instrument オブジェクトに対するポインタを生成して返します。

ActiveXサーバのプロパティとメソッド

エクスポートされたクラスのプロパティとメソッドの一覧表は、『LabVIEWオンラインリファレンス』を参照してください。

プロパティとメソッドの使用方法の例は、examples¥commディレクトリの¥freqresp.xlsに示されています。この例では、Visual Basicのスクリプトマクロを使用してVIを実行し、結果を表にします。

ActiveXオートメーションクライアントの機能

LabVIEWは、他のActiveXオートメーションサーバを制御するActiveXオートメーションクライアントとして機能することができます。LabVIEWは、プロパティの読み込みと設定を行ったり、ActiveXサーバにより使用可能になったメソッドを実行することができます。サーバはType Libraryファイルを介して、サーバのオブジェクト、メソッド、プロパティについての情報をエクスポートします。タイプライブラリは通常、サーバが作成された環境で生成されます。詳しくは、各サーバアプリケーションの資料を参照してください。

表22-1は、ActiveXオートメーションクライアントに使用できる関数の一覧とその説明を示します。

表22-1 ActiveXオートメーションクライアントをサポートする関数

関数	説明
Automation Open	開くオートメーションクラスを選択する
Invoke Node	クラスの関数を実行する
Property Node	クラスのプロパティを設定または読み込む
Automation Close	オートメーションrefnumを閉じる

Cを使用してクライアントアプリケーションを作成するには、次のようにします。

1. アクセスしたいメソッドを含むオブジェクトのIDispatchインタフェースを読み込みます。
2. そのオブジェクトのメソッドのDispatchIDを読み込みます。
3. IDispatchインタフェースのInvokeを使用し、すべてのパラメータをパラメータリストにまとめて、メソッドを起動します。

LabVIEW でクライアントアプリケーションを作成するには、次の手順に従います。

1. Automation Open 関数を使用して、IDispatch インタフェースを固有に定義する自動化 refnum を読み込みます。
2. Invoke Node を使用して、そのオブジェクトに所属するメソッドを実行します。サーバアプリケーションで ActiveX バリアント形式のデータが必要な場合、LabVIEW はデータタイプを ActiveX バリアントに変換します。

次の「ActiveX のクライアントの例」の項で、これらのノードの使用方法を説明します。

ActiveX のクライアントの例

次の例は、上記のリストに含まれている新しい ActiveX 関数の使用方法を示します。

ActiveX バリアントのデータを G のデータに変換する

図 22-2 の最初の例は、ActiveX バリアントのデータを G のデータに変換する方法を示します。どの ActiveX アプリケーションでも、最初に ActiveX オートメーション refnum を開き、最後にオートメーション refnum を閉じる必要があります。この例では、Microsoft Excel のアプリケーションオブジェクトを開き、Property Node を使用してプロパティを見える形で表示します。見える形のプロパティは、ActiveX バリアントの形式になっています。To G Data 関数は、プロパティ情報を、LabVIEW でサポートされている形式に変換する必要があります。

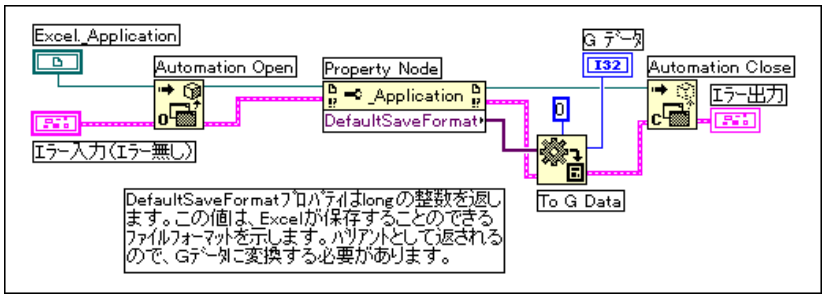


図 22-2 ActiveX 用のデータから G のデータへの変換を示すブロックダイアグラム

他のアプリケーションのプロパティ情報についての詳細は、各アプリケーションに対応するオンラインヘルプを参照してください。『LabVIEW オンラインリファレンス』には、ActiveX を使用可能な他のアプリケーションのプロパティ情報は含まれていません。

LabVIEW から Microsoft Excel にワークブックを追加する

図 22-3 に示す第 2 の例は、LabVIEW から Microsoft Excel にワークブックを追加します。この項の前の部分で説明したように、Automation Open 関数で ActiveX アプリケーションの refnum を開き、Automation Close で ActiveX アプリケーションを閉じる必要があります。別のワークブックを追加するには、ワークブックに refnum を付けなければなりません。この例では、Automation Open で Excel の refnum を開き、Property Node でワークブックの refnum にアクセスします。Excel にワークブックを追加した後、そのワークブックを参照するための refnum が LabVIEW に返されます。Excel を開いておく必要がなくなったら、Excel とワークブックの refnum を閉じます。

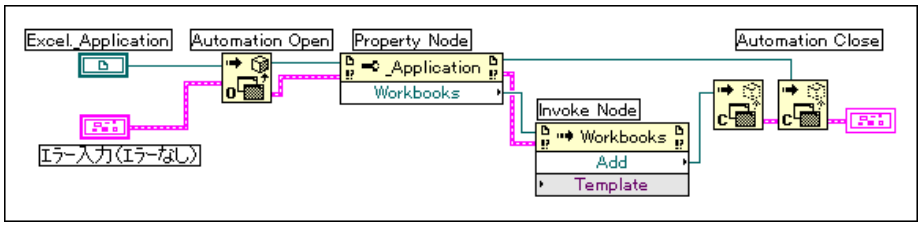


図 22-3 Microsoft Excel にワークブックを追加する

23

DDE を使用する

この章では、Windows 3.1、Windows 95、Windows NT のダイナミックデータ交換（Dynamic Data Exchange: DDE）用の LabVIEW VI について説明します。これらの VI は、DDE 関数を実行することにより、DDE 接続に対応した他のアプリケーションとの間でデータを共有することができます。

DDE の概要

ダイナミックデータ交換（DDE）は、Windows アプリケーション間でデータ交換を行うためのプロトコルです。

TCP/IP による通信では、アプリケーションは通信ラインを開いてから生データを転送します。DDE はこれよりも上位で動作し、複数のアプリケーションは互いにメッセージを送って情報を交換します。単純なメッセージの一つに、他のアプリケーションにコマンドを送るためのものがあります。その他のメッセージのほとんどはデータ転送を処理するためのものであり、データは名前で参照されます。

アプリケーションがともに動作しており、両方がコールバック関数のアドレスを Windows に指定してからでないと、DDE 通信を開始することはできません。コールバック関数は、Windows がアプリケーションに送るすべての DDE メッセージを受け付けます。

DDE クライアントは、接続メッセージを送ることにより他のアプリケーション（DDE サーバ）との間の通信を開始します。接続が確立すると、クライアントはサーバにコマンドを送ったり、サーバが管理しているデータの値を変更したり要求することができます。

クライアントは、要求またはアドバイスによりサーバにデータを要求することができます。クライアントは要求によってデータの現在の値を問い合わせます。クライアントは、ある期間にわたって値をモニタしたい場合は、変更についてアドバイスするように要求しなければなりません。データ値をアドバイスするように要求することで、クライアントはサーバとの間にリンクを確立し、データに変更があった場合にサーバはこのリンクを介してクライアントにアドバイスします。クライアントがデータ値のモニタを停止するには、アドバイスリンクを停止するようにサーバに指示します。

会話のための DDE 通信が完了すると、クライアントはデータ交換を閉じるためのメッセージをサーバに送ります。

Microsoft Excelなどの標準的な既製アプリケーションとの通信には、DDEが最適です。

LabVIEW を使用すると、他のアプリケーションに対してクライアントとして機能する（他のアプリケーションにデータを要求したり送信することを意味します）VIを作成することができます。また、他のアプリケーションからアクセスするための名前で指定された情報を提供するサーバとして機能するVIを作成することもできます。サーバとしてのLabVIEWは接続を基本にした通信を使用しません。その代わりにユーザは名前で指定した情報を他のアプリケーションに提供し、他のアプリケーションは名前によりこの情報を読み込んだり値を設定することができます。

サービス、トピック、およびデータ項目

TCP/IP を使用する場合は、コンピュータのアドレスとポート番号により、会話したい相手のプロセスを指定します。DDEの場合は、サービス名とトピック名を参照することで、会話したい相手のアプリケーションを指定します。サーバは、任意のサービス名とトピック名を決定します。特定のサーバは通常、そのアプリケーション名をサービスに使用しますが、そうでない場合もあります。そのサーバは、通信を行いたいトピックを複数提供することができます。たとえばExcelの場合はスプレッドシート名などがトピックになります。

サーバとの通信を行うには、まず会話したいサービス名とトピック名を検索します。次にこの2つの名前を使用してサーバを指定し、会話を成立させます。

サーバにコマンドを送る場合を除き、通常はサーバが会話したいデータ項目についての処理を行います。これらのデータ項目は、サーバを使用してユーザが操作可能な**変数**のリストとして扱うことができます。名前により変数に新しい値を与えて変更したり、名前により変数の値を要求することができます。

Excelとのクライアント通信の例

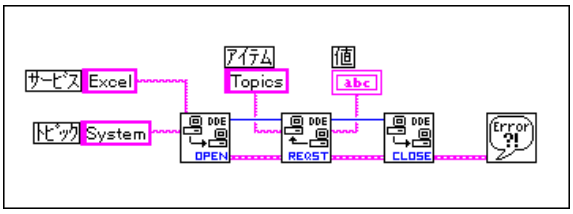
DDEをサポートする各アプリケーションごとに、データ交換可能なサービス、トピック、およびデータ項目は異なります。たとえば、2つの異なるスプレッドシートプログラムでは、セルの指定方法が大きく異なる場合があります。特定のアプリケーションでのサポート内容については、各アプリケーションに付属の資料を参照してください。

Microsoft Excel（一般に広く使用されているWindows用のスプレッドシートプログラム）はDDEをサポートしているため、DDEを使用してExcelにコマンドに送ることができます。また、名前によりスプレッドシートのデータを操作したり読み込むことができます。ExcelでのDDEの使用方法の詳細は、『Microsoft Excelユーザーズガイド2』を参照してください。

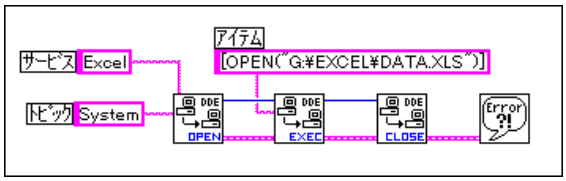
Excelの場合、サービス名は[Excel]となります。トピック名には、スプレッドシートの文書などの開いている文書名、または[System]という文字列を使用することができます。

[System]という名前を使用する場合は、Excelのステータス情報を要求したり、Excelに一般コマンド（特定のスプレッドシート向けでないコマンド）を送ることができます。たとえば[System]というトピックの場合、Excelは[Status]などの項目について報告します。Statusの値は、Excelが処理中である場合はBusy、コマンドを実行できる状態にある場合はReadyになります。トピックがSystemであるときに使用できるもう一つの便利なデータ項目が、[Topics]です。[Topics]はExcelがデータ交換するトピックのリストを返します。このリストの中には、開いているすべてのスプレッドシート文書や[System]トピックが含まれています。

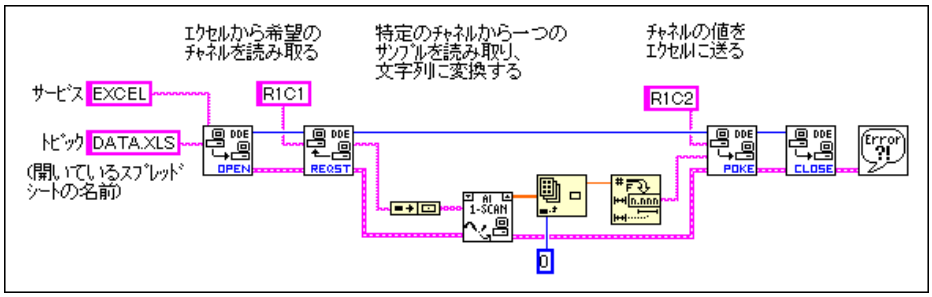
次のVIは、LabVIEWでのTopicsコマンドの使用方法を示します。返される値は、開いているスプレッドシート名と[System]という文字を含む文字列です。



この他にExcelでは、特定の文書を開くようにExcelに指示するときにも[System]トピックを使用できます。このためには、次のLabVIEWのダイアグラムに示すように、DDE Execute.viを使用して、文書を開くようにExcelに指示するExcelマクロを送ります。



スプレッドシートファイルを開いた後、スプレッドシートにコマンドを送ってセル値を読み込むことができます。この場合、トピックはスプレッドシート文書の名前になります。項目はセルの名前、セルの範囲、またはスプレッドシート内の名前の付けられた部分です。たとえば次の図では、LabVIEW は第 1 行第 1 列のセルの値を取り出すことができます。続いて LabVIEW は指定されたチャネルからサンプルを集録し、結果として得られたサンプルを Excel に返します。



DDE サーバとしての LabVIEW VI

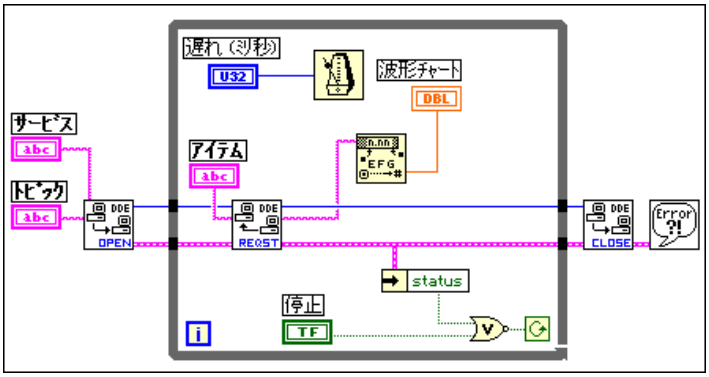
データ項目用のサーバとして動作する LabVIEW VI を作成することができます。一般的な概念として、LabVIEW VI はトピック内の特定のサービスに関する情報を提供できるということです。LabVIEW は、任意の名前をサービス名とトピック名に使用できます。また、サービス名がアプリケーション名（[LabVIEW]）になるように指定したり、トピック名がサーバ VI 名または[Lab Data]のようにLabVIEWが提供するデータの一般的な分類になるように指定することもできます。

サーバ VI は、報告する特定のサービス用のデータ項目を登録します。LabVIEW は、データ名とその値を記憶しており、そのデータに関する他のアプリケーションとの通信を処理します。DDE 通信用に登録されたデータの値をサーバ VI が変更する場合、LabVIEW は、そのデータに関する通知を要求したすべてのクライアントアプリケーションに通知します。同様に、他のアプリケーションがデータ項目の値を変更するための [Poke] メッセージを送った場合も、LabVIEW がこの値を変更します。

サーバとして機能している LabVIEW VI に対しては、DDE Execute コマンドを使用できません。VI にコマンドを送る場合は、データ項目を使用して送る必要があります。

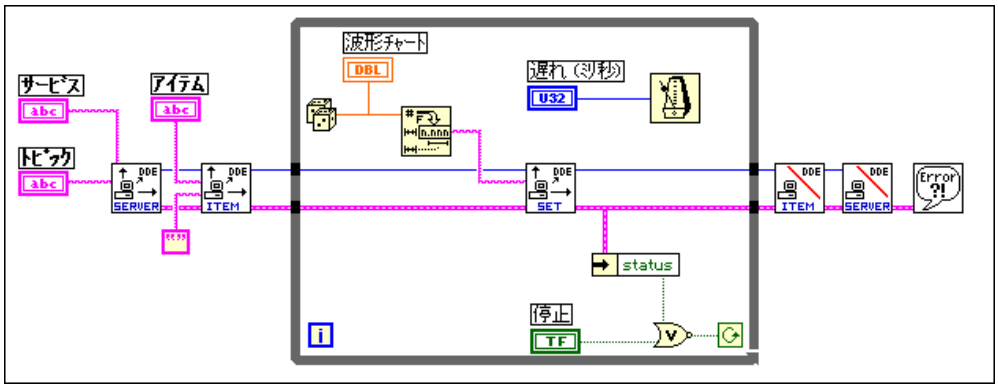
現在のところLabVIEWにはExcelのSystemトピックに相当するものがないことに注意してください。LabVIEWアプリケーション自体は、コマンドを送ったりステータス情報を要求する相手となるサーバではありません。LabVIEW VIはサーバとして機能はしますが、LabVIEW自体は他のアプリケーションに対してサービスを提供しないのでご注意ください。

次の例は、他のクライアントアプリケーションにデータを提供するDDEサーバVIを作成する方法を示します。この場合のデータは乱数です。この乱数を、データ集録ボードや、GPIB、VXI、またはシリアル接続によりコンピュータに接続されたデバイスからの実際のデータに容易に置き換えることができます。



上図のVIはLabVIEWにサーバを登録します。このVIはクライアントに提供する項目を登録します。ループの中で、VIは周期的に項目の値を設定します。前述のように、LabVIEWはデータが用意できたことを他のアプリケーションに通知します。ループが完了すると、VIは項目とサーバの登録を解除して処理を完了します。

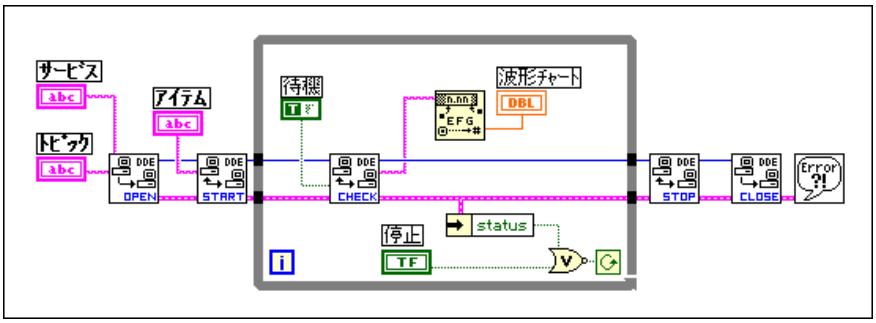
他のLabVIEW VIなど、DDEを理解できるアプリケーションならどれでも、このVIのクライアントになることができます。次の図は、前図のVIに対するクライアントを示します。この場合サービス名、トピック名、項目名は、サーバで使用された名前と同じであることが重要です。



データの要求とデータのアドバイスの比較

前のクライアントの例では、ループの中でDDE Request VIを使用してデータを取り出しました。DDE Requestを使用した場合、前にデータを参照したかどうかに関係なくデータは直ちに取り出されます。サーバとクライアントのループ実行速度が完全に同じでない場合は、データが重複したり失われる恐れがあります。

データの重複を避ける方法の一つとしてDDE Advise VIを使用してデータ項目の値の変化を通知するよう要求する方法があります。この方式を実装する方法を次の図に示します。



前図で、LabVIEWは会話を始めます。次にLabVIEWはDDE Advise Start VIを使用して、データ項目の値の変化を通知するよう要求します。ループを実行するたびに、LabVIEWはデータ項目の値の変化を待つDDE Advise Check VIを呼び出します。ループが終了すると、LabVIEWはDDE Advise Stop VIを呼び出して会話を閉じ、通知ループを終了します。

データの同期化

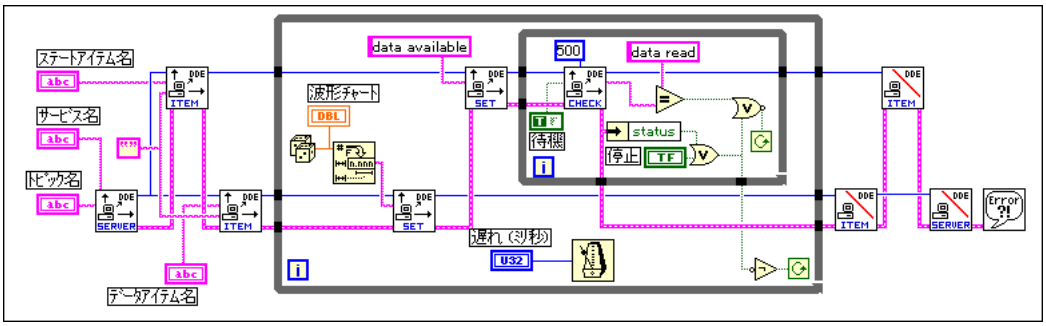
前項のクライアントサーバの例はデータの監視用としては良好に動作しますが、サーバが送るすべてのデータがクライアントに受信されるかどうかは保証されません。DDE Adviseループを使用しても、クライアントがデータの変更をチェックする頻度が十分でなければ、サーバが送ったデータ値をクライアントが受け取れない恐れがあります。

アプリケーションによっては、データが失われることが問題にならない場合もあります。たとえばデータ集録システムを監視するような場合、一般的な傾向を観測するのであれば、データの損失が問題になることは恐らくありません。ただし他のアプリケーションでは、データの損失がないことを確認した方が良いでしょう。

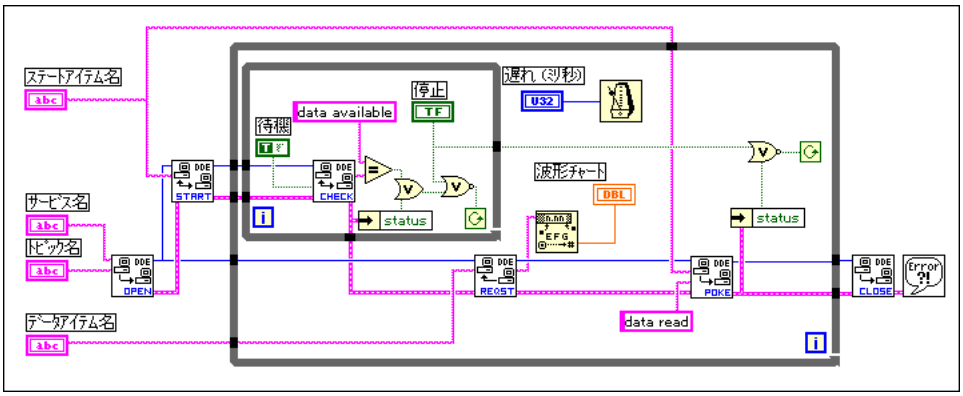
TCPとDDEの主な違いの一つは、TCPではデータが待ち行列に登録されるためデータが失われず、正しい順序で読み込まれることです。DDEではこのようなサービスを提供していません。

DDEでは、ユーザは個別の項目をセットアップし、クライアントはこの項目を使用して最新のデータを受け取ったことを通知することができます。この後、クライアントが前のデータを受け取ったことを通知してきた場合のみ、新しいポイントを含んだ集録されたデータ項目を更新します。

たとえば、本章の「データの要求とデータのアドバイスの比較」の項に示したサーバの例を修正して、集録したデータ項目を修正してから**状態**項目を特定の値に設定するようにできます。この場合サーバは、クライアントからデータを受け取ったことが通知されるまで**状態**項目をモニタします。この修正を、次のブロックダイアグラムに示します。



次の図のように、このサーバのクライアントは、**データの変更が使用可能になる**まで状態項目を監視します。状態項目が変わった時点で、クライアントはサーバから送られた集録データ項目のデータを読み込み、その後、状態項目を**データ読み込み**値に更新します。



この方法を使用すると、サーバと単一のクライアントとの間のデータ転送を同期させることができます。ただし、この方法にはいくつか欠点があります。まず、クライアントが1つしか存在できないことです。複数のクライアントがあると互いに競合する恐れがあります。たとえば、新しいデータが届いたことをあるクライアントが通知する前に、他のクライアントがデータを受信して通知する可能性があります。より複雑なDDEダイアグラムを作成してこの問題に対処することもできますが、この方法では即時の対応が困難になります。

通信を同期させるこの方法のもう一つの問題は、集録速度がデータ転送速度によって制約されてしまうことです。この問題に対処するには、集録と送信を別のループに分解することです。集録側はデータを待ち行列に登録し、送信側はループでこのデータを送信します。この方法は、TCPサーバの例でサーバが複数の接続を処理する方法と似ています。

アプリケーションの中で、確実にデータ転送を同期させる必要がある場合は、TCP/IPを使用した方が良いでしょう。TCP/IPには待ち行列処理やデータ転送に関する通知機能があり、ドライバレベルでの複数の接続がサポートされているからです。

ネットワークによるDDE

DDEを使用すると、同じコンピュータ上のアプリケーションやネットワーク経由で、別のコンピュータ上のアプリケーションと通信することができます。ネットワークDDEを使用するには、Windows for Workgroups 3.1以上、Windows 95、またはWindows NTが動作している必要があります。標

準バージョンのWindows 3.1では、ネットワークによるDDEはサポートされていません。

Windows for Workgroupsが動作している各コンピュータにはネットワークコンピュータ名が付いています。この名前の構成には、ネットワークコントロールパネルを使用します。

ネットワーク経由で通信を行う場合は、サービスとトピックの文字列が変わります。サービス名には、ユーザがネットワークによるDDEを使用したいことを示すと共に、通信先のコンピュータの名前が含まれるようになります。サービス名の形式は次のようになります。

```
¥¥computer-name¥ndde$
```

トピックには任意の名前を付けることができます。次にSYSTEM.INIファイルを編集して、このトピック名を、リモートコンピュータで使用される実際のサービスとトピックに対応させます。また、この構成にはネットワーク接続を構成するパラメータも含まれます。この部分は、たとえば次のようになります。

```
[DDE Shares]

topicname = appname, realtopic, ,31,,0,,0,0,0
```

[topicname] はクライアントVIがトピック用に使用する名前であり、[Appname]はリモートアプリケーションの名前です。ネットワークによるDDEを使用する場合は、この名前がサービス名と同じでなければなりません。[Realtopic]は、リモートコンピュータで使用するトピックです。その他のパラメータは、DDEの動作方法を構成します。パラメータは、上記の例の通りに使用してください。これらのパラメータの意味は、Microsoftの資料には示されておりません。

たとえば、LabVIEWが動作している2台のコンピュータで、ネットワーク接続されたDDE を使用して通信する場合、サーバは、サービス名にLabVIEWを、トピック名に[labdata]などの名前を使用する必要があります。

サーバコンピュータの名前が[Lab]で、クライアントが¥¥Lab¥ndde$というサービス名を使用して通信を開こうとしている場合を考えてみます。クライアントは、トピック名に [remotelab] という名前を使用することができます。

これが有効になるようにするには、サーバコンピュータのSYSTEM.INIファイルを編集して [DDEShares] の部分に次の行を含める必要があります。

```
remotelab=LabVIEW,labdata,,31,,0,,0,0,0
```

Windows NT の場合は、WINNT/system 32 ディレクトリに入っている ddeshare.exe を起動します。**共有→DDE共有...** を選択し、さらに**DDE共有の追加...** を選択してサービス名とトピック名をサーバに登録します。

NetDDEを使用する

NetDDEは、Windows for WorkGroups 3.11、Windows 95、Windows NTに組み込まれています。WonderWare のアドオンパッケージがあれば、Windows 3.1 でも NetDDE を使用できます。Windows 3.1 で WonderWare パッケージを使用する場合は、WonderWare 付属の資料で NetDDE の使用法を参照してください。

Windows for WorkGroups、Windows 95、またはWindows NT を使用する場合は、以下の説明に従ってください。

サーバマシン

Windows for Workgroupsの場合

サーバ（DDEコマンドを受け取る側のアプリケーション）の system.ini ファイルの[DDE Shares]の部分に、次の行を追加します。

```
lvdemo = service_name,topic_name,,31,,0,,0,0,0
```

ただし

lvdemo の部分には、任意の名前を使用できます。

service_name は通常、excel などのアプリケーション名です。

topic_name は通常、sheet1 などの特定のファイル名です。

これ以外のカンマや数字は、上記の通りに入力してください。

Windows 95の場合

☞ **注**　**NetDDE は、Windows 95 により自動的に起動されませんので、¥WINDOWS¥NETDDE.EXE というプログラムを実行する必要があります（このプログラムをスタートアップフォルダに追加しておけば必ず起動されます）。**

Windows 95 上でNetDDEサーバをセットアップするには、

- ¥WINDOWS¥REGEDIT.EXE を実行します。
- ツリー表示の中の、マイコンピュータ ¥HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥NetDDE¥DDE Shares というフォルダを開きます。
- **編集→新規作成→キー**を選択して新しいDDESharesを作成し[lvdemo]という名前を付けます。

- [lvdemo]というキーが選択された状態で、次のように共有に必要な値を追加します（後で参照できるよう、これらのキーはCHAT$ shareからそのままコピーされますが、REGEDITではキーや値の切り取り、コピー、貼り付けを行えません）。新しい値を追加するには**編集→新規作成**を使用します。キーを作成する場合は、（標準）というデフォルト値と（値の設定なし）という値があります。これらの値はそのままにしておき、以下の値を追加します。

表 23-1 デフォルトの代わりに追加する値

値のタイプ	名前	値
バイナリ	Additional item count	00 00 00 00
文字列	Application	service_name
文字列	Item	service_name
文字列	Password1	service_name
文字列	Password2	service_name
バイナリ	Permissions1	1f 00 00 00
バイナリ	Permissions2	00 00 00 00
文字列	Topic	topic_name

- REGEDITを閉じます。
- マシンを再起動します（変更内容を有効にするにはNetDDEを再起動する必要があります）。

Windows NTの場合

Winnt¥system32ディレクトリに入っているddeShare.exeを起動します。**共有→DDE共有→DDE共有の追加...**を選択し、サービス名とトピック名をサーバに登録します。

クライアントマシン

クライアントマシン（DDE会話を開始する側のアプリケーション）では構成を変更する必要はありません。

DDE Open Conversation.viに対しては以下の入力を使用します。

サービス：¥¥machine_name¥ndde$

トピック：lvdemo

ただし、

machine_nameはサーバマシンの名前を指定します。

lvdemoは、サーバの[DDE Shares]の部分で指定された名前に一致します。

examples¥network¥ddeexamp.llbの中にあるChart Cliant.viとChart server.viの例を考えてみましょう。NetDDEを使用する2台のコンピュータ間でこれらのVIを使用して情報を渡すためには、次の指示に従ってください。

サーバマシン：

1. フロントパネル値は変更しません。
2. サーバマシンのsystem.iniファイルの[DDE Shares]の部分に次の行を追加します。

   ```
   lvdemo = TestServer,Chart,,31,,0,,0,0,0
   ```

クライアントマシン：

フロントパネルの制御器を次のように設定します。

サービス = ¥¥machine_name¥ndde$

トピック = lvdemo

項目 = Random

24

AppleEvent

この章では、Macintosh のアプリケーション間でのデータ交換を可能にする、Macintosh 専用のアプリケーション間通信（interapplication communication: IAC）である AppleEvent について説明します。

AppleEvent

AppleEvent は、アプリケーション間の相互通信を可能にする Macintosh 専用のプロトコルです。DDE と同様、アプリケーションはメッセージを使用してプロセスを要求したり他のアプリケーションからの情報を返します。アプリケーションは、アプリケーションそれ自体のほか、同じコンピュータ上のアプリケーション、またはネットワーク上の他のコンピュータで動作中のアプリケーションのいずれにもメッセージを送ることができます。AppleEvent を使用すると、開くや印刷などのコマンドを他のアプリケーションに送ったり、スプレッドシート情報などのデータを要求することができます。

LabVIEW には、ほとんどのアプリケーションに共通ないくつかのコマンドを送るための VI 群が用意されています。これらの VI は使い易いため、AppleEvent の動作に関する詳細知識がなくても使用できます。LabVIEW アプリケーションでの AppleEvent の使用方法のいくつかを、次に示します。

- LabVIEW にコマンドを送って、処理を実行するよう他のアプリケーション（ネットワーク接続された他のコンピュータ上のアプリケーションでもよい）に指示することができます。たとえば、LabVIEW はスプレッドシートプログラムにグラフを作成するように指示することができます。詳細は、本章の「AppleEvent を送る」の項を参照してください。
- AppleScript プログラムをフロントエンドとして使用し、特定の VI を実行するよう LabVIEW に指示することができます。
- 特定の処理を実行するように命令を送ることで、ネットワーク接続された他のマシン上の LabVIEW アプリケーションと通信したり、このアプリケーションを制御することができます。詳細は、本章の「AppleEvent を送る」の項を参照してください。
- LabVIEW にメッセージを送り、特定の VI のロード、実行、アンロードなどを LabVIEW 自身に命令することができます。たとえば、メモリ条件が厳しい大規模なアプリケーションの場合に、サブ VI 呼び出しを

ユーティリティ VI（AESend Open,Run,Close VI）に交換し、VIのロード、実行、アンロードを任意に行うことができます。

これらのVIは、下位AESend VIを使用してAppleEventを送ります。Apple社では、AppleEvent通信の標準化に役立つ膨大な数のメッセージのための**用語**を定義しています。この用語集に含まれる**語**を組み合わせると、複雑なメッセージを作成できます。このVIを使用すると、他のアプリケーションに任意のAppleEventを送ることができますが、このレベルでAppleEventを作成して送るのは複雑で、AppleEvent についての詳細な理解が必要です。『Inside Macintosh』と『AppleEvent Registry』を参照してください。

AppleEventを送る

関数パレットの**通信**サブパレットには、AppleEvent を送るための VI が用意されています。これらの VI を使用すると、AppleEvent 用のターゲットアプリケーションを選択したり、AppleEventを作成したり、ターゲットアプリケーションにAppleEventを送ることができます。

通信サブパレットのAppleEvent VIsパレットには、特定のAppleEventメッセージを送るVIが含まれています。これらのVIを使用すると、いくつかの標準AppleEvent（Open Document、Print Document、Close Application）、およびLabVIEW のすべてのカスタム AppleEvent を送ることができます。これらの上位 VI では、AppleEvent のプログラミングに関する詳細知識はほとんど必要ありません。これらのダイアグラムは、AppleEventの作成や送信の良い例となります。

AppleEventに対応するVIがLabVIEWに用意されていない場合は、下位のAESend VIを使用してそのAppleEventを送ることができます。**通信**サブパレットのAppleEvent VIsパレットには、AppleEventの作成に役立つVIも用意されていますが、このレベルで AppleEvent を作成して送るのは複雑で、AppleEventについての詳細な理解が必要です。『Inside Macintosh』と『AppleEvent Registry』を参照してください。

クライアントサーバモデル

AppleEvent VIを使用して、サーバとして動作するLabVIEWダイアグラムを作成することはできません。これらの VI は、他のアプリケーションにメッセージを送るために使用されます。ダイアグラムベースのサーバ機能が必要な場合は、TCPかPPCを使用する必要があります。

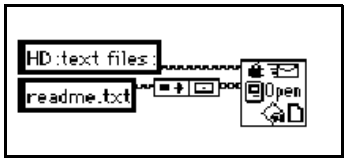
LabVIEW 自体は、一連の AppleEvent を認識して応答するという意味では AppleEvent サーバとして機能します。特に AppleEvent を使用すると、VIを開き、印刷し、実行し、閉じるという操作をLabVIEW に指示することができます。特定のVIが実行中であるかどうかをLabVIEWに問い合わせたり、LabVIEW を終了するよう指示することもできます。

これらのサーバ機能を使用すると、VIの実行をLabVIEWアプリケーションに指示したり、LabVIEWをリモート制御したりできます。また、特定のVIをロードするようにLabVIEW自身にメッセージを送ることもできます。たとえば、メモリが制約される大規模なアプリケーションの場合に、サブVIの呼び出しを、AESend Open, Run, Close VIに交換し、必要に応じてVIをロードして実行することができます。ただし、このような方法でVIを実行すると、**ファイル→開く**...を選択した場合と全く同様にフロントパネルが開くことに注意してください。

AppleEventクライアントの例

他のアプリケーションを起動する

アプリケーションに対してメッセージを送るには、そのアプリケーションが動作中でなければなりません。AESend Finder Open VIを使用すると、他のアプリケーションを起動できます。このVIはFinderに対してメッセージを送ります。Finder自体は、限られた数のAppleEventしか認識できないアプリケーションです。次の簡単な例で、AppleEventを使用して特定のテキストファイルを含むTeach Textを起動する方法を示します。



アプリケーションがリモートコンピュータ上にある場合は、そのコンピュータのロケーションを指定する必要があります。AESend Finder Open VIへの入力を使用すると、通信したいコンピュータのネットワークゾーンとサーバ名を指定することができます。前のアプリケーションのようにネットワークゾーンとサーバ名を指定しない場合は、デフォルトで現在のコンピュータのものに設定されます。

別のコンピュータにメッセージを送る場合は、そのコンピュータにログインするよう促すプロンプトが自動的に表示されることに注意してください。この機能はオペレーティングシステムに組み込まれていますので、このプロンプトを避ける方法はありません。無人コンピュータシステムでアプリケーションを動作させたい場合には、この機能が問題になる可能性があります。

他のアプリケーションにイベントを送る

アプリケーションが動作状態になれば、他のAppleEventを使用してそのアプリケーションにメッセージを送ることができます。すべてのアプリケーションがAppleEventをサポートするわけではなく、また、AppleEventをサポートするアプリケーションが、発行されているすべてのAppleEventをサポートするわけでもありません。各アプリケーションでサポートされるAppleEventについては、そのアプリケーションに付属の資料を参照してください。

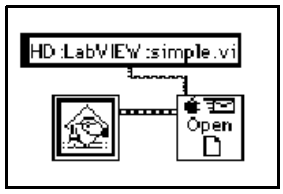
AppleEventを認識できるアプリケーションの場合は、そのアプリケーションに対応するターゲットIDを設定してAppleEvent VIを呼び出します。ターゲットIDは、ネットワーク上のターゲットのロケーション（ゾーン、サーバ、およびサポートするアプリケーション）を記述したクラスタです。

ユーザはLabVIEWが提供するVIを使用してターゲットIDを生成できますので、このクラスタの正確な構造について気にする必要はありません。

ターゲットIDを作成する方法は2つあります。Get Target ID VIを使用すると、アプリケーション名とネットワーク上のロケーションをもとにしてターゲットIDをプログラム的に作成できます。また、PPC Browser VIを使用して、AppleEventに対応しているネットワークアプリケーションをダイアログボックスに表示することもできます。ユーザは、このリストから対話式に選択することでターゲットIDを作成します。

また、PPC Browser VIを使用すると別のアプリケーションがAppleEventを使用するかしないかを調べることができます。このVIを実行し、アプリケーションが実行中のコンピュータを選択すると、AppleEventに対応しているアプリケーションのリストがダイアログボックスに表示されます。

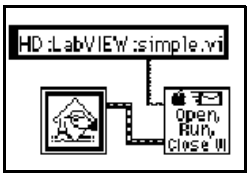
次のダイアグラムの場合、LabVIEWはネットワーク上のAppleEvent対応のアプリケーションを対話式に選択して、文書を開くようにこのアプリケーションに指示します。この場合LabVIEWはアプリケーションに対して、VIを開くように指示しています。



VIを任意にロードして実行する

AESend Open,Run,Close VIは、VIを実行するようにLabVIEWに要求するメッセージを送ります。このVIはまずOpen Documentメッセージを送り、LabVIEWはVIを開きます。続いてOpen Run Close VIはLabVIEW Run VIメッセージを送り、LabVIEWは指定されたVIを実行します。次に、Open Run CloseはVI Active?メッセージを送り、LabVIEWは、VIが実行されなくなるまで、指定されたVIのステータスを返します。最後にVIはClose VIメッセージを送ります。

ターゲットとなるLabVIEWが他のコンピュータにある場合は、次のダイアグラムを使用してVIをロードし実行できます。現在のLabVIEWにVIを送る場合は、PPC Browser VIは必要ありません。



25

プログラム間通信

この章では、Macintosh アプリケーションがデータブロックを送受信するための低レベルの Apple アプリケーション間通信（Apple interapplication communication: IAC）であるプログラム間通信（program-to-program communication: PPC）について説明します。

PPC の概要

プログラム間通信（PPC）は、アプリケーション間でデータブロックを転送するための Macintosh 専用プロトコルです。PPC を使用すると、クライアントやサーバとして機能する VI を作成することができます。PPC は、System 7.*x* が動作するすべての Macintosh でサポートされますが、通常ほとんどの Macintosh アプリケーションでは使用されません。大抵の Macintosh アプリケーションでは、PPC ではなくアプリケーション間でコマンドを送るための上位プロトコルである AppleEvent を使用して通信を行います。

PPC は AppleEvent ほど一般的にはサポートされていませんが、PPC にもいくつかの利点があります。PPC は低いレベルに位置するため、AppleEvent よりも性能的には優れています。また LabVIEW では、PPC を使用してクライアントやサーバとして機能する VI を作成することができます。ただし、AppleEvent サーバとして機能するダイアグラムを作成することはできません。

LabVIEW の VI は、PPC を使用して、同じコンピュータ上のアプリケーション間、またはネットワーク上の異なるコンピュータのアプリケーション間で大量の情報を送受信できます。2 つのアプリケーションが PPC を使用してデータ交換するためには、両者が動作中で、情報を送受信する準備ができていることが必要です。

サーバおよびクライアントのいずれのアプリケーションという点でも、PPC と TCP は構造的に似ています。PPC でリモートアプリケーションを指定する方法は、TCP での方法と異なりますが、この他の点では、この 2 つのプロトコルは性能や機能が似ています。どちらのプロトコルも待ち行列処理をサポートし、信頼性の高いデータ転送を行います。またどちらのプロトコルも、複数の接続を開いた状態で使用することができます。

TCPとPPCのどちらにするかを決定する場合は、VIを動作させる予定のプラットフォームや、通信の相手となるプラットフォームについてよく考慮してください。自分のアプリケーションがMacintosh専用である場合は、PPCがオペレーティングシステムに組み込まれていることから、PPCを選択するのが良いでしょう。TCPは、バージョン7.5以降のMacintoshオペレーティングシステムに組み込まれています。それ以外のシステムでTCPを使用するには、Apple社からTCP/IPドライバを別途購入する必要があります。個別のドライバを購入するのに問題がないのであれば、PPCよりもTCPインタフェースの方が簡単ですので、TCPを使用した方が良いでしょう。PPCでは、アドレスを記述するのにかなり複雑なデータストラクチャを使用します。

作成するアプリケーションが、他のプラットフォームと通信したり、他のプラットフォームでも実行可能にする必要がある場合は、TCP/IPを使用してください。

ポート、ターゲットID、およびセッション

PPCを使用して通信を行うには、クライアントとサーバの両方で、以降の通信に使用するポートを開かなければなりません。Open Port VIは、VIに含まれているクラスタを使用してポートを開きます。クラスタには、ポートに使用する名前などが含まれています。ポートは、アプリケーションが提供するさまざまなサービスを識別するために使用されます。各アプリケーションは、複数のポートを同時に開くことができます。

各ポートは、同時に複数のセッションまたは通信をサポートすることができます。セッションを開くため、クライアントはサーバのロケーションを示すターゲットIDを使用します。PPCは、AppleEvent VIで使用されるのと同じタイプのターゲットIDを使用します。リモートアプリケーション用のターゲットIDを生成するには、PPC BrowserまたはGet Target ID VIを使用します。

サーバは、クライアントがPPC Informセッション VIを使用してセッションを開こうとするのを待ちます。サーバは、PPC Accept Session VIを使用してセッションを受け付けたり拒否することができます。クライアントは、PPC Start Session VI を使用してサーバとのセッションを開くよう試みます。

セッションが起動されると、PPC Read VIやPPC Write VIを使用してデータを転送することができます。セッションを閉じるにはPPC End Session VIを、ポートを閉じるにはPPC Close Port VIを使用します。

PPCクライアントの例

以下の説明では、PPC を使用して一般的なクライアントモデルの各コンポーネントを満たす方法を示します。

サーバへの接続を開くには、PPC Open Port VI と PPC start Session VI を使用します。このためには、ターゲット ID を指定する必要があります。サーバのターゲット ID を取得するには、PPC Browser VI か Get Target ID VI を使用します。最終的な結果として、サーバとの通信に使用されるポート refnum とセッション refnum が得られます。

サーバでコマンドを実行するには、PPC Write VI を使用してサーバにコマンドを送ります。次に、PPC Read VI を使用してサーバから結果を読み込みます。PPC Read VI を使用する場合は、読み込む文字数を指定する必要があります。TCP の場合と同様、応答の長さが変動するためにこの方法がうまく行かない場合があります。サーバ側でも、コマンドの長さが変動するために同様の問題が起こる可能性があります。

コマンドのサイズが異なる問題には、以下の方法で対処することができます。TCP の場合にも、これらの方法を使用することができます。

- コマンドや結果の前に、コマンドや結果のサイズを指定する固定サイズパラメータを付ける。この場合は、サイズパラメータを読み込んでから、サイズで指定された数の文字を読み込みます。このオプションは効果的で、柔軟性もあります。
- 各コマンドや結果のサイズを固定する。このサイズよりコマンドが小さい場合は、固定サイズに合わせるためのダミー要素を小さくコマンドに追加します。
- 各コマンドや結果の後に、特定の終了文字を付ける。この場合は、終了文字を読み込むまで小さいデータを分けて読み込む必要があります。

サーバへの接続を閉じるには、PPC End Session VI と PPC Close port VI を使用します。

PPC サーバの例

以下の説明では、PPC を使用して一般的なサーバモデルの各コンポーネントを満たす方法を示します。

初期化段階で、PPC Open Port を使用して通信ポートを開きます。

PPC Inform Session VI を使用して接続を待ちます。PPC では、入って来る接続を自動的に受け付けることができる他、PPC Accept Session VI を使用してセッションを受け付けるか拒否するかを選択することもできます。このようにセッションを待ってからセッションを承認する処理を行うことで、接続を選別することができます。

接続が確立されると、そのセッションを読み込んでコマンドを取り出すことができます。「PPC クライアントの例」の項で説明したように、コマンドの形式を決定する必要があります。コマンドの前に長さフィールドがある場合は、最初に長さフィールドを読み込んでから、長さフィールドで指定された数のデータを読み込みます。

この処理はローカルコンピュータ上で実行されますので、コマンドの実行はプロトコルからは独立していなければなりません。実行が完了したら、結果を次のステージに渡し、そこで結果をクライアントに送信します。

結果を返すにはPPC Write VI を使用します。「PPC クライアントの例」の項で説明したように、データはクライアントが受信可能な形になっていなければなりません。

PPC End Session VI を使用して接続を閉じます。

サーバの処理が終了したら、PPC Close Port VI を使用して、初期化段階で開いたポートを閉じます。

複数の接続を使用するPPCサーバ

PPCでは複数のセッションや複数のポートに容易に対応できます。前項で説明した、サーバの各コンポーネントを実装する方法は、複数の接続を使用するサーバにも使用できます。図25-1に、PPC VIの使用順序を示します。

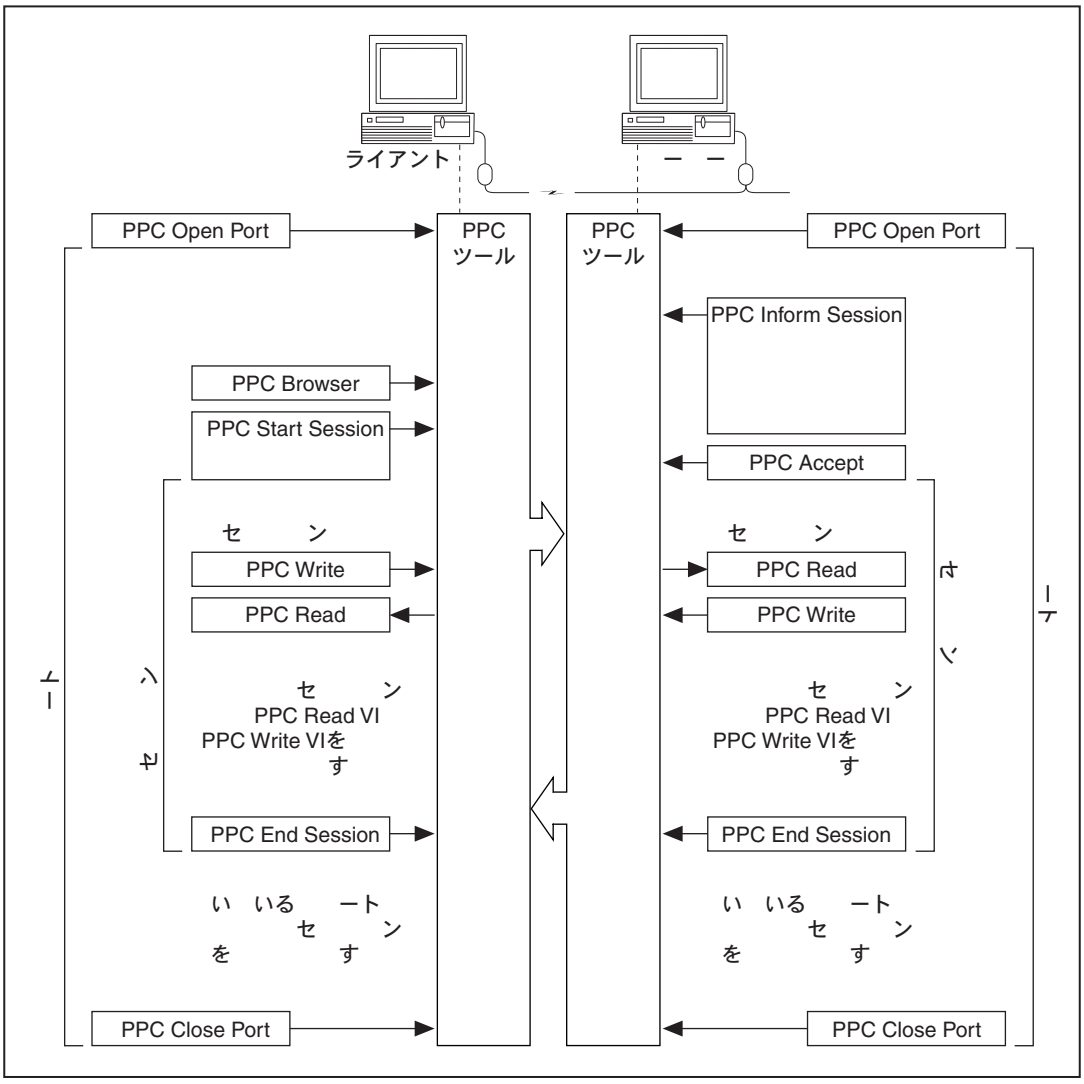


図 25-1 PPC VIの実行順序（Apple Computer, Inc.の許可により使用）

第V部

上級Gプログラミング

第V部では、VIのカスタマイズ、プログラムによるフロントパネルオブジェクトの構成、および複雑なアプリケーションの設計について説明します。

「第V部 上級Gプログラミング」は、以下の章から構成されています。

- 「第26章 VIをカスタマイズする」では、VIをカスタマイズする方法を示します。また、**VI設定...**や**サブVIノード 設定...**オプションを使用して、実行時のVIの外観や動作をカスタマイズする方法を理解していただくための作業も用意されています。
- 「第27章 フロントパネルオブジェクトの属性」では、制御器や表示器の外観や機能特性を制御する特殊なブロックダイアグラムノードである、属性ノードというオブジェクトについて説明します。
- 「第28章 プログラム設計」では、プログラム作成時に使用する技法を説明し、プログラミングスタイルについてのガイドラインを示します。
- 「第29章 次に学習すべき内容」では、アプリケーションを正しく作成するために使用可能なリソースについて説明します。

☞ **注** **Windows 3.1のユーザは、第I部で作成したVIをVIライブラリに保存する必要があります。VIライブラリでは、8文字以上の長さの名前を付けることができます。第V部で行う作業に必要なVIは、VIライブラリの`LabVIEW¥Activity¥Activitiy.llb`にあります。VIライブラリに関する詳細は、『Gプログラミングリファレンスマニュアル』の「第2章 VIを編集する」の「VIを保存する」の項を参照してください。**

フロントパネルのオブジェクトをプログラムにより構成できるだけでなく、VIやLabVIEW自体をプログラムにより制御することもできます。たとえば、VIの外観の変更、VIの実行時の動作の変更、LabVIEWの印刷設定の変更、メモリにロードされるすべてのVIの決定などを行えます。また、**環境設定**や**VI設定...**で対話式に設定可能なオプションも、プログラムにより設定することができます。**環境設定**に用意されているオプションは、LabVIEWのすべてのVIに影響を及ぼすことから、アプリケーション設定とみなされます。**VI設定...**のオプションは、1つのVIと、そのVIをサブVIとして使用するすべての場合だけに影響を及ぼすことから、VIに対する設定ということになります。VIやLabVIEWの構成を変更できるだけで

なく、処理やメソッドを実行することもできます。たとえば、フロントパネルのすべてのオブジェクトをデフォルト値に初期化し直すことができます。

プログラムによる制御のもう一つの例は、VI を動的にロードして呼び出す操作です。多くのサブ VI を含む大規模なアプリケーションの場合、最上位 VI をロードするときにすべてのサブ VI がメモリにロードされます。すべてのサブ VI をメモリにロードする方法は、大規模なアプリケーションの場合には実際的ではありません。このような方法ではなく、Call By Reference ノードを使用して動的にサブ VI をロードして呼び出すことができます。

VI と LabVIEW をプログラムにより構成したり制御する方法についての詳細は、『G プログラミングリファレンスマニュアル』の「第 21 章　VI サーバ」、および『LabVIEW オンラインリファレンス』を参照してください。

ローカルマシンで VI と LabVIEW を構成したり制御できるだけでなく、リモートマシン上の VI と LabVIEW も TCP/IP ネットワーク経由で構成したり制御することができます。詳細は、『G プログラミングリファレンスマニュアル』の「第 7 章　環境をカスタマイズする」を参照してください。また、別の ActiveX アプリケーションにより VI と LabVIEW を構成したり制御することもできます。LabVIEW における ActiveX のサポートに関しては、本書の「第 22 章　ActiveX のサポート」を参照してください。

26

VIをカスタマイズする

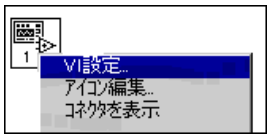
この章ではVIをカスタマイズする方法を説明します。また、次のタスクの実行方法を示すための作業も用意されています。

- **VI設定...**オプションを使用する
- **サブVIノード設定...**オプションを使用する

カスタムVIの例については、`Examples¥General¥viopts.llb`を参照してください。

VIのカスタマイズ方法

VIの実行を構成する方法はいくつかあります。これらのオプションにアクセスするには、フロントパネルの右上隅のアイコンペーンをポップアップして**VI設定...**を選択します。



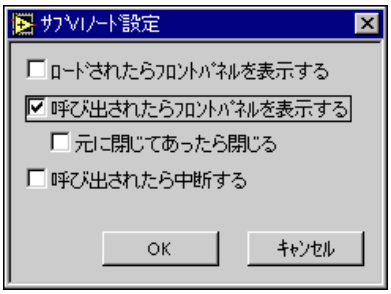
VI設定ダイアログボックスが現れ、VIの実行、パネルの外観、文書の動作時のメニューバーの外観などのセットアップオプションが表示されます。これらのオプションの使用方法は、作業26-1で学習します。詳細は、『Gプログラミングリファレンスマニュアル』の「第6章 VIおよびサブVIをセットアップする」を参照してください。

ウィンドウオプションを設定する

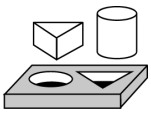
実行時のVIの外観を制御するには**ウィンドウオプション**を使用します。**実行オプション**から**ウィンドウオプション**に切り換えるには、メニューバーの下向きの矢印をクリックします。

サブVIノードの設定

サブVIの実行内容も構成することができます。構成オプションを使用するには、呼び出し側のVIのブロックダイアグラム内のサブVIアイコンをポップアップして**サブ VI ノード設定...**を選択します。サブ VI ノード設定ダイアログボックスを次に示します。



☞ **注**　VI設定ダイアログボックスからオプションを選択すると、そのVIに対しては常にそのオプションが適用されます。サブ VI ノード設定ダイアログボックスからオプションを選択した場合は、その特定のノードに対してのみそのオプションが適用されます。



作業 26-1.　サブVI用の設定オプションを使用する

ここでは、オペレータに情報を入力するように促すプロンプトを表示するVIを作成するのが目的です。

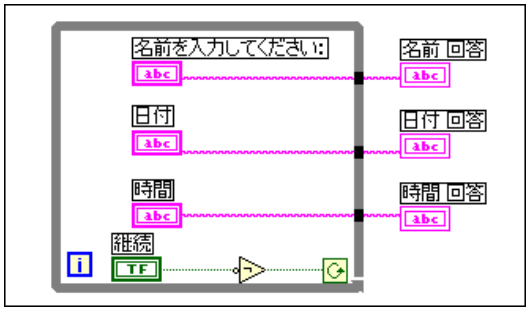
実行時にユーザから情報を得るためのダイアログボックスを起動するVIを作成します。ユーザが情報を入力してボタンをクリックすると、ダイアログボックスが消えます。

フロントパネル

1. 新しいフロントパネルを開き、次の図のように文字列制御器とボタンを追加します。

ブロックダイアグラム

2. 次の図のようなブロックダイアグラムを作成します。

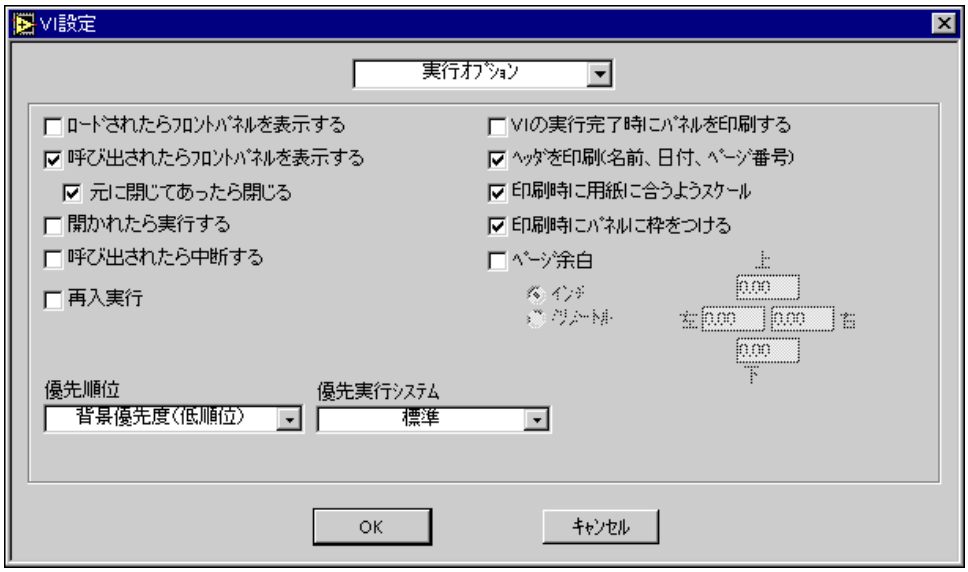


3. 左の図のようなVI用のアイコンを作成します。アイコンエディタにアクセスするには、フロントパネルのアイコンペーンをポップアップしてアイコンを編集を選択します。
4. アイコンペーンをポップアップしてコネクタを表示を選択し、コネクタペーンに切り換えます。

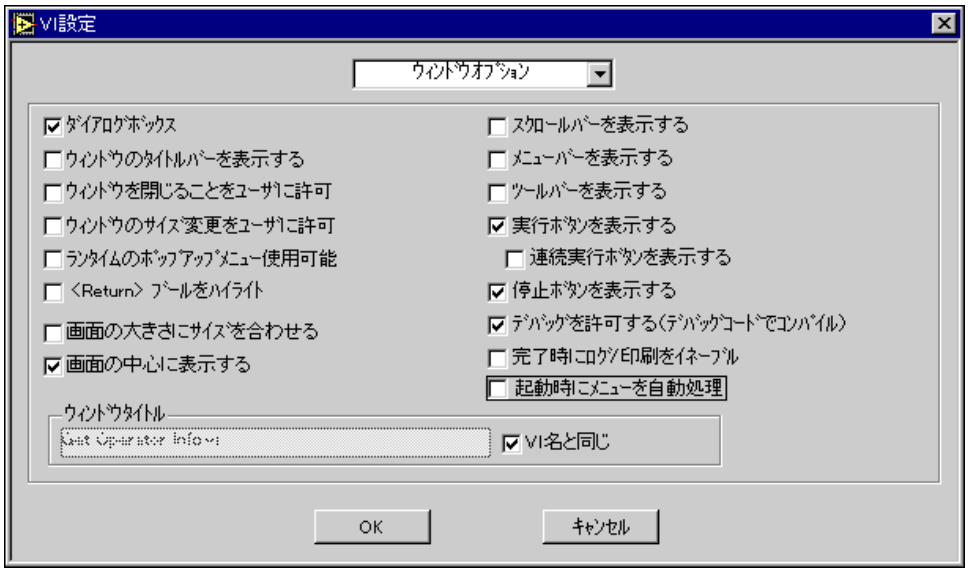
5. コネクタを作成します。デフォルトのコネクタペーンは左の図のコネクタペーンと異なることに注意してください。正しいコネクタペーンを作成するには、コネクタのポップアップメニューからパターンを選択します。3つの入力と2つの出力が付いたパターンを選択し、左右逆転を選択します。続いて次の図のように、日付、時間各制御器をアイ

コンの左側の2つのコネクタに接続し、**名前回答**、**日付回答**、**時間回答**の各表示器をアイコンの右側の3つのコネクタに接続します。コネクタを作成したら、アイコン表示に戻ります。

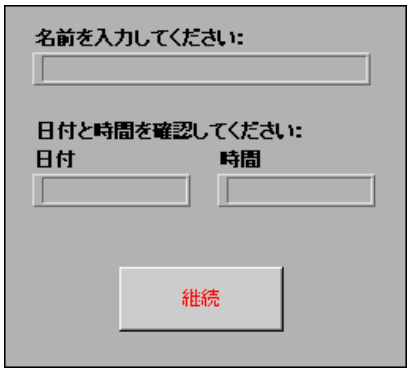
6. このVIを、`Get Operator Info.vi`という名前で`LabVIEW¥Activity`ディレクトリに保存します。
7. 次に**VI設定**のオプションでVIをカスタマイズし、VIの外観をダイアログボックスのようにします。
   a. アイコンをポップアップして**VI設定**を選択します。次の図のように**実行オプション**を構成します。



   b. **ウィンドウオプション**を選択し、さらに次の図のように選択します。



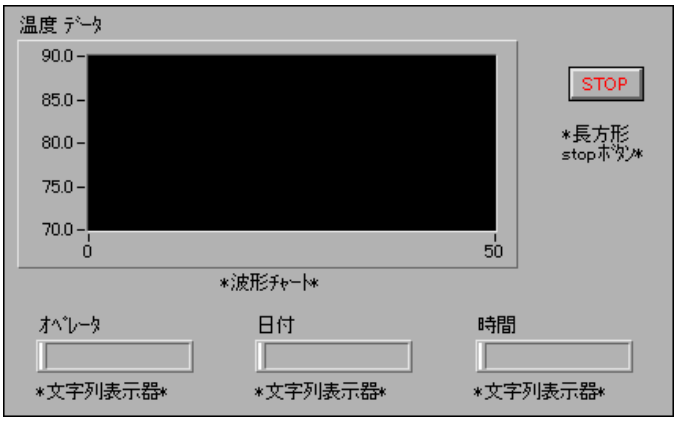
8. **VI設定**オプションを選択したら、次の図のように、3つの文字列表示器が表示されないようフロントパネルのサイズを変更します。



9. VIを保存して閉じます。これで、このVIをサブVIとして使用できるようになりました。

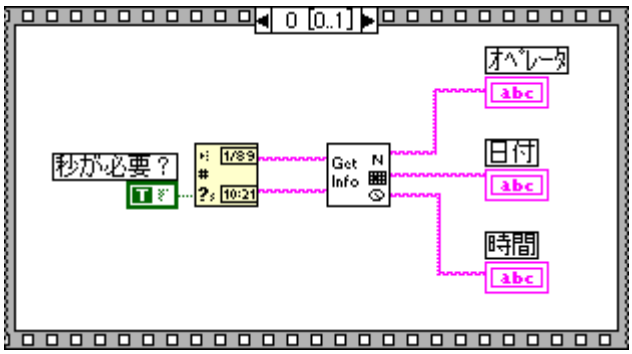
フロントパネル

10. 新しいフロントパネルを開きます。
11. 波形チャート（**制御器→グラフ**）をフロントパネル上に配置して、［温度データ］というラベルを付けます。
12. 上限値が[90.0]、下限値が[70.0]になるようにチャートの目盛りを変更します。チャートをポップアップして**表示→凡例**を選択し、凡例を非表示にします。さらに、もう一度チャートをポップアップして**表示→パレット**を選択し、パレットを非表示にします。
13. 次の図のように、フロントパネルの残りの部分を作成します。



ブロックダイアグラム

14. 次の図のように、シーケンスストラクチャを作成し、以下のオブジェクトをフレーム0に追加します。



Get Date/Time String関数（**関数→時間＆ダイアログ**）— 現在の日付と時刻を出力します。

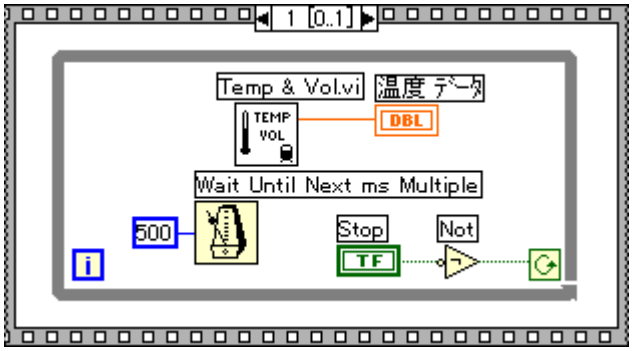
Get Operator Info VI（**関数→VIを選択...**の後`LabVIEW¥Activity`ディレクトリから選択）— 対応するフロントパネルを開き、名前、日付、時刻を入力するようにユーザに促すプロンプトを表示します。

ブール定数（**関数→ブール**）— 入力された日付と時刻の文字列がTRUEであるかどうかを制御します。このオプションをTRUEに設定するには、操作ツールでこの定数をクリックします。

15. シーケンスストラクチャをポップアップし、ポップアップメニューから**後にフレームを追加**を選択します。

16. シーケンスストラクチャのフレーム1の中にWhileループを配置します。

17. 次の図のようにオブジェクトを追加します。



Temp & Vol VI（**関数→VIを選択...**の後`LabVIEW¥Activity`ディレクトリから選択）— シミュレーションによる温度センサから温度測定値を1つ返します。

Wait Until Next ms Multiple関数（**関数→時間＆ダイアログ**）— Whileループがミリ秒単位で実行されるようにします。

数値定数（**関数→数値**）— Wait Until Next ms Multiple関数をポップアップして**定数を作成**を選択し、自動的に数値定数を作成して配線することもできます。この数値定数はループの実行を500ミリ秒（0.5秒）遅らせます。

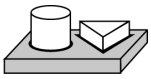
Not関数（**関数→ブール**）— ユーザがSTOPをクリックするまでWhileループが繰り返し実行されるように、STOPボタンの値を反転します。

18. このVIを`Pop-up Panel Demo.vi`という名前で`LabVIEW¥Activity`ディレクトリに保存します。

19. VIを実行します。Get Operator Info VIのフロントパネルが開き、名前、日付、時刻を入力するよう促すプロンプトが表示されます。**継続**ボタンをクリックして呼び出し側のVIに戻ります。この後、ユーザがSTOPボタンをクリックするまで温度データが集録されます。

☞ **注**　Get Operator Info VIのフロントパネルが開くのは、VI設定ダイアログボックスでオプションを選択したからです。Pop-Up Panel Demo VIのブロックダイアグラムからサブVIのフロントパネルを開くことは避けてください。

20. すべてのウィンドウを閉じます。



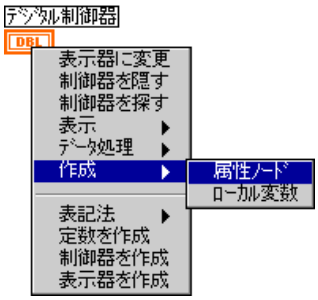
これで作業 26-1 は完了です。

27

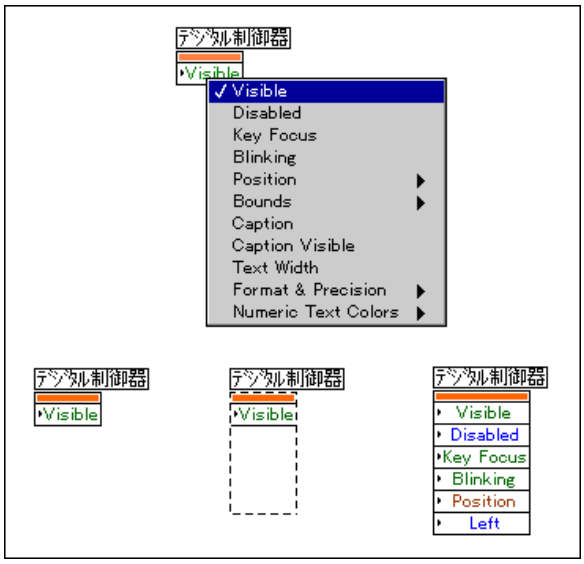
フロントパネルオブジェクトの属性

この章では、制御器や表示器の機能特性や外観を制御する特殊なブロックダイアグラムノードである、属性ノードと呼ばれるオブジェクトについて説明します。

属性ノードを使用すると、表示色、表示／非表示、位置、点滅を始めとするさまざまな属性を設定できます。属性ノードを作成するには、次の図のように、ブロックダイアグラム内の端子、またはフロントパネルオブジェクトのポップアップメニューから**作成→属性ノード**を選択します。



最初、属性ノードには 1 つの特性が表示されます。複数の特性が表示されるようにノードを拡大することもできます。ノードを拡大するには、位置決めツールで属性ノードを選択します。ノードの右下隅付近にカーソルを移動し、カーソルがフレームに変わったら、ドラッグして希望する数の特性を作成します。さらに次の図のように操作ツールでノードをクリックしてポップアップメニューから新しい属性を選択すると、属性を変更することができます。



フロントパネルオブジェクトにはさまざまな属性があるため、ヘルプウィンドウを使用して、説明、データタイプ、属性の許容値などを確認してください。ヘルプウィンドウにアクセスするには、**ヘルプ→ヘルプを表示**を選択します。

LabVIEW のヘルプへのアクセス方法についての詳細は、『G プログラミングリファレンスマニュアル』の「第1章　G プログラミングの概要」の「ヘルプの呼び出し」の項を参照してください。

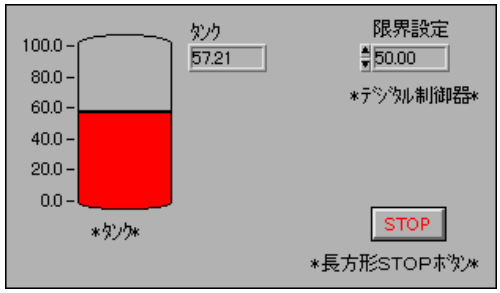
属性ノードを使用すると、特性を設定したり、属性をポップアップして**読み取りに変更**を選択することで属性の現在の状態を読み込むことができます。

作業 27-1. 属性ノードを使用する

ここでは、属性ノードを使用して上限条件を示すVIを作成することが目的です。タンク表示器の**Fill Color**属性を使用して、ランダムに生成されたタンクレベルがユーザの定義した限界を超えたかどうかを表示します。

フロントパネル

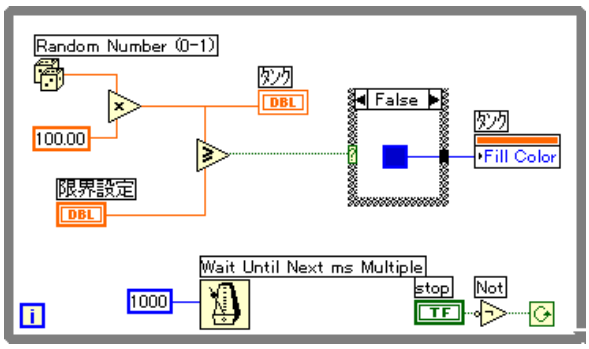
1. 次の図のように新しいフロントパネルを開いて作成します。



2. タンクの目盛りを[0～100.0]に直します。
3. **限界設定**のデフォルト値を[50.00]に設定します。

ブロックダイアグラム

4. 次のようなブロックダイアグラムを作成します。



Not関数（**関数→ブール**）— この練習問題で、Not関数はユーザがSTOPボタンをクリックまでWhileループが繰り返し実行されるように、STOPボタンの値を反転します（ボタンのデフォルトの状態はFALSEです）。

Random Number Generator（**関数→数値**）— フロントパネルのタンクの量を示す0と1の間の生データを発生します。この値に100をかけて0と100の間の値にします。

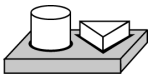
Greater or Equal?（**関数→比較**）— 生データを**範囲設定**入力値と比較します。この値が限界入力値以上の場合、値TRUEがCaseストラクチャに渡されます。

属性ノード（タンク端子をポップアップ）— タンク端子から**作成→属性ノード**を選択します。属性をポップアップして**項目選択→**Fill Colorを選択します。

カラーボックス定数（**関数→数値→その他の数値定数**）— この定数を配線して、TRUEケースのFill Colorに対する赤い色とFALSEケースの青い色を定義します。操作ツールで定数をクリックして色を選択します。

Wait Until Next ms Multiple（**関数→時間＆ダイアログ**）— 1秒ごとにループを実行するよう、定数1000を接続します。

5. VIを実行します。タンクのレベルは**限界設定**制御器と比較され、タンクの値が**限界設定**の値以上になると、タンクの色が赤に変わります。データが限界値より小さくなると、タンクの色が青に変わります。
6. このVIを、`Tank Limit.vi`という名前で`LabVIEW¥Activity`ディレクトリに保存します。



これで作業 27-1 は完了です。

28

プログラム設計

ここまでGプログラミングに関する多くのことを学習してきましたので、次に、その知識を利用して自分のアプリケーションを開発してみる必要があります。この章では、プログラムを作成する際に使用するいくつかの方法や、推奨するプログラミングスタイルを示します。

トップダウン方式の設計を使用する

大規模なプロジェクトを扱う場合は、**トップダウン方式の設計**を採用してください。Gでは、最終的なユーザインタフェースから始め、後から動作を設定していきますので、トップダウン式の設計には他のプログラミング言語より有利です。

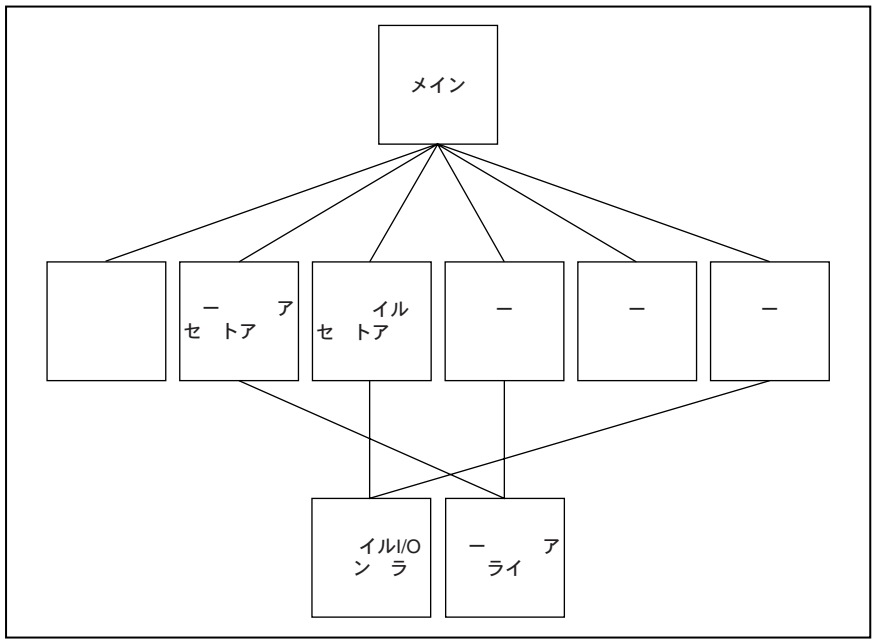
ユーザ条件のリストを作成する

ユーザが対話式に操作できるパネル、パネルの制御器と表示器の数とタイプ、リアルタイムでの解析とデータ表示の必要性などのリストを作成します。次に、アプリケーションを使用するユーザに見せるための（自分がユーザである場合は自分で操作するための）フロントパネルを試作します。機能や特徴を考察、検討します。このような対話による過程を通して、必要に応じユーザインタフェースの設計を変更していきます。仕様を確実に満足できるようにするため、初期の段階で低レベル動作についての検討が必要になる場合もあります。

VI階層を設計する

Gが強力なのは、VIが階層的な性質を持っているからです。VIを作成した後、このVIを上位VIのブロックダイアグラム内でサブVIとして使用できます。階層内の層の数は基本的には無制限です。

実現すべきタスクを、処理しやすい論理的な断片に分類します。次のフローチャートに示すように、どのようなデータ集録システムでも、さまざまな形の主要ブロックがいくつか考えられます。



これらのブロックの中には不要なものがあったり、また別のブロックが必要なこともあります。たとえば、アプリケーションによってはファイルI/O動作が含まれないものもあります。逆に、ユーザプロンプトを表示するブロックなど、別のブロックが必要な場合もあります。第一の目的は、プログラミングタスクを複数の扱いやすい上位ブロックに分割することです。

必要な上位ブロックが決まったら、それらの上位ブロックを使用するブロックダイアグラムを作成してみます。各ブロックごとに、新しい**スタブVI**（将来のサブVIを示す機能を持たないプロトタイプ）を作成します。このスタブVIに対応するアイコンと必要な入出力を含むフロントパネルを作成します。このVI用のブロックダイアグラムはまだ作成する必要はありませんが、最上位ブロックダイアグラムの一部としてこのスタブVIが必要かどうかを確認してください。

スタブVIを作成したら、各ブロックの機能や各ブロックで希望する結果を得る方法を、一般的な用語で理解するようにしてください。各ブロックで、それ以後のVIに必要な情報が生成されているかを確認します。生成されているようであれば、最上位ブロックダイアグラムの概略図の中に、VI間でデータを受け渡すためのワイヤが含まれているか確認します。

不必要なグローバル変数を使用するとVI間のデータの依存性がわからなくなりますので、このようなことは避けてください。VI間でグローバル変数に依存した情報転送を行っている場合は、システムが大きくなるほどデバッグが難しくなります。

プログラムを作成する

以上で、Gを使用してプログラムを作成する準備は完了です。

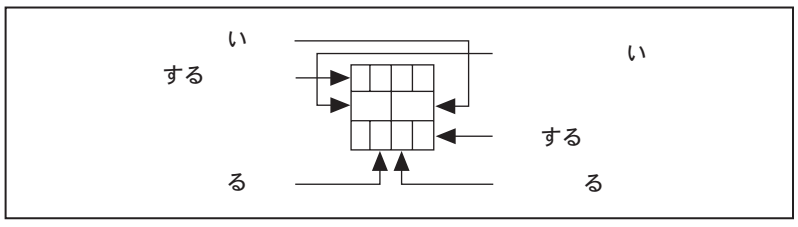
- 作業を論理的に分割できる場合やコードを再利用できる場合は、サブVIを作成してモジュール式にします。
- 一般的な問題と特殊な問題を解決します。
- サブVIを作成したらテストします。上位のテストルーチンが必要な場合もありますが、複数のVIの階層よりも単体の小さいモジュールの方が、容易にバグを発見できます。

サブVIの細部を考えている段階で、初期設計の不完全な部分が見つかる場合もあります。たとえば、1つのサブVIから別のサブVIに多くの情報転送が必要なことがわかる場合などです。このような場合はすぐに最上位部分の設計を再検討する必要があります。モジュール式のサブVIを使用した特定のタスクを実現する方法では、プログラムの再構成も容易に行えます。

コネクタペーンを使用して先に計画を立てる

後で別の入力や出力を追加する必要があると思われる場合は、追加端子を持つコネクタペーンパターンを選択します。これらの追加用端子は未接続の状態でかまいません。このような追加端子があると、後で別の入出力が必要なことがわかった場合でも、VIのコネクタペーンを変更しなくて済みます。このような柔軟性をもたせることによって、後で変更を行う場合、階層構造に与える影響を最小限に抑えることができます。

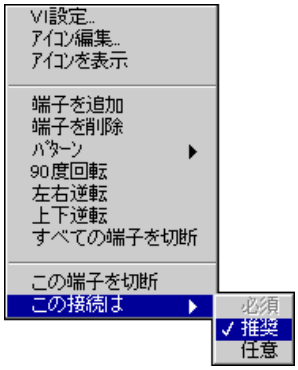
制御器や表示器をコネクタにリンクする場合は、入力が左に、出力が右になるように配置します。こうすることで、VIの配線パターンが複雑でわかりにくくなるのを避けることができます。



一緒に使用されることが多いサブVIのグループを作成する場合は、これらのサブVIで、同じ位置に共通の入力があるコネクタペーンを使用するように統一してください。こうしておくと、ヘルプウィンドウを使用しなくても、各入力を配置すべき位置を容易に思い出すことができます。別のサブVIへの入力として使用される出力を生成するサブVIを作成する場合は、入出力接続の位置を揃えてください。このようにすると、配線パターンが簡潔になります。

必要な入力を含むサブVI

フロントパネルでサブVIに必要な入力を編集するには、ウィンドウの右上のアイコンペーンをクリックして**コネクタを表示→この接続は**を選択し、サブメニューから**必須、推奨、任意**オプションのいずれかを選択します。サブメニューのオプションを次の図に示します。



フロントパネルのアイコンペーンに戻るには、コネクタペーンをポップアップして**アイコンを表示**を選択します。

望ましいダイアグラムスタイル

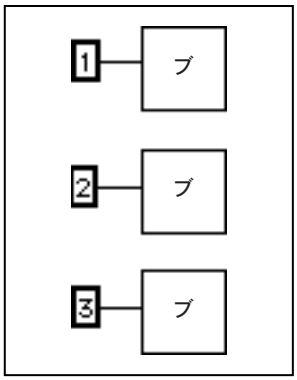
通常は、複数の画面にまたがるブロックダイアグラムを作成することは避けてください。ダイアグラムが大きすぎる場合は、ダイアグラムのコンポーネントの中に他のVIで使用できるものがないか、またダイアグラムの一部を1つの論理的コンポーネントとするのに適していないか判断します。そうであるならば、ダイアグラムをサブVIに分割することを検討します。

前もって慎重に計画するようにすれば、サブVIを使用して特定のタスクを実行するダイアグラムの設計は一層簡単になります。サブVIを使用すると、ダイアグラムの変更やデバッグに迅速に対処できます。プログラムが良く構造化されていれば、簡単な試験を行うだけで機能をチェックできます。

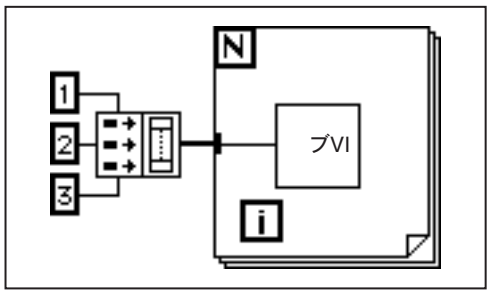
共通処理を見つける

プログラムを作成するとき、特定の処理を頻繁に行うことに気付く場合があります。場合によっては、サブVIやループを使用して繰り返し処理を実行することを検討してください。

たとえば、3つの似た動作を独立して実行する次のダイアグラムの場合を考えてみましょう。



この設計の別の方法として、同じ動作を3回行うループがあります。異なるパラメータから成る配列を作成して自動指標付けを使用すると、ループの各繰り返しごとに正しい値を設定することができます。



配列要素が定数の場合は、ブロックダイアグラムで配列を作成せずに、配列定数を使用することができます。

左から右へレイアウトする

Gは、左から右へ（場合によっては上から下へ）レイアウトするように設計されています。プログラムのすべての要素を、できるだけこのレイアウトで配置してください。

エラーチェック

どのような I/O 動作を行う場合も、エラーが発生する可能性を考慮する必要があります。ほとんどすべての I/O 関数は、エラー情報を返します。ダイレクト I/O を使用する場合は必ず、ユーザプログラム側でエラーを確認し正しく処理してください。

通常はユーザによって行いたいエラー処理が異なりますので、LabVIEW では自動的なエラー処理は行いません。たとえば、ブロックダイアグラムで I/O VI がタイムアウトになった場合でも、プログラム全体を停止させたいときとそうでないときがあります。また、一定時間内に VI を再実行したい場合もあります。LabVIEW では、このようなエラー処理をユーザが自分の VI のブロックダイアグラム内で決定できるようになっています。

エラーが頻繁に発生するのは、次のような場合です。

- 通信処理の初期化が正しくない、または外部機器に書き込まれたデータが不適切
- 外部機器の電源が入っていないか、故障、または動作異常
- オペレーティングシステムソフトウェアをアップグレードしたときにアプリケーションやライブラリの機能が変更されている

エラーが発生した場合、それ以降の特定の動作は行わないようにした方が良いでしょう。たとえば、誤った機器を指定したためにアナログ出力動作が正常に行われなかった場合は、それ以降のアナログ入力動作は行わない方が良いでしょう。

このような問題に対処する一つの方法として、すべての関数の後にエラーをテストし、Case ストラクチャ内に以降の関数を配置する方法があります。ただし、この方法ではダイアグラムが複雑になってアプリケーションの目的がわからなくなってしまう恐れがあります。

他に、数多くのアプリケーションや VI ライブラリで使用され成功している方法として、I/O を実行するサブ VI の中にエラー処理を組み込んでしまう方法があります。エラー入力とエラー出力は、各 VI ごとに設けることができます。エラー入力をチェックして、前にエラーが発生していないか調べるように VI を設計することができます。エラーがある場合、何もしないでエラー入力をそのままエラー出力に渡すように VI を構成することもできます。エラーがない場合、VI は処理を実行し結果をエラー出力に渡すことができます。

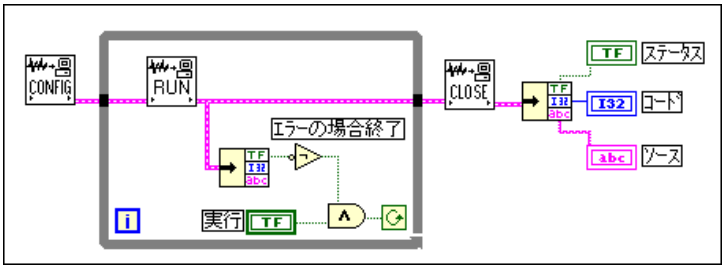
☞ **注　閉じる動作など、VI によっては渡されたエラーに関係なく処理を実行した方がよい場合もあります。**

前述の方法を使用する場合、いくつかの VI を一緒に配線して、一つの VI から次の VI にエラーが伝達されるようにエラー入力とエラー出力を接続

することができます。一連のVIの終わりにSimple Error Handler VIを使用すると、エラーが発生した場合にダイアログボックスを表示することができます。Simple Error Handler VIは**関数→時間 & ダイアログ**の中にあります。この方法は、エラー処理を組み込む場合だけでなく、複数のI/O動作の順序を決定する場合にも使用することができます。

エラー入力と出力のクラスタを使用する一つの利点は、これらを使用して異なる動作の実行順序を制御できることです。

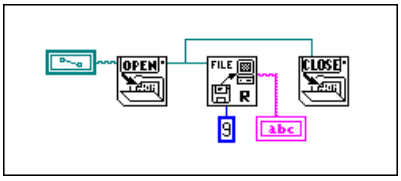
エラー情報は通常、番号によるエラーコード、エラーの発生した関数名を含む文字列、およびすばやくテストを行うためのエラーブール値を含むクラスタを使用して表示されます。次の図は、ユーザアプリケーションにこの方法がどのように使用されているのかを示します。エラーを検出するとWhileループが停止することに注意してください。



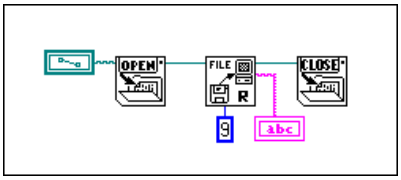
依存性の欠けている部分に注意する

イベントの順序が明確に定義されているかどうかを、必要に応じ確認してください。データ依存性がない場合は、左から右、上から下に実行されるとは考えないでください。

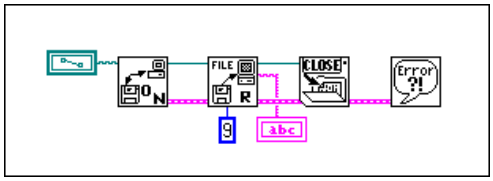
次の例ではRead File VIとClose File VIの間に依存性がありませんので、このプログラムは予想通りに動作しない可能性があります。



次のようなブロックダイアグラムではRead File VIの出力をClose File VIに配線することで依存性を持たせています。この場合はClose File VIがRead File VIの出力を受け取るまでは、動作が終了することはありません。



ただし、前の例ではエラーチェックを行っていないことに注意してください。たとえばファイルが存在しない場合でも、このプログラムでは警告が表示されません。次のブロックダイアグラムに、この問題に対処する一つの方法を示します。この例のブロックダイアグラムでは、これらの関数のエラーI/O入力と出力を使用してすべてのエラーをSimple Error Handler VIに伝達しています。



シーケンスストラクチャの使いすぎを避ける

VIは本質的にかなりの並行動作ができるようになっていますので、シーケンスストラクチャは使わないようにしてください。シーケンスストラクチャを使用すると実行順序は保証されますが、並行動作を妨げてしまいます。たとえばI/Oデバイス（GPIB、シリアルポート、データ集録ボード）を使用する非同期タスクは、シーケンスストラクチャによって妨げられない限り、他の動作と並列して動作することができます。

シーケンスストラクチャは、プログラムの各部分をわかりにくくし、左から右への自然なデータフローを妨げる傾向があります。シーケンスストラクチャを使用したために性能が悪くなることはありませんが、シーケンス動作が必要な場合は、データフローを使用することを検討した方が良いでしょう。たとえばI/O動作の場合は、前述のエラーI/Oテクニックを使用して、一つのI/O動作が必ず別のI/O動作の前に行われるようにした方が良いでしょう。

例の研究

プログラム設計についての詳細は、LabVIEWに含まれている多くのブロックダイアグラム例を参照していただくことができます。これらのサンプルプログラムを見れば、Gのプログラミング方式やテクニックのポイントがわかります。これらのブロックダイアグラムは、Examplesディレクトリのいずれかの VI を開いてご覧ください。

29

次に学習すべき内容

ここまで行ってきた作業で、LabVIEW アプリケーションを作成するための準備はできましたが、この他にもいくつかのリソースが用意されていますので、ユーザ独自のアプリケーション作成を開始する前に、チェックしておいた方が良いでしょう。

役に立つその他の資料

アプリケーションを効率的に作成するのに使用可能なリソースの概要を、以下に示します。

ソリューションウィザードとサンプル検索

自分のアプリケーションに似ている例を探すには、LabVIEW ダイアログボックスから**ソリューションウィザード**オプションと**サンプル検索**オプションを開きます。**ソリューションウィザード**は、ユーザが指定した基準に従ってデータ集録 I/O と計測器 I/O のサンプルを作成します。**サンプル検索**オプションは、G プログラミングに関するいくつかの概念や、解析、ネットワーク処理、データ集録、計測器I/Oなどを説明したサンプルを開きます。

データ集録アプリケーション

(Windows、Macintosh)『LabVIEW データ集録ベーシックマニュアル』には、アナログ入力、アナログ出力、デジタル I/O、カウンタ／タイマを使用したデータ集録アプリケーションの作成を開始するための情報が掲載されています。また、このマニュアルは、データ集録の背景となる基本概念や、それらの概念を実行するのに使用する VI について説明しています。

G プログラミングのテクニック

基本的な G プログラミングのテクニックについての概要は、本書の「第 I 部 G プログラミングの概要」と「第 V 部 上級 G プログラミング」で説明しています。G の機能についてさらに詳しく学習するには、『G プログラミングリファレンスマニュアル』を参照してください。このマニュアルでは、実行とデバッグ、VI のセットアップ、フロントパネルオブジェクト、配線、ストラクチャ、属性ノードなどについて詳しく説明しています。また、印刷、**環境設定**を使用して G 環境をカスタマイズする方法、カスタム制御器、マルチスレッド処理、性能上の問題など、『LabVIEW ユーザマニュアル』では説明していない情報も掲載されています。

関数とVIに関する参考資料

LabVIEWで使用可能な関数とVIの概要は『LabVIEWオンラインリファレンス』を参照してください。**関数**パレットに表示されるのと同じ順序に区分された関数とVIの説明が掲載されています。

さらに高度な問題に関する資料

『LabVIEW ユーザマニュアル』では、LabVIEW アプリケーションの作成に関する基本事項を説明しています。LabVIEW に含まれている機能の中には、本書で一部しか、または全く説明されていない高度なものもいくつかあります。以下の項ではこれらの機能の概要を示し、アプリケーションで必要に応じて使用できるその他のリソースについても説明します。

属性ノード

本書の「第27章　フロントパネルオブジェクトの属性」では属性ノードについて簡単に説明しています。属性ノードを使用すると、制御器や表示器に関連する設定をプログラムにより操作することができます。たとえば制御器の表示／非表示などを変更することができます。リング制御器やリスト制御器のオプションを変更したり、チャートの内容を消去したり、チャートやグラフの目盛りをプログラムによって変更する必要がある場合に、属性ノードを使用してください。『G プログラミングリファレンスマニュアル』の「第22章　属性ノード」では、属性ノードについて詳細に説明しています。

VIのセットアップと環境設定

VIサーバ機能を使用すると、LabVIEWの属性だけでなくVIのさまざまな属性もプログラムにより制御することができます。VI をメモリにロードし、VIを実行し、VIの外観を変更し、実行時のVIの動作を変更することができます。**環境設定**や**VI設定...**で対話式に設定可能なオプションのいくつかは、プログラムによっても設定できます。**環境設定**で使用可能なオプションはLabVIEWで表示されるすべてのVIに影響を及ぼすことから、アプリケーションに対する設定と考えることができます。また、**VI設定...**オプションは、1つのVIと、そのVIをサブVIとして使用する場合にしか影響を及ぼさないことから、VIに対する設定と考えることができます。

呼び出されるプロパティである属性の変更だけでなく、LabVIEW や個々の VI の動作も実行することができます。たとえば VI を保存するための、メソッドとも呼ばれる動作を設定することができます。LabVIEW や VI のプロパティやメソッドは、ローカルマシンで設定できるほか、TCP/IP ネットワークを経由したり他の ActiveX アプリケーションを使用して設定することもできます。ActiveX の機能についての詳細は、本書の「第 22 章

ActiveXのサポート」を参照してください。TCP/IPの設定やこの機能の使用方法に関する詳細は、『Gプログラミングリファレンスマニュアル』の「第7章　G環境をカスタマイズする」と「第21章　VIサーバ」を参照してください。

ローカル変数とグローバル変数

ブロックダイアグラムの制御器を複数の位置から読み込む場合には、ローカル変数を使用します。また、フロントパネルオブジェクトを、ある位置では制御器として扱い別の位置では表示器として扱う必要がある場合も、ローカル変数を使用します。ローカル変数の使用は最小限にとどめてください。理由は、ローカル変数を多用するとダイアグラムでのデータフローが見えなくなるため、プログラムの目的がわかりにくくなり、ローカル変数のデバッグが難しくなるからです。ローカル変数についての説明は、『Gプログラミングリファレンスマニュアル』の「第23章　グローバル変数とローカル変数」を参照してください。

グローバル変数は、複数のVIで使用されるデータを格納します。グローバル変数を使用するとダイアグラムでのデータフローが見えなくなるため、注意して使用してください。アプリケーションによってはグローバル変数が必要なものがありますが、データ転送に別のデータフローメソッドを使用したプログラム構成が可能な場合は、グローバル変数を使用しないでください。詳細は、『Gプログラミングリファレンスマニュアル』の「第23章　グローバル変数とローカル変数」を参照してください。

サブVIを作成する

サブVIを作成するには、ブロックダイアグラムの一部を選択し、**編集→選択範囲をサブVIに変更**を選択します。また、LabVIEWは自動的に正しい入力と出力をサブVIに配線します。場合によっては、VIからサブVIを作成できないことがあります。これについての詳細は、『Gプログラミングリファレンスマニュアル』の「第3章　サブVIを使用する」を参照してください。

VIプロフィール

VIプロフィール機能（**プロジェクト→プロフィールウィンドウを表示**）を使用すると、VIのタイミングに関する統計データや詳細情報にアクセスできます。プロフィール機能についての詳細は、『Gプログラミングリファレンスマニュアル』の「第28章　性能に関する問題」を参照してください。

制御器エディタ

フロントパネルの制御器の外観をカスタマイズするには制御器エディタを使用します。また、このエディタを使用してカスタマイズした制御器を保存しておき、他のアプリケーションでその制御器を使用することもできます。制御器エディタについての詳細は、『G プログラミングリファレンスマニュアル』の「第24章 カスタム制御器とType Def」を参照してください。

リスト制御器とリング制御器

オプションのリストをユーザに表示する必要がある場合は、リスト制御器とリング制御器を使用します。これらのフロントパネルオブジェクトの詳細は、『G プログラミングリファレンスマニュアル』の「第13章 リストとリングの制御器および表示器」を参照してください。

Call Library 関数

LabVIEW には、共有ライブラリや DLL を呼び出すための Call Library 関数が用意されています。この関数を使用すると、既存のコードやドライバを呼び出すインタフェースを LabVIEW で作成することができます。Call Library 関数についての詳細は、『G プログラミングリファレンスマニュアル』の「第25章 他の言語で書かれたコードの呼び出し」を参照してください。

コードインタフェースノード

従来のテキストベースのプログラミング言語で記述されたソースコードを LabVIEW のブロックダイアグラムから呼び出す別の方法として、コードインタフェースノード（Code Interface Node: CIN）を使用することができます。LabVIEW よりも従来のプログラミング言語の方が実行速度が速いタスクや、ブロックダイアグラムから直接実行できないタスクの場合に、また、既存のコードを LabVIEW にリンクする場合に、CIN を使用します。ただし、ソースコードの呼び出しに使用する場合は通常、CIN よりも Call Library 関数の方が簡単です。LabVIEW やソースコードとの密接な統合が必要な場合は CIN を使用します。CIN についての詳細は、『LabVIEW Code Interface Reference Manual』を参照してください。このマニュアルは PDF（Portable Document Format）ファイルでのみ利用できます。また、『G プログラミングリファレンスマニュアル』の「第25章 他の言語で書かれたコードの呼び出し」も参照してください。

A

解析に関する参考文献

この付録では、本書で説明した解析 VI を作成する際に使用した参考文献の一覧を示します。これらの参考文献には、解析ライブラリに実装されている理論やアルゴリズムに関する詳細が説明されています。

Baher, H. *Analog & Digital Signal Processing*. New York: John Wiley & Sons, 1990.

Bates, D.M. and Watts, D.G. *Nonlinear Regression Analysis and its Applications*. New York: John Wiley & Sons, 1988.

Bracewell, R.N. “Numerical Transforms.” *Science* 248 (11 May 1990).

Burden, R.L. and Faires, J.D. *Numerical Analysis*. 3d ed. Boston: Prindle, Weber & Schmidt, 1985.

Chen, C.H. et al. *Signal Processing Handbook*. New York: Marcel Decker, Inc., 1988.

DeGroot, M. *Probability and Statistics*. 2d ed. Reading, Massachusetts: Addison-Wesley Publishing Co., 1986.

Dowdy, S. and Wearden, S. *Statistics for Research*. 2nd ed. New York: John Wiley & Sons. 1991.

Dudewicz, E.J. and Mishra, S.N. *Modern Mathematical Statistics*. New York: John Wiley & Sons, 1988.

Duhamel, P. et al. “On Computing the Inverse DFT.” *IEEE Transactions on ASSP* 34, 1 (February 1986).

Dunn, O. and Clark, V. *Applied Statistics: Analysis of Variance and Regression* 2nd ed. New York: John Wiley & Sons. 1987.

Elliot, D.F. *Handbook of Digital Signal Processing Engineering Applications*. San Diego: Academic Press, 1987.

Golub, G.H. and Van Loan, C.F. *Matrix Computations*. Baltimore: The John Hopkins University Press, 1989.

Harris, Fredric J. “On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform.” *Proceedings of the IEEE-66* (1978)-1.

Maisel, J.E. “Hilbert Transform Works With Fourier Transforms to Dramatically Lower Sampling Rates.” *Personal Engineering and Instrumentation News* 7, 2 (February 1990).

Miller, I. and Freund, J.E. *Probability and Statistics for Engineers*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1987.

Neter, J. et al. *Applied Linear Regression Models*. Richard D. Irwin, Inc., 1983.

Neuvo, Y., Dong, C.-Y., and Mitra, S.K. “Interpolated Finite Impulse Response Filters” *IEEE Transactions on ASSP*. ASSP-32, 6 (June, 1984).

O’Neill, M.A. “Faster Than Fast Fourier.” *BYTE*. (April 1988).

Oppenheim, A.V. and Schafer, R.W. *Discrete-Time Signal Processing*. Englewood Cliffs, New Jersey: Prentice Hall, 1989.

Parks, T.W. and Burrus, C.S. *Digital Filter Design*. New York: John Wiley & Sons, Inc., 1987.

Pearson, C.E. *Numerical Methods in Engineering and Science*. New York: Van Nostrand Reinhold Co., 1986.

Press, W.H. et al. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge: Cambridge University Press, 1988.

Rabiner, L.R. & Gold, B. *Theory and Application of Digital Signal Processing*. Englewood Cliffs, New Jersey: Prentice Hall, 1975.

Sorensen, H.V. et al. “On Computing the Split-Radix FFT.” *IEEE Transactions on ASSP*. ASSP-34, 1 (February 1986).

Sorensen, H.V. et al. “Real-Valued Fast Fourier Transform Algorithms.” *IEEE Transactions on ASSP*. ASSP-35, 6 (June 1987).

Spiegel, M. *Schaum’s Outline Series on Theory and Problems of Probability and Statistics*. New York: McGraw-Hill, 1975.

Stoer, J. and Bulirsch, R. *Introduction to Numerical Analysis*. New York: Springer-Verlag, 1987.

Vaidyanathan, P.P. *Multirate Systems and Filter Banks*. Englewood Cliffs, New Jersey: Prentice Hall, 1993.

Wichman, B. and Hill, D. “Building a Random-Number Generator: A Pascal Routine for Very-Long-Cycle Random-Number Sequences.” *BYTE* (March 1987): 127–128

B

一般的な質問

付録 B では、LabVIEW のネットワーク通信、計測器 I/O、特に GPIB とシリアル I/O についてよくお問い合せをいただく質問にお答えします。

通信に関する一般的な質問

この項では、LabVIEW のネットワーク通信について頻繁にお問い合せいただく質問にお答えします。質問は、関連するプラットフォームによって、全プラットフォーム、Windows、Macintosh の各場合ごとに分類してあります。LabVIEW についてさらに詳しい情報が必要な場合やご提案については、ナショナルインスツルメンツまでご連絡ください。

すべてのプラットフォームに共通な質問

LabVIEW を使用して他のアプリケーションと通信する方法を教えてください。

他のアプリケーションとの通信は、多くの場合プロセス間通信またはアプリケーション間通信と呼ばれ、各プラットフォームの標準ネットワークプロトコルを使用して行うことができます。LabVIEW は、すべてのプラットフォーム上で TCP（転送制御プロトコル）と UDP（ユーザデータグラムプロトコル）をサポートしています。

（Windows） Windows 対応の LabVIEW では、この他に DDE（動的データ交換）をサポートしています。

（Macintosh） Macintosh 対応の LabVIEW では、この他に IAC（アプリケーション間通信）をサポートしています。IAC には、AppleEvent と PPC（プログラム間通信）が含まれます。

（UNIX） UNIX 対応の LabVIEW では、TCP と UDP しかサポートされません。

LabVIEW を使用して他のアプリケーションと通信する方法の詳細は、「第 II 部　I/O インタフェース」を参照してください。また、多くの計測アプリケーションでは、ファイル I/O を使用すると、アプリケーション間で適切な情報転送を簡単に行えます。

LabVIEW で他のアプリケーションを起動する方法を教えてください。

Windows や UNIX の場合は、`System Exec.vi`（**関数→通信**）を使用してください。Macintosh の場合は、AESend Finder Open（**関数→通信→**AppleEvent）を使用してください。

TCP よりも UDP を使用した方が良いのは、どのような場合ですか？

精度を特に問わないアプリケーションでは通常 UDP が使用されます。たとえば、アプリケーションが接続先に対して重要なデータを頻繁に送るため、データのセグメントが多少失われても問題にならないような場合です。また、サーバに対するリスン動作を行うマシンへのブロードキャストにも、UDP を使用することができます。

TCP や UDP に使用できるポート番号を教えてください。

ポートは、0 ～ 65535 の間の番号で指定されます。UNIX の場合、1024 未満のポート番号は特権アプリケーション（ftp など）用に予約されています。ローカルポートを指定する場合は、未使用のポートを使用するか選択する際に値 0 を使用できます。

UDP をブロードキャストに使用できない理由を教えてください。

ブロードキャストアドレスはドメインによって変わるため、どのブロードキャストアドレスを使用したら良いか、システム管理者に問い合わせなければなりません。たとえば、ご使用されているドメインでは、0XFFFFFFFF はブロードキャストアドレスとして不適切だと思われます。また、ご使用のマシンは、処理がルートユーザにより起動された場合を除き、デフォルトではブロードキャストを行えないようになっている可能性もあります。

Windows のみ

LabVIEW で使用可能な WinSock.DLL を教えてください。

Windows 95 と Windows NT ではこのファイルシステムに含まれていますので、この質問が関係するのは Windows 3.*x* の場合のみです。

規格 1.1 に適合する WinSock ドライバであればどれでも LabVIEW で動作するはずです。

推奨品

ナショナルインスツルメンツは Microsoft Windows for Workgroups から提供されている WinSock DLL を使用されることをお奨めします。この最新の WinSock DLL 以前にリリースされている DDL については、弊社で各種 WinSock DLL についてテストを行った結果、以下を推奨します。

- TCPOpen バージョン 1.2.2（Lanera Corporation）
- Trumpet バージョン 1.0

- Super-TCP バージョン 3.0 R1（Fronteir Technologies Corporation）
- NEWT/Chameleon バージョン 3.11（NetManage, Inc.）
- NetManage, Inc.（米国 408 973 7171）の NEWT/Chameleon バージョン 3.11
- Microsoft の Windows for Workgroups の winsock.dll

推奨できないもの

ナショナルインスツルメンツでテストした範囲では、以下の製品で、TCP/IP 通信を試みようとするとさまざまな問題やクラッシュが発生しました。現時点では、当社は以下の製品は推奨できません。また、当社ではこれらの WinSock.DLL で TCP/IP 通信を行おうとしているユーザのサポートもお引受けできません。

- Distinct TCP/IP バージョン 3.1（Distinct Corporation）
- PCTCP バージョン 2.*x*（FTP Software, Inc.）

DDE を使用して Excel のマクロを呼び出す方法を教えてください。

`DDE Execute.vi` を使用してください。この VI は、ユーザが Excel に実行させる処理とマクロ名を含むコマンド文字列を実行するように、DDE サーバに指示します。必ず、コマンドを正しい括弧やカギ括弧で囲ってください。詳しくは、『Excel ユーザーガイド』を参照してください。一般的に使用されるいくつかの例を、次に示します。

コマンド文字列	処理
[RUN("MACRO1")]	MACRO1 を実行する
[RUN("MACRO1!R1C1")]	第 1 行第 1 列から MACRO1 を実行する
[OPEN("C:¥EXCEL¥SURVEY.XLS")]	SURVEY.XLS を開く。

Microsoft Access では DDE Poke を使用できない理由を教えてください。

Microsoft Access は、DDE クライアントから直接データを受けることができません。Access データベースにデータを書き込むためには、そのデータベース内でマクロを作成してファイルからデータをインポートしなければなりません。簡単な場合は、これらのマクロは処理 2 つ分しか必要としません。最初に Access のダイアログを抑制するための SetWarnings を実行し、次に TransferSpreadsheet または TransferText を実行してデータを読み込みます。このマクロを定義した後、マクロ名を付けた実行可能ファイルをデータとしてそのデータベースに送れば、このマクロを呼び出すことができます。これを行う方法については、`examples¥comm¥access.llb` の `Sending Data to Access.vi` というサンプル VI を参照してください。

DDE を使用して LabVIEW 以外のアプリケーションと通信するためには、どのようなコマンドを使用すればよいでしょうか？

DDE コマンドは、インタフェースするアプリケーションによって異なります。個々のアプリケーションのユーザマニュアルまたはオンラインマニュアルで、使用できるコマンドを調べてください。

LabVIEW を共有アプリケーションとしてファイルサーバにインストールする方法を教えてください。

各クライアント用のライセンスをご購入されている場合、次の手順で行います。

- LabVIEW 開発システムをサーバにインストールします(サーバ上にNIハードウェアがある場合以外は、NI-DAQ や GPIB.DLL をインストールする必要はありません)。
- 各ローカルマシンは、LabVIEW のユーザ設定に合ったそれぞれのマシンの labview.ini ファイルを使用する必要があります。ローカルマシンに labview.ini ファイルがまだない場合は、Microsoft Notepad などのテキストエディタを使用してこの空のテキスト文書を作成してください。labview.ini の 1 行目は [labview] でなければなりません。labview.ini 用のローカル設定を行うために、LabVIEW は、ユーザ設定へのパスを含むコマンド行パラメータを必要とします。たとえば、labview.exe ファイルがドライブ W:¥LABVIEW にあり、labview.ini ファイルが C:¥LVWORK（ローカルマシンのハードディスクドライブ）にある場合は、プログラムマネージャの LabVIEW アイコンのコマンド行オプションを次のように修正します。

```
W:¥LABVIEW¥LABVIEW.EXE -pref
C:¥LVWORK¥LABVIEW.INI
```

☞ **注**　pref は小文字でなければなりません。また、各ローカルマシンに、それぞれの LabVIEW テンポラリディレクトリが存在しなければなりません。LabVIEW でこれを行うには、編集→環境設定 ... を選択します。

- サーバマシンでは、GPIB ボードを使用している場合以外は GPIB.DLL は必要ありません。LabVIEW ディレクトリには gpibdrv ファイルが必要です。この場合は GPIB ボードが付いている各マシンに、ボードに対応するドライバをインストールする必要があります。このためには、ボードに付属のドライバを使用するか、LabVIEW を独自にインストールして希望する GPIB ドライバだけをローカルマシンにインストールします。
- NI-DAQ に対しても GPIB.DLL の場合と同じ手順が適用されます。

NT で同期 DDE クライアント／サーバにより何度も転送を行うとハングアップしてしまう理由を教えてください。

NT 対応 LabVIEW の DDE にはいくつかの問題があり、DDE Poke や DDE Request の動作中に VI がハングアップしてしまいます。このような制約は、Windows NT の場合に限られます。

Macintosh のみ

ターゲット ID とは何ですか？

ターゲット ID は、Macintosh の AppleEvent VI や PPC VI で使用されます。ターゲット ID は、ユーザが起動、実行、中止しようとしているアプリケーションを参照するためのものとして機能します。アプリケーションに対応するターゲット ID にアクセスするには、以下のいずれかのコマンドを使用します。

- Get Target ID は、アプリケーションの名前とロケーションを入力として受け取り、ネットワーク上を探して、ターゲット ID を返します。
- PPC Browser はダイアログボックスをポップアップしますので、ユーザはこのダイアログボックスを使用して、ネットワーク経由で、または自分のコンピュータ上のアプリケーションを選択できます。

VI の開始時に生成したターゲット ID は、アプリケーションを開いたり閉じたり、印刷や実行などを行うこれ以降のすべての AppleEvent に対する入力として使用しなければなりません。

PPC Browser により生成されるダイアログボックスに自分のアプリケーションが表示されないのはなぜですか？

接続したいアプリケーションを AppleEvent で使用できない場合、このアプリケーションは `PPC Browser` ダイアログボックスには表示されません。希望するアプリケーションが AppleEvent をサポートすることがわかっている場合は、アプリケーションが入っているマシンのアップルメニューを開きます。**コントロールパネル→ファイル共有（MacOS8）**または**コントロールパネル→共有設定（System 7 以前のバージョン）**を選択し、**プログラムリンク**を ON にします。

AppleEvent を使用して Finder を閉じる方法を教えてください。

Finder やその他のアプリケーションを終了するには、`AESend Quit Application` という VI を使用します。

GPIB

すべてのプラットフォーム

LabVIEW の計測器ドライバを使用するときに、計測器との通信がうまくできません。

GPIB インタフェースが正常に動作しているか確認してください。examples¥instr¥smplgpib.llb の LabVIEW<->GPIB.vi の例を使用して、計測器に簡単なコマンドを送ってみます。たとえば、計測器が488.2である場合、文字列 *IDN? は、計測器の識別文字列（約100文字）を送るよう計測器に要求します。

通信が確立されれば、計測器ドライバで問題は発生しないはずです。

LabVIEW でGPIB Read/Write を使用すると、タイムアウトエラーになります。

簡単なプログラムを実行してLabVIEWとGPIBの間の通信を確立してみてください。examples¥instr¥smplgpib.llb の LabVIEW<->GPIB.vi の例を使用して、計測器に簡単なコマンドを送ってみます。たとえば、計測器が 488.2 である場合、文字列 *IDN? は、計測器の識別文字列（約 100 文字）を送るよう計測器に要求します。

それでも GPIB に関するエラーが発生する場合は、構成に問題があると思われます。GPIB 構成ユーティリティを開き（Windows3.*x*/NT の場合はコントロールパネル、Windows 95 の場合は、デバイスマネージャでご使用の GPIB ボードのプロパティ、Mac の場合はNI-488 Config、Sun の場合はibconf、HP-UX の場合はibconf）設定がハードウェアの設定と一致しているか確認します。構成ユーティリティを終了して、Windows プラットフォーム用のibic（対話式制御ユーティリティ）を実行します（Mac:ibic GPIB ボードに付属; Sun:ibic; HP-UX:ibic）。

次の手順を試してください。

`: ibfind gpib0`	GPIB インタフェースを見つける
`id = 32000` `gpib0: ibsic`	GPIB バスを消去する （Interface Clear を送る）
`[0130] [cmpl cic atn]`	動作が正常終了
`gpib0: ibfind dev1`	デバイス 1 を見つける。 適切な計測器アドレスを使用する。
`id = 32xxx` `dev1: ibwrt "*IDN?"`	計測器に文字列を書き込む。488.2 計測器はこのコマンドを認識して識別文字列を返す。
`count = 5` `dev1: ibrd 100`	5 バイトが送られた。計測器から最大 100 バイトを読み込む。
`Fluke xxx Multimeter?`	計測器は識別文字列を返す。

構成に間違いがあれば、上記のステップのいずれかでエラーメッセージが送られてきます。エラーメッセージに関する説明は、GPIB ボードに付属の『NI 488.2 Software Reference Manual』に掲載されています。

LabVIEW の VI を実行ハイライトモードで動作させているときには GPIB 計測器と通信できるのに、フルスピードで動作させているときには通信できないのはなぜですか？

これは、タイミングの問題のように思われます。実行ハイライトが有効になっていると、有効でないときに比べて、VI の実行速度ははるかに遅くなりますので、計測器が送信するデータの準備に要する時間が長くなる場合があります。`GPIB Read.vi` の前に Wait 関数を追加するかサービス要求を使用して、コンピュータに送り返すデータを生成するのに十分な時間を計測器に与えるようにしてください。

GPIB 計測器への書き込みは正常にできるのに、読み込みはできません。理由を教えてください。

`GPIB Write.vi` の実行時、コンピュータはトークモード、計測器はリスンモードになっています。`GPIB Read.vi` の実行時は、デバイスがトークモード、コンピュータがリスンモードになっていることが前提となります。デバイスのモード切り換えを促すのは終了信号で、それは文字（End Of String）や GPIB バスライン（End Or Identify）である場合があります。したがって、`GPIB Read.vi` でタイムアウトが発生したり EABO（処理中止）エラーが返ってくる場合は、デバイスが正しい終了信号を受信していないことを意味します。個々の計測器の終了モードを判断するには、各計

測器に付属のマニュアルを参照してください。原則として、IEEE 488.2 デバイスはすべて、<CR><LF>、または GPIB バスの EOI（End Or Identify）ラインのアサートで終了します。

終了文字を変更する場合は、各プラットフォームに合った構成ユーティリティを使用してください（Windows3.*x*/NT の場合はコントロールパネル、Windows 95 の場合は、デバイスマネージャでご使用の GPIB ボードのプロパティ、Mac の場合は NI-488 Config、Sun の場合は ibconf、HP-UX の場合は ibconf）。

GPIB 計測器と通信する VI が、プラットフォームによって正常に動作する場合とそうでない場合があります。

計測器がコントロールパネル、NI-488 Config、または ibconf で正常に構成されているか確認してください。旧型の 488.1 計測器の中には、自動的にリモート状態にならないものがあります。ご使用のプラットフォームに合った GPIB 構成ユーティリティに移動して Assert REN when SC フィールドを設定してください。こうしておくと、計測器はアドレス指定されたとき確実に（ローカル状態でなく）リモート状態になります。

Windows のみ

Quick Basic プログラムを使用すると計測器と通信できるのですが、LabVIEW からは通信できません。

GPIB ボードには、DOS と Windows 用に個別のハンドラがあります。Quick Basic は DOS ハンドラにアクセスするのに対し、LabVIEW は Windows ハンドラにアクセスします。wibconf.exe でボードやデバイスが正しく構成されているか確認してください。

シリアル I/O

すべてのプラットフォーム

Serial Port Write VI で送ったコマンドに計測器が応答しませんが、理由を教えてください。

多くの計測器では、コマンド文字列の最後に復帰文字または改行文字が付いているものと想定しています。LabVIEW の Serial Port Write vi は文字列入力に含まれる文字しか送らないため、終了文字は文字列に追加されません。多くの端末エミュレーション（Windows Terminal など）パッケージでは、送信の最後に必ず自動的に復帰文字が追加されます。

LabVIEW では、計測器の必要に応じて、Serial Port Write vi への文字列入力に適当な終了文字を付ける必要があります。

計測器によって、復帰文字（¥r）が必要なものと、改行文字（¥n）が必要なものがあります。キーボードでリターン（PC のキーボードの場合は主英数字キーパッドの Enter キー）を入力すると、LabVIEW で ¥n が挿入されます。復帰文字を挿入するには、Concatenate Strings 関数を使用して、復帰文字定数を文字列に追加するか、文字列ポップアップメニューから '¥' Codes Display を選択して手動操作で［¥r］を入力します。

ケーブルが正常に動作するか確認します。当社に技術サポートを依頼される場合の問題の多くは、ケーブルの不良に関係があります。シリアル I/O によるコンピュータ間通信を行う場合は、null モデムのケーブルを使用して受信信号と送信信号を逆にしてください。

計測器との通信の確立については、examples¥instr¥smplserl.llb の LabVIEW<->Serial.vi の例を参照してください。この VI には、シリアルポートからデータを読み返す前に Bytes at Serial Port.vi を使用する方法も示されています。

他のアプリケーションが使用できるようにシリアルポートを閉じる方法を教えてください。

シリアルポートは、使用後閉じた方が良いでしょう。たとえば Windows で、ある VI が Serial Port Write.vi を使用して、プリンタに接続された lpt1 に情報を書き込む場合があります。処理の終了後も、そのシリアルポートは LabVIEW が制御しているため、LabVIEW が制御を解除するまで、他のアプリケーションはこのポートを使用することができません。

LabVIEW には、すべてのプラットフォーム用の Close Serial Driver.vi が用意されています。この VI は、指定されたポートの制御を解除するよう LabVIEW に指示します。Close Serial Driver.vi は**シリアル**パレットではなく vi.lib¥lnstr¥_sersup.llb の中にあります。この VI にアクセスするには、**関数→VI を選択**...または**ファイル→開く**...コマンドを使用します。

シリアルポートのバッファを消去する方法を教えてください。

バッファに残っているデータを読み込みます。データの内容は無視します。

コンピュータにシリアルポートを増設する方法を教えてください。

IBM PC 互換機にシリアルポートを増設するには、ナショナルインスツルメンツの AT バスシリアルインタフェースボードを使用します。その他のプラットフォームをご使用の場合は、他社のボードを使用してコンピュータにシリアルポートを増設します。他社のボードの中には、プラットフォームに装備されているシリアルポート用の標準 API に適合しない特殊な言語インタフェースが必要なものもあります。このような場合は、通常は CIN

またはDLL によりドライバにアクセスするためのユーザ独自のインタフェースを作成する必要があります。次の項では、LabVIEW に含まれているシリアルポートVIを使用して、さまざまなプラットフォーム上の標準シリアルポートインタフェースを使用するボードをアドレス指定する方法を説明します。

（Windows 3.*x*）Windows 対応 LabVIEW は、シリアルポートに対しては標準のMicrosoft Windowsインタフェースを使用しています。したがって、LabVIEW におけるシリアルポートの使用上の制約は、Windows での制約によるものです。たとえば、Windows ではCOM1 ～ COM9 のポートしかアドレス指定できないため、LabVIEW でもこれらのポートしかアドレス指定できません。また、Windowsでは常に8個のシリアルポートにしかアクセスできないため、LabVIEWでも同時に8個までのシリアルポートしか制御できません。

現在ナショナルインスツルメンツでは、さまざまなシリアル通信ソリューション用のプラグ＆プレイATシリアルボードを販売しています。AT-485およびAT-232非同期シリアルインタフェースボードは、2または4ポート構成で使用できます。プラグ＆プレイに完全対応しているため、設定を切り換える必要がなく、インストールやメンテナンスが簡単です。AT-485とAT-232にはWindowsで使用するためのデバイスドライバ、ハードウェア診断テスト、構成ユーティリティなどのソフトウェアコンポーネントが含まれています。これらのボードは、Windows対応LabVIEWでテスト済みです。詳細は、ナショナルインスツルメンツにお問い合せください。

メーカー：　ナショナルインスツルメンツ

製品名：　AT-485およびAT-232

Tel：　03 5472 2980

Fax：　03 5472 2977

Windowsの制約に対する何らかの対策を施した他社のボードは、LabVIEWでは動作したとしてもまず良好には動作しません。このようなボードにアクセスするCINやDLLを作成することはできますが、お勧めできません。一般に、標準のWindowsインタフェースを使用するボードはLabVIEWと完全な互換性があります。

（Windows 95およびWindows NT）Windows 3.*x*と異なり、Windows 95やWindows NTでは、シリアルポートが8個に制約されることはありません。Windows 95とWindows NTの場合、LabVIEWは最大256個までのシリアルポートをアドレス指定できます。デフォルトのポート番号パラメータは、COM1が0、COM2が1、COM3が2などのようになります。

`labview.ini`ファイルには、LabVIEWを構成するためのオプションが含まれています。シリアルポートVIで使用するデバイスを設定するには、使

用されるデバイスのリストに合わせて構成オプション labview.serialDevices を設定します。たとえばWindows 3.*x* と同様にデバイスをセットアップするには、次のようにします。

```
serialDevices="COM1;COM2;COM3;COM4;COM5;COM6;COM7;COM8;
COM9;¥¥.¥COM10;¥¥.¥COM11;¥¥.¥COM12;¥¥.¥COM13;¥¥.¥COM14;
¥¥.¥COM15;¥¥.¥COM16;LPT1;LPT2"
```

この行は、構成の中では1 つの行として表示されます。

（Macintosh）LabVIEW は、標準システム INIT を使用してシリアルポートと通信します。デフォルトの場合 LabVIEW は、ドライバ .aIn と .aOut とモデムポートをポート 0 に使用し、ドライバ .bIn と .bOut とプリンタポートをポート 1 に使用します。増設ポートにアクセスするには、増設ボードと添付されている INIT を一緒にインストールする必要があります。

ボードと INIT（複数の場合あり）をインストールしたら、増設ポートの使用方法を LabVIEW に指示します。LabVIEW 5.0 で使用する方法は、旧バージョンで使用した方法と多少異なります。

（vi.lib¥Instr¥_sersup.llbの中にある）グローバルなserpOpen.viを、増設ポートに対応できるように修正します。シリアルポート用の VI はすべて Open Serial Driver.vi を呼び出し、この VI が serpOpen.vi から対応する入出力ドライバ名を読み込みます。serpOpen.vi のフロントパネルには、入力ドライバ名、出力ドライバ名という 2 つの文字列配列があります。

Open Serial Driver.vi を呼び出すと、この VI は文字列配列に対する指標としてポート番号入力を使用します。したがって、LabVIEW が増設シリアルポートを認識できるようにするためには、入力ドライバ名と出力ドライバ名に別の文字列要素を追加し、**現在のすべての設定をデフォルトに設定する**を選択して、serpOpen.vi に対する変更内容を保存します。

プラグインボードの各シリアルポートには 2 つの名前があり、1 つが入力に、1 つが出力に使用されます。ドライバのインストールに関する正確な名前と説明は、ボードに添付されている資料に掲載されています。説明がない場合やわかりにくい場合は、ボードのメーカーにお問い合せください。

以下のボードは、Macintosh 対応LabVIEW で動作します。

メーカー：　Creative Solutions, Inc.
製品名：　Hurdler HQS（4 ポート）またはHDS（2 ポート）
Tel：　301 984 0262（米国）
Fax：　301 770 1675（米国）

メーカー：　Greensprings
製品名：　RM 1280（4 ポート）
Tel：　415 327 1200（米国）
Fax：　415 327 3808（米国）

（UNIX）Solaris 1 とConcurrent PowerMAXが動作しているSun SPARCstation の場合のシリアルポート VI 用の**ポート番号**パラメータは、`/dev/ttya` が 0、`/dev/ttyb` が1 などのようになります。Solaris 2 の場合は、ポート 0 が `/dev/cua/a`、1 が `/dev/cua/b` などのようになります。HP-UX の場合は、ポート番号0 が `/dev/tty00`、1 が `/dev/tty01` などのようになります。

Concurrent PowerMAX の場合は、ポート 0 が `/dev/console`、ポート 1 が `/dev/tty1`、ポート 2 が `/dev/tty2` などのようになります。

他のメーカのシリアルポートボードでは自由にデバイス名が付けられているため、LabVIEW ではポートの番号付けを簡略化するための簡易インタフェースを開発しました。Sun、HP-UX、Concurrent PowerMAX 対応のLabVIEW には、シリアルポートのアドレス指定方法をLabVIEW に指示するための構成オプションがあります。標準の UNIX デバイスを使用するボードはすべて、LabVIEW でサポートされます。メーカーによっては、そのメーカーのボードに対しては`tty`デバイスノードでなく`cua`を使用するように勧めています。LabVIEW は、どちらのタイプのノードもアドレス指定することができます。

ファイル`.labviewrc`にはLabVIEWの構成オプションが含まれています。シリアルポート VI が使用するデバイスを設定するには、使用する予定のデバイスのリストに合わせて構成オプション `labview.serialDevices` を設定します。

たとえば、デフォルトでは次のようになります。

```
labview.serialDevices:/dev/ttya:/dev/ttyb:/dev/ttyc:...
:/dev/ttyz.
```

☞ **注**　**このためには、他社のどのシリアルボードをインストールする場合にも、`standard/dev file`（ノード）の作成方法が含まれ、ユーザがそのファイル名をわかっている必要があります。**

以下のボードは、Sun 対応LabVIEW で動作します。

メーカー：　Sun
製品名：　SBus シリアルパラレルコントローラ
（8 シリアル、1 パラレル）

メーカー：　Artecon, Inc.
製品名：　SUNX-SB-300P
3 シリアル／1 パラレルポートを装備した
ArtePort SBus カード

SUNX-SB-400P
4 シリアル／1 パラレルポートを装備した
ArtePort SBus カード

SUNX-SB-1600
16 シリアルポートを装備した ArtePort SBus カード

これらの製品に関するお問い合せはすべて、米国 (800)873-7869 の SunExpress までお願いします。

シリアルの DTR ラインと RTS ラインの制御方法を教えてください。

`Serial Port Init.vi` を使用すると、ハードウェアによるハンドシェイクを行うようにシリアルポートを構成できますが、アプリケーションによっては、DTR ラインと RTS ラインを手動で切り換えなければならないものもあります。シリアルポートに対するインタフェースはプラットフォームによって異なりますので、各プラットフォームごとにラインの制御メカニズムが異なります。

（Windows） お届けした Windows 対応 LabVIEW には、DTR ラインと RTS ラインをドライブするのに使用される VI が含まれています。この `serial line ctrl.vi` という VI は `vi.lib¥Instr¥_sersup.llb` にあり、これらのラインの制御に使用できます。この VI は関数への入力に従ってこれらのラインを切り換えます。入力には以下のコードを使用できます。

0　ノーオペレーション
1　DTR をクリア
2　DTR を設定
3　RTS をクリア
4　RTS を設定
5　DTR プロトコルを設定
6　DTR プロトコルをクリア
7　ノーオペレーション 2

（Macintosh）Macintosh では、Device Control/Status 関数を使用してシリアルポートを制御できます。『Inside Macintosh』（第 II 巻の 245 ～ 259 ページと第 IV 巻の 225 ～ 228 ページを参照）には、シリアルポートに送信可能な csCode についての情報が掲載されています。コードと対応する関数の一覧を、次の表に示します。

コード	パラメータ	機　　能
13	baudRate	ボーレートを設定する（実際の速度を整数で）
14	serShk	ハンドシェイクパラメータを設定する
16	byte	その他の制御オプションを設定する
17		DTR をアサート
18		DTR を否定
19	char	パリティエラーを置き換える
20	2 chars	パリティエラーを別の文字に置き換える
21		出力フロー制御用の XOff を無条件に設定する
22		出力フロー制御用の XOff を無条件にクリアする
23		最後に送られたのが XOff である場合は入力フロー制御用の XOn を送る
24		入力フロー制御用の XOn を無条件に送る
25		最後に送られたのが XOn である場合は入力フロー制御用の XOff を送る
26		入力フロー制御用の XOff を無条件に送る
27		SCC チャネルをリセットする

（Sun）Sun 対応 LabVIEW では、シリアルポートのハードウェアのハンドシェイクラインの切り換えは特にサポートしていません。これらのラインを手動操作で制御するには、CIN を作成する必要があります。CIN の作成方法に関する詳細は、『LabVIEW Code Interface Reference Manual』の「Chapter 1 CIN Overview」の「Steps for Creating a CIN」の項を参照してください。ただし、このマニュアルは PDF ファイル形式でのみ使用できます。

32k バイトを超えるシリアルバッファを割り当てることができないのはなぜですか？

`Serial Port Init.vi` で32kバイトを超えるサイズのバッファを使用できないのは、Windows と Macintosh ではシリアルポートバッファが 32k に制限されているためです。したがって、これを超えるバッファを割り当てた場合、LabVIEW はバッファをカットして 32k のサイズにします。ただし、Sun ではこの問題は起こりません。

Windows のみ

パラレルポートへのアクセス方法を教えてください。

Windows 3.*x* 対応LabVIEW の場合ポート 10 はLPT1、ポート 11 はLPT2... のようになります。Windows 95/NT の場合は、ポートをシリアルデバイスの LPT1 に設定することができます。パラレルポートに接続されたプリンタにデータを送るには、`Serial Port Write.vi` を使用してください。

シリアルポート VI から送られてくるエラー番号の意味を教えてください。

Windows 対応LabVIEW のシリアルポート VI は、Windows GetCommError 関数から報告されたエラーを返します。シリアルポート VI から返されるエラー番号は、次の表のエラー番号と 0x4000（16,384）の論理和です。返されるエラーにはシリアルポートのステータスが反映されていることに注意してください。エラーは前のシリアルポート関数の結果として発生する場合もあります。戻り値は、以下のエラーの組み合わせになっています。

16進数値	エラー名	意　味
0x0001	CE_RXOVER	受信用待ち行列がオーバーフローしました。入力用待ち行列に空き領域がないか、ファイル終了文字を受信後に受信しました。
0x0002	CE_OVERRUN	ハードウェアが文字を読み込む前に次の文字を受信し、文字が失われました。
0x0004	CE_RXPARITY	ハードウェアがパリティエラーを検出しました。
0x0008	CE_FRAME	ハードウェアがフレーミングエラーを検出しました。
0x0010	CE_BREAK	ハードウェアがブレーク条件を検出しました。

16進数値	エラー名	意　味
0x0020	CE_CTSTO	CTS（clear-to-send）がタイムアウトになりました。文字の送信中に、COMSTAT の fCtsHold メンバで指定された時間、CTS がローになっていました。
0x0040	CE_DSRTO	DSR（data-set-ready）がタイムアウトになりました。文字の送信中に、COMSTAT の fDsrHold メンバで指定された時間、DSR がローになっていました。
0x0080	CE_RLSDTO	RLSD (receive-line-signal-detect) がタイムアウトになりました。文字の送信中に、COMSTAT の fRIsHold メンバで指定された時間、RLSD がローになっていました。
0x0100	CE_TXFULL	関数が待ち行列に文字を登録しようとしたところ、送信用待ち行列に空き領域がありませんでした。
0x0200	CE_PTO	パラレルデバイスと通信しようとしたところ、タイムアウトが発生しました。
0x0400	CE_IOE	パラレルデバイスと通信しようとしたところ、I/O エラーが発生しました。
0x0800	CE_DNS	パラレルデバイスが選択されていませんでした。
0x1000	CE_OOP	パラレルデバイスから用紙切れの信号を受信しました。
0x8000	CE_MODE	要求されたモードがサポートされていないか、`idComDev` パラメータが無効です。設定されている場合、有効なエラーは CE_MODE だけです。

この表を使用するには、エラー番号を確認し、この番号の元になっているコンポーネントに分解します。たとえば、Serial Port Write.vi からエラー16,408 が返された場合、そのエラーは CE_BREAK と CE_FRAME になります（16,408 = 16,384 + 16 + 8 = 0x4000 + 0X0010 + 0x0008 ）。

Sun のみ

シリアル I/O を実行するとエラー –37 が返ってくるのですが。

エラー –37 は、LabVIEW が対応するシリアルデバイスを見つけることができなかったことを意味します。このエラーは、a) /dev/tty? ファイルがマシン上に存在しないか、b) LabVIEW がファイル serpdrv を見つけられなかったことを示します。

デフォルトの場合、LabVIEW は、/dev/ttya をポート 0、/dev/ttyb をポート 1 などのようにアドレス指定します。デバイスにアクセスするには、これらのデバイスが存在し、ユーザが読み込みと書き込みの許可を得ていることが必要です。シリアルポート VI で LabVIEW がアクセスするデバイスを変更するには、serialDevices 構成オプションを .Xdefaults ファイルに追加します。このオプションの使用方法については、『LabVIEW リリースノート』を参照してください。

serpdrv ファイルは LabVIEW に添付されており、LabVIEW と Sun のシリアルポート間のインタフェースとして機能します。このファイルは、libdir 構成オプションで指定されたロケーションになければなりません。このオプションはデフォルトでは LabVIEW ディレクトリに設定されています。したがって、serpdrv は gpibdrv や vi.lib と同じディレクトリになければなりません。

Solaris 1.*x* マシンでシリアル I/O コールがハングアップするのですが。

Sun 対応 LabVIEW では、シリアルポートに関する操作を行うとき非同期 I/O コールを使用します。Generic_Small カーネルでは、非同期 I/O コールの部分がコメントとして除外されています。Sun 対応 LabVIEW からシリアルポートにアクセスするには、Generic_Small でなく標準の Generic カーネルを使用するか、Generic_Small カーネルを修正して SPARC を再起動する必要があります。

C

カスタマーコミュニケーション

この付録には、お客様の技術的問題点を弊社が解決する際に必要な情報をご記入していただく書式と、ドキュメントに関するお客様のご意見やご要望をご記入いただく書式が添付されています。弊社にお問い合わせいただく際には、「テクニカルサポートフォーム」と、システムに関する構成フォーム（お客様のマニュアルに添付されている場合）を事前にご記入のうえご連絡いただきますと、お客様のお問い合わせにさらに迅速に対応できます。

ナショナルインスツルメンツでは、お客様が必要とする情報を速やかに提供するため、電子メール、ファックス、電話によるテクニカルサポートを提供しています。弊社の電子サポートには、Webサイト、FTPサイト、ファックスバックシステムおよび電子メールサポートなどがあります。お持ちのハードウェアまたはソフトウェアに問題が発生した場合は、まず電子サポートシステムをお試しください。そのシステムから入手できる情報では問題が完全に解決しない場合は、ファックスまたは電話サポートをご利用ください。弊社のアプリケーションエンジニアがご質問にお答えします。

電子サポート

Web サポート

弊社のWebサイトにアクセスするには、http://www.natinst.com/nniのアドレスをご利用ください。

FTP サポート

弊社のFTPサイトにアクセスするには、弊社のインターネットホスト `ftp.natinst.com`にanonymousでログオンし、パスワードはお客様のインターネットアドレス（`nihanako@anywhere.co.jp`など）をお使いください。サポートファイルおよび文書は、`/support`ディレクトリにあります。

電子メールサポート

以下のインターネットアドレスで、技術的なご質問を弊社アプリケーションエンジニア宛に電子メールでお送りいただくこともできます。その際には、お客様のご氏名、ご住所、電話番号、および電子メールアドレスを忘れずにご記入ください。折り返し解決方法やご提案を返答します。

`support@nni.co.jp`

なお、一般的なご質問は`info@nni.co.jp`へお送りください。

Fax-on-Demand サポート

Fax-on-Demandは24時間体制の情報検索システムで、広範な技術情報に関する文書ライブラリを収録しています。Fax-on-Demandにアクセスするには、プッシュホンで以下の番号にダイヤルしてください。資料はすべて英文になります。

512-418-1111（米国）

電話およびファックスサポート

ナショナルインスツルメンツの支社は世界各地にあります。以下のリストから、各国の技術サポートの番号をお探しください。国内に支社がない場合は、お客様がソフトウェアをご購入された販売店にご連絡いただき、サポートをご依頼ください。

国名	電話番号	Fax 番号
日本	03 5472 2981	03 5472 2977
イスラエル	03 573 4815	03 573 4816
イタリア	02 413091	02 41309215
英国	01635 523545	01635 523154
オーストラリア	03 9879 5166	03 9879 6277
オーストリア	0662 45 79 90 0	0662 45 79 90 19
オランダ	0348 433466	0348 430673
カナダ（オンタリオ）	905 785 0085	905 785 0086
カナダ（ケベック）	514 694 8521	514 694 4399
韓国	02 596 7456	02 596 7455
シンガポール	2265886	2265887
スイス	056 200 51 51	056 200 51 55
スウェーデン	08 730 49 70	08 730 43 70
スペイン	91 640 0085	91 640 0533
台湾	02 377 1200	02 737 4644
デンマーク	45 76 26 00	45 76 26 02
ドイツ	089 741 31 30	089 714 60 35
ノルウェー	32 84 84 00	32 84 86 00
フィンランド	09 725 725 11	09 725 725 55
フランス	01 48 14 24 24	01 48 14 24 14
米国	512 795 8248	512 794 5678
ベルギー	02 757 00 20	02 757 03 11
香港	2645 3186	2686 8505
メキシコ	5 520 2635	5 520 3282

テクニカルサポートフォーム

この書式はコピーして使用し、ソフトウェアまたはハードウェアの構成を変更するたびにその内容も書き換えてください。ご記入いただいた書式はお客様の現在の構成を参照する資料としてご利用いただけます。技術サポートをご希望の場合は、この書式に必要事項をもれなくご記入のうえご連絡いただきますと、弊社のアプリケーションエンジニアがお客様のご質問にさらに効率よくお答えできます。

ご質問いただいた問題に関連して、弊社の他のハードウェアもしくはソフトウェア製品が使用されている場合には、該当する製品のユーザマニュアルに添付されている構成フォームもご記入ください。

1 ページに納まらない場合は、必要に応じて用紙を追加してください。

ご氏名 _____

貴社名 / 部課名 _____

ご住所 _____

電話(____）_____ ファックス（____）_____

コンピュータのメーカー名_____ 機種_____CPU タイプ_____

OS（バージョン番号もご記入ください）_____

クロック周波数_____MHz　RAM _____MB　ディスプレイアダプタ_____

マウス___ 有り ___ 無し　インストールされている他のアダプタ _____

ハードディスク容量_____MB　メーカー_____

使用している計測器 _____

ナショナルインスツルメンツのハードウェア製品_____ レビジョン番号 _____

構成 _____

ナショナルインスツルメンツのソフトウェア製品_____バージョン _____

構成 _____

問題点を詳しく説明してください。_____

_____

_____

_____

_____

_____

_____

表示されるエラーメッセージを記入してください。_____

_____

_____

どのような手順で操作すると問題が発生しますか。_____

_____

_____

_____

_____

LabVIEW上で使用するハードウェアおよびソフトウェアに関する構成フォーム

各項目の右側の空欄に、ご使用のハードウェアおよびソフトウェアの構成とレビジョン番号をご記入ください。この書式はコピーして使用し、ソフトウェアまたはハードウェアの構成を変更するたびにその内容も書き換えてください。ご記入いただいた書式はお客様の現在の構成を参照する資料としてご利用いただけます。技術サポートをご希望の場合は、この書式に必要事項をもれなくご記入いただいてからご連絡いただきますと、弊社のアプリケーションエンジニアがお客様のご質問にさらに効率よくお答えすることができます。

ナショナルインスツルメンツの製品

ハードウェアの製品名 _____

ハードウェアのレビジョン _____

ハードウェアの割り込みレベル _____

ハードウェアのDMAチャネル _____

ハードウェアのベースI/Oアドレス _____

ナショナルインスツルメンツ社のソフトウェア _____

システムで使用している他のボード _____

他のボードのベースI/Oアドレス _____

他のボードのDMAチャネル _____

他のボードの割り込みレベル _____

他社の製品

コンピュータの型式とモデル名 _____

CPU _____

クロック周波数_____MHz　RAM _____MB

インストールされているビデオボードのタイプ _____

OS名およびバージョン _____

OSの実行モード _____

プログラミング言語およびそのバージョン _____

システムに搭載されている他のボード _____

他のボードのベースI/Oアドレス _____

他のボードのDMAチャネル _____

他のボードの割り込みレベル _____

マニュアルについてのご意見をお聞かせください

ナショナルインスツルメンツでは、弊社製品のマニュアルについてお客様からのご意見をお待ちしております。このような情報は、お客様のニーズを満たす品質の高い製品を提供する貴重な資料として活用させていただきます。

マニュアル名： LabVIEW™ ユーザマニュアル

発行： 1998年7月

製品番号： 321187B-01

マニュアルの完成度、わかりやすさ、構成についてのご意見をお聞かせください。

マニュアルに誤りを見つけられた場合は、そのページ番号を明記し、その誤りの内容をご説明ください。

ご協力ありがとうございました。

ご氏名 _____

役職名 _____

貴社名 _____

ご住所 _____

電話 _____

送付先 〒105-0011
東京都港区芝公園2-4-1
秀和芝パークビルB館5F
日本ナショナルインスツルメンツ株式会社
技術部テクニカル出版課
Fax：03-5472-2977

用語集

接頭語	意味	値
n-	ナノ	10^{-9}
m	ミクロ	10^{-6}
m-	ミリ	10^{-3}

記号

∞　無限。

π　パイ。

Δ　デルタ。偏差。Δx は、ある指標から次の指数まで x の変化値を示します。

1D　一次元。

2D　二次元。

A

A/D　アナログ/デジタル。

ANSI　American National Standards Institute（米国規格協会）。

ASCII　American Standard Code for Information Interchange（情報交換用米国標準コード）。

ATE　Automatic test equipment（自動テスト装置）。

B

BNF　Backus-Naur（バッカスナウル）形式。コンピュータサイエンスにおける言語文法の一般的な表記法です。

C

Case ストラクチャ	入力によって、実行するサブダイアグラムが異なるストラクチャ。制御フロー言語では、IF、THEN、ELSE および CASE 文の組み合わせになります。
CIN	「コードインタフェースノード」の項を参照。
CPU	Central processing unit（中央処理装置）。

D

DAQ	「データ集録」の項を参照してください。
DDE	「ダイナミックデータ交換」の項を参照してください。
DUT	Device under test（テスト中のデバイス）。

F

FFT	Fast Fourier transform（高速フーリエ変換）。
For ループ	サブダイアグラムを一定回数実行する反復ループ構造。 次のような従来のコードに相当します。`For I=0 to n-1, do･･･`

G

G	LabVIEW および BridgeVIEW アプリケーションの開発に使用するグラフィカルプログラミング言語。
GPIB	General Purpose Interface Bus（汎用インタフェースバス）。ANSI/IEEE 規格集 488.1-1987 と ANSI/IEEE 規格集 488.2-1987 に規定されている通信インタフェースシステムの一般的な名称のことです。このバスを開発したヒューレットパッカード社ではこれを HP-IB と呼んでいます。

H

hex	16 進数。基数を 16 とする数。
Hz	ヘルツ。1 秒あたりのサイクル数。

I

I/O	入出力。通信チャネル、オペレータ入力装置、またはデータ集録インタフェースおよびデータ制御インタフェースを使用して、コンピュータシステムとの間でデータを転送すること。
IEEE	Institute for Electrical and Electronic Engineers（米国電子電気技術者協会）。
Inf	無限を意味する浮動小数点のデジタル表示値。

L

LabVIEW	Laboratory Virtual Instrument Engineering Workbench（ラボラトリ仮想計測器エンジニアリングワークベンチ）。G プログラミング言語に基づいたプログラム開発アプリケーションで、テストおよび計測の目的に幅広く使用されています。
LED	Light-emitting diode（発光ダイオード）。

M

MB	Megabyte。メモリのメガバイト数。

N

NaN	「数字ではない」を表す浮動小数点のデジタル表示値。通常は未定義の演算の結果で、log(–1）のように表示されます。
not-a-path	パス制御用にあらかじめ定義された値で、パスが無効であることを意味します。
not-a-refnum	パス制御用にあらかじめ定義された値で、refnum（参照番号）が無効であることを意味します。

O

OPC サーバ	OLE for Process Control。OPC 協会が規定する COM に基づく規格で、デバイスサーバとの相互通信の方法を取り決めています。COM は、Windows の 32 ビット技術です。

R

refnum	関連する VI によって参照できる DDE 会話またはオープンファイルの識別子。

U

UUT	Unit under test（テスト中の装置）。

V

VI	virtual instrument。「バーチャルインスツルメンツ（仮想計測器）」の項を参照。
VISA	「バーチャルインスツルメンツソフトウェアアーキテクチャ」の項を参照。
VI ライブラリ	特定の用途に関連する VI 群を含む特殊ファイル。
VXI	VME eXtensions for Instrumentation（バス）。

W

While ループ	一定の条件が満たされるまでコードの一部を繰り返し実行するループストラクチャ。従来のプログラミング言語の Do ループや Repeat-Until ループに相当します。

あ

アイコン	ブロックダイアグラム上のノードを図式的に表したもの。
アイコンエディタ	VI アイコンを作成するためのインタフェースで、ペイントプログラムのインタフェースに似ています。
アイコンペーン	フロントパネルウィンドウとブロックダイアグラムウィンドウの右上隅にある領域で、VI アイコンを表示します。
アクティブウィンドウ	現在ユーザの入力が可能なウィンドウ。通常は一番上に表示されているウィンドウで、アクティブウィンドウのタイトルバーはハイライト表示となります。ウィンドウをアクティブにするには、アクティブにするウィンドウをクリックするか、ウィンドウメニューから**ウィンドウ**を選択します。

位置決めツール	オブジェクトの移動、選択、サイズ変更に使用するツール。矢印の形をしています。
インプレース実行	メモリを追加して割り当てないでも、関数またはVIがメモリを再利用できる機能。
エラーメッセージ	ソフトウェアまたはハードウェアの動作不能、または受け付けられないデータ入力が行われたことを示します。
オートスケーリング	スケールがプロットした値の範囲に合わせて調整される機能。グラフのスケールでは、この機能によりスケールの最大値と最小値が決定します。
オブジェクト	制御器やノード、ワイヤ、取り込んだピクチャなど、フロントパネルまたはブロックダイアグラム上の項目の総称。
オブジェクトポップアップメニューツール	オブジェクトのポップアップメニューにアクセスするために使用するツール。

か

階層ウィンドウ	VIとサブVIの階層を図で表示するウィンドウ。
階層パレット	パレットおよびサブパレットを含むメニュー。
開発者	「システム開発者」の項を参照。
外部ルーチン	「共有外部ルーチン」の項を参照。
カウント端子	Forループの端子で、カウント端子の値によりForループがサブダイアグラムを実行する回数が決まります。
カスタムPICT制御器とカスタムPICT表示器	標準の制御器および表示器をもとに、ユーザが提供するグラフィックスなどを追加または変更して作成した特殊な制御器および表示器。
カラーツール	前景色および背景色を設定するのに使用するツール。
空の配列	データタイプは定義されていますが、要素が0の配列。例えば、データディスプレイウィンドウに数値制御器はありますが、要素のどれにも値が定義されていない配列は、空の数値配列です。
関数	内蔵されている実行要素で、従来の言語の演算子、関数または文に相当します。
関数パレット	ブロックダイアグラムストラクチャ、定数、通信機能、およびVIを含むパレット。

疑似コード	簡易言語から独立したプログラミングコードの表記法。
キャスト	データ自体の内容を実質的に変えずに、データ要素のタイプ記述子を変更すること。
強制	データ要素の数値表現を変更するため、Gが実行する自動変換。
強制ドット	端子上のグレーのドットで、配線された2個の端子のうち、一方の端子が他方の端子のデータタイプに合わせて変換されたことを表します。
共有外部ルーチン	複数のCINコードリソースで共有できるサブルーチン。
行列	二次元の配列。
クラスタ	数値、ブール、文字列、配列、またはクラスタなど、順序付けられてはいますが、指標付けされていない任意データタイプの要素の集合。クラスタ要素は全部制御器、または全部表示器のいずれかでなければなりません。
クラスタシェル	クラスタ要素が入っているフロントパネルオブジェクト。
グラフ制御器	デカルト平面上にデータを表示するフロントパネルオブジェクト。
繰り返し端子	現時点までに完了した繰り返し回数が入っているForループまたはWhileループの端子。
グリフ	小さなピクチャまたはアイコン。
グローバル変数	実行から次の実行までのデータを保存するために、初期化されないシフトレジスタを使用するサブVI。ローカルメモリを備えていますが、再入力はできません。これらサブVIのコピーが入ったメモリは共有されているため、サブVI間のグローバルデータの受け渡しに使うことができます。
クローン化	制御器やGオブジェクトをコピーすること。<Ctrl>（Windows）、<option>（Macintosh）、<meta>（Sun）または<Alt>（HP-UX）キーを押したまま、制御器またはその他のGオブジェクトをクリックして新しい位置までコピーをドラッグします。
計測器ドライバ	プログラム可能な計測器を制御するVI。
ケース	Caseストラクチャの1つのサブダイアグラム。
現在のVI	フロントパネル、ブロックダイアグラム、またはアイコンエディタがアクティブウィンドウになっているVI。
コードインタフェースノード	CIN。ブロックダイアグラムの特殊ノードで、従来のテキストベースのコードをVIにリンクすることができます。

コネクタ	入力端子と出力端子を含むVIまたは関数ノードの一部。データはコネクタを経由してノード間で受け渡しが行われます。
コネクタペーン	フロントパネルウィンドウの右上隅にあり、VI端子のパターンを表示します。通常はアイコンペーンの下に隠れています。
壊れたVI	コンパイルや実行ができないVI。実行ボタンに壊れた矢印が表示されます。
壊れた実行ボタン	エラーのためVIが実行できないとき、実行ボタンに代わるボタン。このボタンをクリックすると、エラーリストダイアログボックスが開きます。
コンパイル	ハイレベルコードをマシンが実行可能なコードに変換するプロセス。VIを作成または変更後初めて実行する場合、実行前に自動的にコンパイルされます。

さ

最上位VI	VI階層の最も上位にあるVI。この語によってこのVIとそのサブVIを区別します。
サイズ変更ポインタ	オブジェクトの隅に表示されるL字部分で、サイズ変更点を示します。
再入可能実行	1つのサブVIの複数インスタンスに対する呼び出しが、別のデータ記憶域を利用して並行して実行できるモード。
サブVI	別のVIのブロックダイアグラムで使用されるVI。サブルーチンに相当します。
サブダイアグラム	ストラクチャ境界内にあるブロックダイアグラム。
シーケンスストラクチャ	数値順にサブダイアグラムを実行するプログラム制御構造。一般的には、データに依存しないノードを指定した順に強制実行するために使用します。
シーケンスローカル	シーケンスストラクチャのフレーム間で、データを受け渡すのに使用される端子。
次元	配列のサイズおよび配列構成の属性。
システム開発者	実行するアプリケーションソフトウェアの作成者。
実行ハイライト	VIのデータフローを示すためVIの実行を図式的に表示する機能。
自動サイズ調整	入力されたテキストに合わせて、ラベルのサイズを自動的に変更します。

自動指標付け	ループストラクチャがその境界で配列を分解したり組み立てたりする機能。「指標付け使用」を選択した状態で、配列がループに入ると、ループは自動的にそれを 1 次元配列から抽出したスカラや、2 次元配列から抽出した 1 次元配列などに分解します。配列がループを出るとき、ループはその逆の手順でデータを配列に組み立てます。
シフトレジスタ	オプションのループストラクチャのメカニズムで、ループの一つの繰り返しから次の繰り返しへ変数の値を渡すために使用します。
条件端子	ブール値を含むWhileループの端子で、このブール値によってVIが次の繰り返しを実行するかどうかが決定されます。
シンク端子	データを受け取る端子。接続先端子とも呼びます。
人工的データ依存	データフロープログラミング言語において、データの値ではなくデータの到着によってノードの実行がトリガされる状態。
スウィープチャート	オシロスコープの操作を模倣した数値制御器。スコープチャートに似ていますが、スウィープチャートは、ディスプレイを線で区切り、新しいデータと古いデータを区別します。
数値制御器と数値表示器	数値データの操作と表示、または入出力に使用されるフロントパネルオブジェクト。
スカラ	スケール上の一点で表すことのできる数字。配列とは異なり、スカラは 1 つの値です。スカラのブールとクラスタは、それぞれのデータタイプの明示的な単一のインスタンスです。
スクロールツール	ウィンドウのスクロールに使用するツール。
スケール	機械動作、チャート、およびグラフの制御器と表示器の一部で、測定単位を示すために一定の間隔で一連のマークまたは点を付けた部分。
スコープチャート	オシロスコープの動作を模倣した数値表示器。
スタブVI	サブVIのプロトタイプで、それ自体では機能せず、入力と出力がありますが不完全です。VI の設計の初期計画段階で、後で開発する VI の配置場所を確保するために使用します。
ストラクチャ	シーケンス、Case、Forループ、Whileループなどのプログラム制御要素。
ストリップチャート	紙テープに記録するチャートレコーダを模倣した数値プロット表示器で、データをプロットするたびにスクロールします。
スポイトツール	カラーツールで貼り付ける色をコピーするのに使用するツール。
スライダ	スライド制御器および表示器の可動部分。

制御器　対話形式でVIにデータを入力したり、プログラムによってサブVIにデータを入力するためのフロントパネルオブジェクト。

制御器パレット　フロントパネル制御器および表示器を含むパレット。

制御フロー　命令の順序によって実行の順序が決定するプログラミング。C言語やPascal、BASICなど、従来からのテキストベース言語の多くは制御フロー言語です。

接続先端子　「シンク端子」の項を参照。

絶対パス　ファイルシステムの最上位に対する相対的な位置を表すファイルまたはディレクトリパス。

説明ボックス　Gオブジェクト用のオンライン文書を含むダイアログボックス。

センサ　速度、温度、流量などの物理的特性を、電圧や電流出力として表示するデバイス。

操作ツール　制御器にデータを入力したり、制御器の操作に使用するツール。指をさす手の形をしています。

ソース端子　データを送り出す端子。

ソリューションギャラリ　DAQソリューションウィザードに含まれるオプション。ソリューションギャラリでは一般的なデータ集録アプリケーションが様々なカテゴリに分類され、この中から選択できます。

た

ダイアログボックス　アプリケーションで、コマンドを実行するために、さらに情報が必要な場合表示されるウィンドウ。

ダイナミックデータ交換　DDE。ユーザの操作や監視なしに行われるアプリケーション間のデータの移動。

タイプデスクリプタ　「データタイプデスクリプタ」の項を参照。

多形性　異なる表現、タイプ、またはストラクチャのデータに応じてノードが自動的に調整できる機能。

端子　データの受け渡しが行われるノード上のオブジェクトまたは領域。

チェックボックス　選択や選択の解除ができるダイアログボックス内の小さな正方形。通常チェックボックスは複数のオプションが設定できる場合に使用され、2つ以上のチェックボックスを選択することが可能です。

チャート	「スコープチャート」、「ストリップチャート」、および「スウィープチャート」の項を参照。
ツール	特定の操作を実行するために使用する特殊カーソル。
ツールバー	VIの実行やデバッグに使用するコマンドボタンの入ったバー。
ツールパレット	フロントパネルやブロックダイアグラムのオブジェクトの編集およびデバッグに使用するツールの入ったパレット。
定数	「ユニバーサル定数」および「ユーザ定義定数」の項を参照。
ディレクトリ	ファイルを扱いやすいグループに分けるストラクチャ。ディレクトリとはファイルの位置を示すアドレスのようなもので、中にはファイルまたはファイルを含むサブディレクトリが入っています。
データ依存	データフロープログラミング言語で、別のノードからデータを受け取るまでノードが実行できない状態。「人工的データ依存」の項も参照。
データ記憶形式	メモリ上に格納されるデータの配置と表記の方法。
データ集録	DAQ。データを集録するプロセス。一般的にはA/Dまたはデジタル入力プラグインボードから集録。
データタイプ	情報の形式。BridgeVIEWでは、タグ構成で使用できるタイプはアナログ、離散式、ビット配列、および文字列です。LabVIEWでは、数値、配列、文字列、クラスタといったデータタイプが、ほとんどのすべての関数に使用できます。
データタイプデスクリプタ	データタイプの識別コード。データの格納や表記に用いられます。
データフロー	実行可能なノードで構成されるプログラミング体系。ノードは必要な入力データをすべて受け取ったときだけ実行され、実行されると自動的に出力を生成します。G言語はデータフローシステムです。
データロギング	データを集録すると同時にそれをディスクファイルに保存すること。Gのファイル入出力関数はデータのロギングが可能です。
データログファイル	ファイル作成時に、指定した1つの任意データタイプの一続きのレコードとして、データを保存するファイル。データログファイルのレコードはすべて1つのタイプでなくてはなりませんが、そのタイプが複合的であってもかまいません。例えば、各レコードを文字列や数値、配列を含むクラスタとして指定することができます。

テーブル駆動の実行　実行方法の一つで、各タスクはWhileループにおけるCaseストラクチャ内の個別のケースの処理を行います。順序はケース番号の配列順に指定されます。

ドライブ　aからzまでの文字にコロン（:）がついたもので、論理ディスクドライブを指します。

トンネル　ストラクチャ上のデータ入力端子または出力端子。

な

ニーモニック　整数値に対応する文字列。

二次元　2つの次元を持つこと。例えば複数の行と列を持つ配列など。

ノード　プログラム実行要素。従来のプログラミング言語の、文、オペレータ、関数そしてサブルーチンに似ています。ブロックダイアグラムでは、関数、ストラクチャ、およびサブVIが含まれます。

は

バーチャルインスツルメンツ（仮想計測器）　VI。グラフィカルプログラミング言語Gで書かれたプログラムで、実際の計測器の外観や機能を模倣しているためこう呼ばれます。

バーチャルインスツルメンツソフトウェアアーキテクチャ　GPIB、VXI、RS-232、その他の計測器を制御するための単独インタフェースライブラリ。

配線ツール　ソース端子とシンク端子間のデータパスを定義するのに使用するツール。

バイトストリームファイル　一連のASCII文字またはバイトでデータを保存するファイル。

配列　一定の順序で並べられ、指標付けされている、同一タイプのデータの集合。

配列シェル　配列が入っているフロントパネルオブジェクト。指標ディスプレイ、データオブジェクトウィンドウ、およびラベル（オプション）で構成され、様々なデータタイプを受け付けることができます。

ハウジング　スライダとスケールのあるフロントパネルの制御器および表示器の固定部分。

波形チャート　一定の速度でデータポイントをプロットする表示器。

パレット　使用可能なオプションを表すアイコン表示。

ハンドル	ブロックメモリのポインタに対するポインタで、配列と文字列の参照を行います。文字列の配列は、文字列に対するハンドルが入っているメモリブロックを参照するハンドルです。
バンドルノード	様々なタイプの構成要素からクラスタを作成する関数。
凡例	チャートまたはグラフが所有するオブジェクトで、そのチャートまたはグラフにプロット名とプロット方式を表示します。
ピクセルマップ	画像を記憶するための標準フォーマットで、各ピクセルを色の値で表します。ビットマップは、ピクセルマップを白黒で表示したものです。
非同期実行	複数の処理がプロセッサ時間を共有するモード。例えば、他の処理がデバイス入出力中割り込み待ちの間、またはクロックのタイムアウト待ちの間に、1つの処理を実行します。
表記法	数値データタイプのサブタイプ。符号付きと符号なしバイト、ワード、倍長整数のほか、単精度、倍精度および拡張精度の浮動小数点数があります。
表示器	出力を表示するフロントパネルオブジェクト。
表示不能文字	改行やタブなど、表示できないASCII文字。
ヒントラベル	オブジェクト名、制御器名、あるいは端子名を表示するテキストラベル。
ファイルrefnum（参照番号）	ファイルを開く場合のGに関係のあるファイルの識別子。ファイル参照番号を使って、開いているファイルに対し関数やVIが操作を実行するよう指定します。
ファイルの終わり	EOF。ファイルの初めに対するファイルの終わりの文字オフセット（すなわち、EOFはファイルのサイズです）。
ブール制御器とブール表示器	ブール（TRUEまたはFALSE）データの操作および表示に使用するフロントパネルオブジェクト。スイッチ、ボタン、LEDなど様々な種類があります。
フォーミュラノード	テキストとして入力された式を実行するノード。ブロックダイアグラムの形で作成すると手間がかかる長い式に特に便利です。
プラットフォーム	コンピュータシステムの基盤となるハードウェアもしくはソフトウェア。
フリーラベル	フロントパネルまたはブロックダイアグラム上にあり、他のどのオブジェクトにも属さないラベル。
プルダウンメニュー	メニューバーからアクセスするメニュー。通常プルダウンメニューのオプションは一般的なものです。

ブレークポイント	サブVIが呼び出されたとき実行が停止する点。ブレークポイントを設定するには、**ツール**パレットのブレークポイントツールを使って、VIやノード、あるいはワイヤをクリックします。
ブレークポイントツール	VI、ノード、またはワイヤにブレークポイントを設定するために使用するツール。
フレーム	シーケンスストラクチャのサブダイアグラム。
プローブ	VIの中間値をチェックするデバッグ機能。
プローブツール	ワイヤ上にプローブを作成するのに使用するツール。
プログラム印刷	実行後、VIフロントパネルを自動的に印刷する機能。
ブロックダイアグラム	プログラムまたはアルゴリズムを図式的に説明または表記したもの。Gのブロックダイアグラムは、VIのソースコードです。ノードという実行可能なアイコンと、ノード間でデータをやり取りするワイヤで構成されています。
プロット	データ配列を、グラフまたはチャートのいずれかで図式的に表すこと。
フロントパネル	VIの対話式ユーザインタフェース。実際の計測器のフロントパネルを模倣したもので、スイッチ、スライド、メータ、グラフ、チャート、ゲージ、LEDまたはその他の制御器および表示器で構成されています。
平坦化されたデータ	文字列に変換された何らかのタイプのデータで、通常ファイルに書き込むためのものです。
ヘルプ	Windowsアプリケーションの使い方を説明するオンラインドキュメント。**ヘルプ**メニューは特定のヘルプトピックを表示します。<F1>を押すとヘルプトピックの一覧が表示されます。
ヘルプウィンドウ	関数またはサブVIの端子名および位置、制御器や表示器の説明、ユニバーサル定数の値、制御器属性の説明およびデータタイプを表示する特殊なウィンドウ。
変換	データ構成要素のタイプを変更するプロセス。
ポップアップ	オブジェクト上でマウスの右ボタンをクリックする(Windows)か、コマンドクリック（Macintosh）して、そのオブジェクト専用のメニューを呼び出すこと。
ポップアップメニュー	オブジェクトを右クリック（Windows）、またはコマンドクリック(Macintosh)（することによりアクセスできるメニュー。そのオブジェクト特有のメニューオプションが付属しています。

ま

マーキー	選択したオブジェクトのまわりを囲む破線で、移動することができます。
メニューバー	アプリケーションのメインメニューの名前を一覧表示する水平のバー。メニューバーは、ウィンドウのタイトルバーの下に表示されます。メニュー(およびコマンド)の中には多くのアプリケーションに共通するものがありますが、各アプリケーションのメニューバーはそれぞれ異なっています。
モジュール式プログラミング	相互に交換性のあるコンピュータルーチンを使用するプログラミングのタイプ。
文字列制御器と文字列表示器	テキスト処理や表示、または入出力に使用するフロントパネルオブジェクト。

や

ユーザ定義定数	設定した値を送り出すブロックダイアグラムオブジェクト。
ユニバーサル定数	特定の ASCII 文字や pi など、標準の数値定数を送出する編集不可能なブロックダイアグラムオブジェクト。

ら

ラベリングツール	ラベルを作成して、テキストウィンドウにテキストを入力するために使用するツール。
ラベル	フロントパネルやブロックダイアグラム上の他のオブジェクトや領域に名前を付けたり、説明を加えたりするために使用するテキストオブジェクト。
リアルタイム	対象とする物理的処理が進行する実時間内に演算処理を行い、その処理の結果を対象とする物理的処理の制御に使用する演算処理能力に関する用語。
リストボックス	コマンドに対する選択肢をすべて一覧にして表示するダイアログボックス内のボックスで、例えばディスク上のファイル名などがあります。通常、リストボックスから項目を選び OK をクリックします。リストボックスに入りきらない数の選択肢が存在する場合は、縦のスクロールバーが表示されます。リストの最初の項目の隣りにある下向きの矢印を選択すると、リストボックスの残りの部分が表示されます。
リング制御器	32 ビット整数を一連のテキストラベルまたはグラフィックスに結びつける特殊な数値制御器。0 から始まり順次増分します。

連続実行	オペレータが停止しない限りVIの実行を繰り返す実行モード。**連続実行**ボタンをクリックすると連続実行モードが実行可能になります。
ローカル変数	VIのフロントパネル上の制御器や表示器へ、読み込みや書き込みができるようにさせる変数。

わ

ワイヤ	ノード間のデータパス。「データフロー」の項も参照。
ワイヤセグメント	水平または垂直な一本のワイヤ。
ワイヤ接合点	3本以上のワイヤセグメントの結合点。
ワイヤブランチ	接合点から他の接合点まで、端子から次の接合点まで、あるいは端子間に接合点がない場合は端子から他の端子まで、そのワイヤセグメント全体を含むワイヤの部分。

索引

数値

A

B

C

D

E

F

L

M

N

O

P

R

S

T

U

V

W

X

Y

あ

い

う

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

に

ね

の

は

ひ

ふ

へ

ほ

ま

む

め

も

ゆ

よ

ら

り

る

れ

ろ

わ