ユーザーズマニュアル

RELEASE 4.2.2

SATELLITE を使ってみよう!

Neuroinformatics Laboratry

SATELLITE: ユーザーズマニュアル

RELEASE 4.2.2

Neuroinformatics Laboratry

目次

第 1 章 SATELLITE とは

1.1. システム解析支援環境 : SATELLITE

脳・神経系に代表される未知システムを対象とする解析支援環境には，解析機能として通常のシステム解析手法はもちろん，解析を通じて生じた新しいアプローチの導入に即座に対応するシステム使用の設計が必要です．また，試行錯誤的処理手法の模索が中心となる研究であり，プログラミングをはじめ，データフォーマット変換といった付随的作業をなくし，研究者の試行を中断させない親和的なマン・マシン・インタフェースを備えていなければなりません．

こうした思想をもとに開発されたのが，システム解析支援環境 ： SATELLITE (System Analysis Total Environment for Laboratory - Language and InTeractive Execution) です．

SATELLITE は，会話環境および C 言語に似た言語体系 (SATELLITE 言語) を提供するシェルを中心に，信号処理やシミュレーションなど計 248 種類に及ぶ解析コマンドを機能別にまとめたシステム・モジュールによって構成されます．

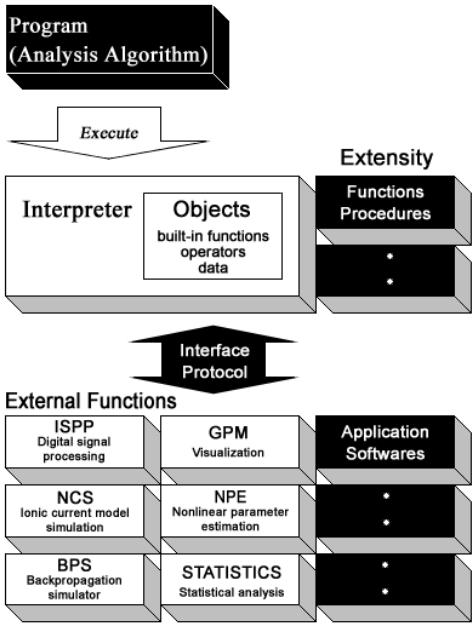


図 1.1. SATELLITE 言語処理系の概要

ユーザは，シェルが提供する対話環境から，組み込み関数，およびシステムモジュールの任意の関数を組み合わせて，解析・シミュレーションを進めることができます．

1.2. システム・モジュール

系統的手法の存在しない未知システムの解析では，多方面に亙る知見を試行錯誤的に適用し，新しい理論・概念を導いていかねばなりません．そこには，基本となる信号処理理論をはじめ，シミュレーション技術，パラメータ推定法など，様々な解析アルゴリズムをデータベースとして備え，研究者の試行を中断させずに新しい方法論を容易かつ迅速に導入・評価するための柔軟な支援環境が必要です．

こうしたことから，SATELLITE では，解析の対象や手法によって機能をシステム・モジュールとして分類し，解析アルゴリズムの体系化を図っています．システム・モジュールとしては，システム解析の基本となるディジタル信号処理をはじめ，シミュレーションやパラメータ推定法などがあります．

1.2.1. システム関連モジュール ： SYSTEM

SYSTEM は，データの切り出し，データの連結，データ形式の変更など，様々なデータの編集機能を中心に，データバッファのモニタリングコマンド，データの最大，最小値を求める関数，データバッファのヘッダ情報を調べる関数などを集めたモジュールです.

1.2.2. ディジタル信号処理パッケージ ： ISPP

ISPP (Interactive Signal Processing Package) は，SATELLITE のシステムモジュールの中核をなすものです.このモジュールには，データ補間，ウィンドウ等の前処理や，FFT，線形予測によるスペクトル推定，フィルタリング，ケプストラム解析等，ディジタル信号処理の方法論を網羅する処理機能が関数として実装されています.こうした様々な関数は，全て複数のデータ格納領域 (バッファ) またはデータファイルを対象として処理を実行することができ，関数を組み合わせて処理を記述することによって，計測されたデータの信号処理，統計処理をはじめ多角的なデータ解析を行うことができます.

1.2.3. グラフィックシステム ： GPM

処理結果の検討時に，データを単なる数値の羅列として出力したのでは，その特性を対極的に把握することは困難です．GPM (Graphic Package Module) はグラフィック機能として通常のグラフ表示はもちろん，データの重ね書き，等高線表示，3 次元表示，カラーマップ表示などを提供しています.また，グラフィック機能はデバイスに依存しない構成となっているため，画面イメージと同一な印刷情報を出力することができます.

1.2.4. バックプロパゲーション・シミュレータ ： BPS

BPS (Back Propagation Simulator) は，誤差逆伝搬 (BP) 学習アルゴリズムを用いたニューラルネットワークシミュレータです．コマンドによりネットワーク構造や内部状態の設定， データファイルの管理などの各種条件が指定可能で，大規模なニューラルネットのシミュレーションが容易に行なえます．また，複数の重み初期値設定アルゴリズムや学習アルゴリズムを搭載し，各種アルゴリズムの評価，開発を支援すると共に，学習過程のグラフィックトレース機能などにより，学習の進行状況を瞬時に把握できる環境を提供しています．その結果，学習によって得られた内部表現の解析，学習状態や汎化の過程，あるいは，ローカルミニマムの問題など複数の学習試行に対する統計的処理結果から多角的に解析でき，計算論的神経科学の要請に基づいた解析環境を実現しています.

1.2.5. 神経回路シミュレータ ： NCS

NCS (Neural Circuit Simulator) は，神経細胞における情報の受容，処理，伝達の基本であるイオン電流特性を記述した細胞モデルから，ネットワークとしての神経回路のモデル化，シミュレーション解析を支援するシミュレータです．電気生理実験に対応した網膜電位固定，電流注入シミュレーションをはじめ，システムとしての神経回路の挙動のイオン電流レベルの機能メカニズム解析に威力を発揮する仕様となっています．さらに，ダイナミックスを含めたリカレントニューラルネットワークのシミュレーションなど新しい学習アルゴリズムの検証，開発にも有効な環境を提供するものです.

1.2.6. 非線形パラメータ推定システム ： NPE

NPE (Nonlinear Parameter Estimation Module) は，最適化法を利用して最適化問題の解を求めるためのシステムです．装備されている最適化法には，Simplex 法，BFGS 法，DFP 法，SSVM 法，共役勾配法があり，用途に応じて選択することができます．また，統計学，数値解析の知識は特に必要なく，最適化法の特徴を知るだけで容易に利用できるような仕様になっているため，実験データからモデルパラメータを推定したり，要求された条件を満たすフィルタ，制御系，電気回路の設計など，モデリングにおいて大きな力を発揮します.

1.2.7. データ変換モジュール ： DCM

DCM (Data Conversion Module) は，様々なデータ解析ソフトウェアやシミュレータ間と SATELLITE 間とのデータ変換を行うシステムです．これを利用することにより，各システムの特徴を生かした処理を行うことができます.

1.2.8. 統計解析パッケージ ： STATISTICS

STATISTICS は，様々な標本入力に対して統計処理を行い結果を算出するシステムです．提供されている機能は 2 種類に大別され，主に一標本の入力に対して一つの統計量を出力する関数，および，複数の標本入力に対して検定を行い結果を出力する関数で構成されています．

1.3. 動作環境

1.3.1. 対応ハードウェアおよび OS

SATELLITE は，次に示すハードウェアと OS に対応しています．

- PC/AT 互換機 (Windows98, Windows2000, Windows XP)
- PC/AT 互換機 (FreeBSD 4.7 以降, Linux kernel 2.4 以降)
- Apple Machintosh (MacOS X 10.2 以降)
- その他 POSIX ベースの OS (動作は未検証)

1.3.2. ウィンドウ環境

SATELLITE を Windows 以外の環境で利用する場合，以下のシステムが必要となります．

- X Window System Version 11 Release 6 以降

1.4. インストール

提供されている配布パッケージは，「Windows 用インストーラ実行形式」 と 「ソースコード一式」の 2 種類です．それぞれ，http://satellite.sourceforge.jp/download.html からダウンロードできます．

1.4.1. Windows

Windows 用には，実行形式のインストーラパッケージ `satellite-win32-4.2.2.exe` が用意されています．

ダウンロード後，ダブルクリックすることによりセットアップウィザードが起動します．

図 1.2. セットアップウィザードの様子

セットアップウィザードのメニューに従い操作を進めていくことで，インストールが完了します．

1.4.2. MacOS X/Linux/FreeBSD

MacOS X/Linux/FreeBSD などの環境では，ソースコードからコンパイルを行う必要があります.

1. ソースコード `satellite-4.2.2.tar.gz` をダウンロードします.
2. ダウンロードしたソースコードを展開します.

```
% tar zxvf satellite-4.2.2.tar.gz
```

3. 展開されたディレクトリに移動し，configure スクリプトを実行します.

```
% cd satellite-4.2.2
% ./configure
```

4. コンパイルします.

```
% make
```

5. 管理者権限を用いてインストールします.

```
% sudo make install
```

第 2 章 SATELLITE 言語とシェルの機能

2.1. 概要

脳・神経系などの未知のシステムの構造，機能を解明するには，信号解析，モデリング・シミュレーション等が必要です．これらを容易に扱うことができるように，数式処理と文章作成の統合システムである Mathematica，計測制御とデータの収録，解析，および表示を行うための LabVIEW，データ解析とビジュアライゼーションのための AVS など，様々なソフトウェアシステムが整備されてきています．しかし，これらのソストウェアシステムを用いて，一貫した処理を行う場合，他のシステムとのデータ変換が必要であり，全体の効率が著しく低下してしまいます．一方，SATELLITE 言語では，シミュレータや信号処理パッケージなどのアプリケーション・ソフトウェアを SATELLITE 言語の外部関数として位置付け，API (Application Program Interface) 仕様に沿って構成することで，全く異なるアプリケーション・ソフトウェアであっても，一貫した処理を行うことが可能になります．

また，生体システムにおいては，(多次元) 時系列やマトリックスのようにいくつかの要素からなるデータ集合が解析対象となります．SATELLITE 言語では，こうした時系列データを扱えることが特徴です．さらに，実数型，文字列型等のデータの型に関わらず，同一の処理をデータに施すことも可能です．

SATELLITE 言語の処理系はインタプリタであり，計算機と対話を行いながら処理 (解析) を行うことができます．プログラムは，端末またはファイルから読み込まれると，いくつかのフェーズを経てスタックマシンのコードに変換されます．スタックマシンコードは，スタックマシンにより実行されます．従って，for 文，while 文等の繰り返し処理や関数等は，幾分高速に実行されます．内部構成を 図 2.1.「処理系の内部構成」 に示します．

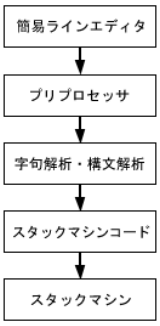


図 2.1. 処理系の内部構成

本処理系は，プログラムが構文規則どおりに記述されていれば受理し，スタックマシンコードに変換・実行します．しかし，エディタの使用，ファイル名の確認，ディレクトリの移動等，外部コマンドを使用したい状況が頻繁に起こることは明らかです．したがって，行頭に現れるトークンが未定義で，かつ代入文でない限り，外部コマンドとして解釈して処理を行います．

2.2. 起動と終了

2.2.1. 起動方法

- Windows
  - IDE 環境

    「スタート」メニュー,「 SATELLITE 4.2.2 」メニューと選択し,「 SATELLITE IDE 」 を選択して起動します．

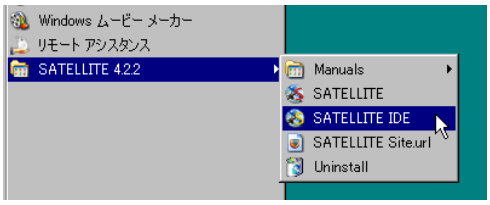


図 2.2. Windows IDE 環境の SATELLITE 起動

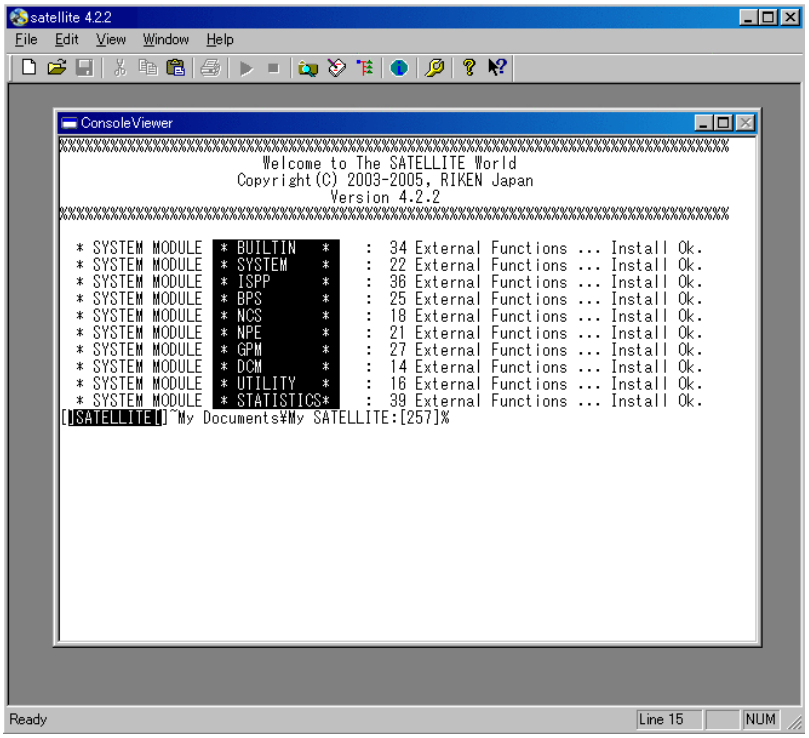


図 2.3. Windows IDE 環境の SATELLITE 起動後

- コマンドプロンプト

  「スタート」メニュー,「SATELLITE 4.2.2」メニューと選択し,「SATELLITE」を選択して起動します.

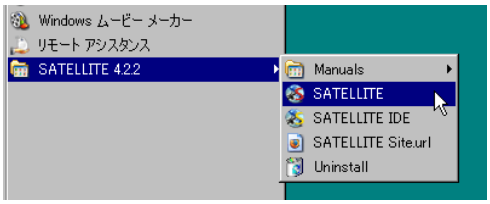


図 2.4. コマンドプロンプト環境の SATELLITE 起動

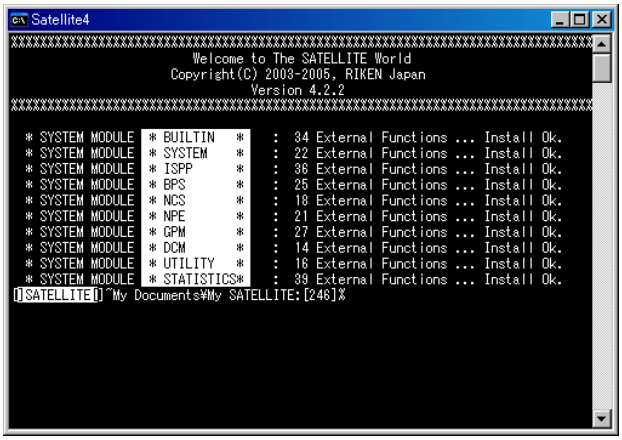


図 2.5. コマンドプロンプト環境の SATELLITE 起動後

- MacOS X/Linux/FreeBSD

  以下のコマンドを入力することで，SATELLITE を実行することができます.

  ```
  % /usr/local/satellite4/bin/sl4
  ```

  SATELLITE 起動後の様子を 図 2.6. 「SATELLITE 起動後の様子」 に示します.

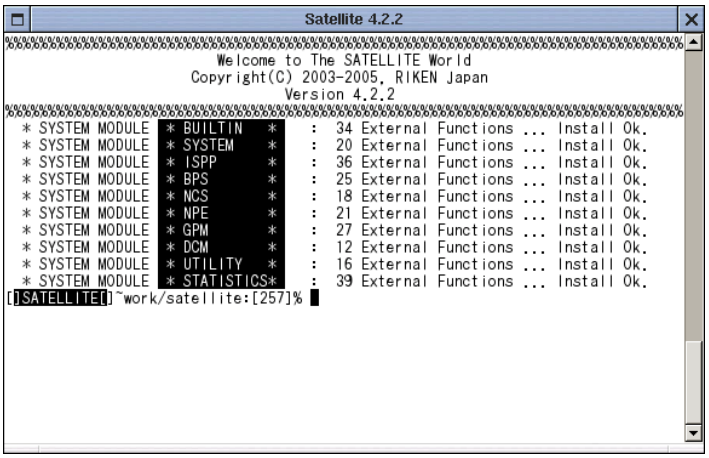


図 2.6. SATELLITE 起動後の様子

2.2.2. 終了方法

本処理系は以下に示すように "exit" とタイプすることにより終了することができます．また，行頭で **Ctrl-D** (CTRL キーと D を同時に押す) を入力によっても終了できます.

```
[]SATELLITE[]~/home/demo:[1]% exit
[]SATELLITE[]~/home/demo:[2]% ^D
```

2.2.3. 設定ファイル

起動時には，システムが用意したシステムリソースファイル `rc.sl` が読み込まれたあとに，セットアップファイル `setup.sl` が自動的に読み込まれます．このとき，セットアップファイルはユーザ設定ディレクトリ以下にユーザセットアップファイルを配置することにより，ユーザが内容を自由に変更することができます．一般的には，`rc.sl` にはシステムモジュールの定義，`setup.sl` にはユーザモジュールの定義・読み込み，エイリアス，サンプリング周波数，よく使用する関数定義，常に同じ意味で使用する変数定義などを記述します.

また，終了時には，クリーンファイルが実行され，その後，システムモジュールの終了関数の実行，システムコモンエリアの開放，システムパラメータエリア (テンポラリディレクトリ) の破壊，全ての子プロセスに対する終了シグナルの発信等が行われた後，コマンド実行履歴がセーブされます.

各設定ファイルの配置場所は以下の通りです.

- システムリソースファイル

  Windows : %INSTDIR%[1]\etc\rc.sl

  MacOS X/Linux/FreeBSD : ${prefix}[2]/etc/satellite4/rc.sl

- システムセットアップファイル

  Windows : %INSTDIR%/etc/setup.sl

  MacOS X/Linux/FreeBSD : ${prefix}/etc/satellite/setup.sl

- ユーザセットアップファイル

  Windows : Document and Settings\%USERNAME%[3]\Application Data\SATELLITE4\setup.sl

  MacOS X/Linux/FreeBSD : ~/.sl4rc/setup.sl

- システムクリーンファイル

  Windows : %INSTDIR%/etc/setup.sl

  MacOS X/Linux/FreeBSD : ${prefix}/etc/satellite/clean.sl

- ユーザクリーンファイル

  Windows : Document and Settings\%USERNAME%\Application Data\SATELLITE4\clean.sl

  MacOS X/Linux/FreeBSD : ~/.sl4rc/clean.sl

2.2.4. プログラムの引き数

SATELLITE 起動時に引き数を指定することにより，いくつかの挙動を変更できます．以下に SATELLITE 起動時の引き数を示します.

- - ： 標準入力 (端末) からプログラムの読み込みを行います.
- -rc ： システム rc ファイルを読み込みません.
- -setup ： ユーザ setup ファイルを読み込みません.
- -clean ： ユーザ clean ファイルを読み込みません.
- -help ： 起動オプションの説明を表示します.

また，ユーザセットアップファイルの他に自動的に読み込ませたいファイルがあれば

```
% sl4 setup2.sl -
```

とします．引数の最後に "-" がなければ標準入力 (端末) からの読み込みは行われず実行終了と同時に処理系も終了します.

[1] インストーラで指定したディレクトリ
[2] configure 実行時に --prefix で指定したディレクトリ
[3] ログオンユーザ名

2.2.5. 環境変数

SATELLITE で利用される環境変数を以下に示します.

- SL_HOME

  この環境変数で指定されたディレクトリをホームディレクトリとし，起動時に読み込まれる設定ファイルの格納場所などをこのディレクトリから検索します．この環境変数が指定されていなければ，SATELLITE は，システムの提供するホームディレクトリ 環境変数 HOME を利用します．また，Windows では，SL_HOME や HOME が設定されていない場合 My Documents\My SATELLITE を使用します.

- SL_TEMP

  この環境変数で指定されたディレクトリに作業ディレクトリ (SATELLITE-Langxxxx) を作成し，実行中に作成される一時ファイルの格納場所に利用します．この環境変数が指定されていなければ，SATELLITE は，システムの提供するテンポラリディレクトリを使用します.

2.3. 操作方法

2.3.1. プロンプト

インタプリタは，ユーザのプログラム入力を催促するためにプロンプトを表示します．プロンプトには，以下に示すような SATELLITE 言語のプロンプトであることを明示する文字とカレントディレクトリ名，行番号が含まれます．カレントディレクトリ名はプロンプトが長くなるのを防ぐために最後の 2 つのパスだけ表示されます．完全なパス名でないときには，パス名の前に "~" がつき，それ以外は絶対パスを示すために "/" からはじまるパス名が使用されます．例えば,

```
[]SATELLITE[]~/home/demo:[1]% cd Tex
[]SATELLITE[]~/home/demo/Tex:[2]%
```

です．また，プログラムが 1 行で終らず継続する場合には "+" のプロンプトで催促されます．例えば,

```
[]SATELLITE[]~/home/demo:[3]% n=0
[]SATELLITE[]~/home/demo:[4]% for( i = 0; i < 10; i++) {
```

と入力した場合,

```
+
```

と表示されます．この時,

```
n = n + 1 }
```

と入力することで，式が評価され，処理が終了します.

2.3.2. エディット機能

端末からの対話的なプログラム入力においては，文字の削除，挿入等を処理するマクロライン・エディタが快適な環境を提供します.

このラインエディタは，内部にエディットバッファを持っており，編集作業は全てこのバッファ内で行われます．エディットバッファは通常ユーザが入力した文字列と一致し，エディット行 (プロンプトより後ろ) に表示されます.

●エディット行の編集

本処理系においては対話処理を円滑にすすめるため，GNU-Emacs ライクな操作性をもつマイクロ・ラインエディタを装備しています．すなわち，エディット行は常にインサートモードになっており，**^F**，**^B**，**^A**，**^E** によりカーソルを移動し編集できます．また，**DEL**，**^H** (**BS**)，**^K** により文字の削除等を行うことができます．但し，^ はコントロールキーを押した状態であることを示しています．例えば,

```
[]SATELLITE[]~/home/demo:[1]% n=0
```

を入力した時，カーソルは 0 の右隣にあります．ここで，**^H** を入力することにより，0 が消去され，カーソルはひとつ左に移動します．すなわち，0 が存在した位置です．一方，**^B** を入力することにより，0 は消去されずに，カーソルはひとつ左に移動します．また，**^A** を入力することにより，行頭すなわち n の位置にカーソルが移動します．このように，GNU-Emacs ライクな操作を行うことが可能です．キーバインドの一覧を 表 2.1. 「キーバインド一覧」 に示します．

表 2.1. キーバインド一覧

イベント	キーバインド
set mark	^@ (Ctrl-SPACE)
begining of line	^A
backward char	^B, ←
interrupt	^C
delete char	DEL
end of file	^D
file listup	^D
end of line	^E
forward char	^F, →
backward delete char	^H, BS
newline	^J, ^M
kill line	^K
search next history	^N, ↓
search prev history	^P, ↑
transpose chars	^T
yank kill	^Y
kill region	^W
filename completion	TAB
command completion	TAB

●ヒストリの検索

端末から入力されたプログラムは，ヒストリとしてバッファに記録されます．**^P** によりヒストリバッファをわたってエディットバッファへヒストリをコピーすることができます．また，**^N** は，ヒストリバッファを順方向へ検索します．ヒストリバッファからエディットバッファにコピーされたプログラムは，自由に編集，実行することができます．

例えば，

```
[]SATELLITE[]~/home/demo:[2]% n = 0
[]SATELLITE[]~/home/demo:[3]% j = 0
```

を入力したとします．ここで，**^P** を入力することにより，

```
[]SATELLITE[]~/home/demo:[4]% j = 0
```

を表示されることができます．再び，**^P** を入力することにより，

```
[]SATELLITE[]~/home/demo:[5]% n = 0
```

を表示させることができます．また，ここで，**^N** を入力することにより，

```
[]SATELLITE[]~/home/demo:[6]% j = 0
```

を表示させることができます.

あらかじめエディットバッファに文字列が入力されている場合は，先頭がその文字列とマッチするヒストリだけ呼び出されます．例えば,

```
[]SATELLITE[]~/home/demo:[7]% n = 0
[]SATELLITE[]~/home/demo:[8]% j = 0
```

を入力したとすると，ここで，^**P** を入力することにより

```
[]SATELLITE[]~/home/demo:[9]% j = 0
```

が表示されます.

●ファイル名，キーワードの補完

関数を入力する際に，エディットバッファの先頭において文字列を入力したあと，**TAB** を入力すれば，文字列に一致する実行可能な関数名もしくは外部コマンド名を補完できます．このとき，補完すべき残りの文字列に対して複数の選択肢がある場合，一致するところまでが端末に表示されます.

```
[]SATELLITE[]~/home/demo:[10]% ii[TAB]
[]SATELLITE[]~/home/demo:[11]% iir
```

このとき，^**D** を入力すれば，残りの補完対象の選択肢一覧が表示されます.

```
[]SATELLITE[]~/home/demo:[12]% iir^D
iir()     iirmake()
```

引き続き，ユニークに一致する続きの文字を入力したあと，再度 **TAB** を入力すれば，補完が完了します.

```
[]SATELLITE[]~/home/demo:[13]% iirm[TAB]
[]SATELLITE[]~/home/demo:[14]% iirmake
```

●ファイルのリストアップ機能

文字の入力途中に ^**D** を入力すれば，ファイル名をリストアップすることができます．この機能は，プログラムを記述している途中でのファイル名の確認や，ディレクトリ変更 (cd コマンド) を利用する時に有効です．例えば,

```
[]SATELLITE[]~/home/demo:[15]% cd /home/okumura/TeX/
```

と入力したあと，/home/okumura/Tex/ の下にあるサブディレクトリ名が知りたいときには，^**D** を入力することにより，以下に示すように文字列の入力を中断する事なく確認することができます.

```
[]SATELLITE[]~/home/demo:[16]% cd /home/okumura/TeX/^D
DSP/            report1.tex     report2.tex
report3.tex     RETINA1/        RETINA2/
work1.tex       work2.tex       work2.tex
[]SATELLITE[]~/home/demo:[17]% cd /home/okumura/TeX/
```

この時ディレクトリは "/" が付けられます．また，実行可能なファイルには "*" が，さらに MacOS X/Linux/FreeBSD 環境では，シンボリックリンクには "@" が，ソケットには "=" がそして FIFO (名前付きパイプ) には "|" が，キャラクタデバイスには "%" が，ブロックデバイスには "#" がそれぞれ最後に付けられます．一覧が表示された後，入力中のコマンドが再表示されます.

さらに，ある文字からはじまるファイルの一覧を知ることもできます．ユーザが次のように入力すると,

```
[]SATELLITE[]~/home/demo:[18]% cd /home/okumura/TeX/RE^D
/home/okumura/RETINA1/      /home/okumura/RETINA2/
[]SATELLITE[]~/home/demo:[19]% cd /home/okumura/TeX/
```

このように "RE" から始まる全てのファイルとサブディレクトリが表示されます.

●外部コマンドの呼びだし

本処理系では，行頭に予約語や変数として登録されていないトークンが現れたときには外部コマンドとして取り扱い，UNIX のシェル (Windows ではコマンドプロンプト) と同じ感覚で外部コマンドを扱うことができます．すでに外部のコマンドと同じ名前の変数が登録されている場合には，行頭に相対パス，もしくは絶対パスを付けて明示的にコマンドを実行することで名前の衝突を回避できます.

2.3.3. プリプロセッサ

簡易ラインエディタにより編集された文字列は，次にプリプロセッサへ引き渡されます．ここでは，主に「エイリアス置換」,「SATELLITE 関数のパラメータの読み込み」が行われます.

1. エイリアス置換

   あらかじめエイリアスが定義されている場合に，入力されたコマンドのトークンの置換を行います.

2. 関数パラメータの取得

   SATELLITE 言語の特徴であるインタラクティブに関数パラメータの説明を表示しながら入力を催促する機能です．詳細は 項 2.9.1.「パラメータ補完機能」で述べます.

2.3.4. 算術演算

ここでは，算術演算を行う SATELLITE 言語の操作方法について述べます．例えば，3 と 6 の乗算を行うには，以下に示すように 3*6 を入力することで，乗算結果が表示されます.

```
[]SATELLITE[]~/home/demo:[1]% 3*6
18
```

また，3 を 6 で除算するには，^ P を入力すれば，前回入力した行が,

```
[]SATELLITE[]~/home/demo:[2]% 3*6
```

と表示されるので，^ B および ^ D を用いて "*" を "/" に変更することで，以下に示すように除算結果が表示されます.

```
[]SATELLITE[]~/home/demo:[3]% 3/6
0.5
```

2.4. データの取り扱い

2.4.1. オブジェクトとクラス

実環境から観測されたデータ，もしくはシミュレーション等で得られたデータは，時間的・空間的，又は何らかの変化に伴う多次元の系列です．従って 1 点のみの値が大きな意味をもつことはまれであり，時系列全体を対象にして解析を進めることが多くなります．そこで，本言語処理系では時系列を一つのデータ型 (オブジェクトクラス) とみなし，図 2.7.「時系列オブジェクトの構造及びその関連図」に示すように時系列を空間的なデータとその時間的変化を区別して扱える Series オブジェクトを中心としたデータ構造を提供しています．ここでは，SATELLITE 言語において対象となる実体をオブジェクトと呼びます.

オブジェクトクラスには，多次元時系列配列を効率良く扱える Series クラスを中心に Snapshot，String，Scalar，File の 5 種類があります．それらのクラスは数値と文字列を扱うものに大別されます．数値を扱うオブジェクトには，Scalar，Snapshot，Series，File があり，Scalar をスーパークラスとして列挙順に特性が継承されています．Snapshot は，Scalar の集合であり汎用言語の多次元配列に相当します．また，Series は，Snapshot が時刻順に並んだオブジェクトを表し，Snapshot の 1 次元配列と考えることもできます．File オブジェクトに関しては，メッセージとしてファイル名を受け取り指定された UNIX のファイルシステム上にあるデータを操作します.

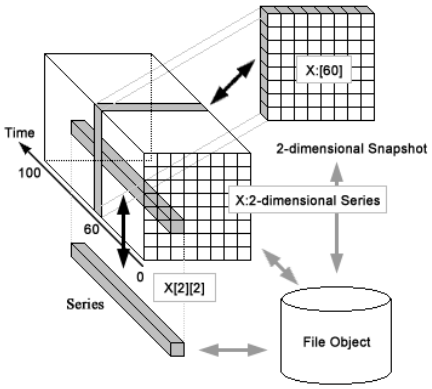


図 2.7. 時系列オブジェクトの構造及びその関連図

また，オブジェクトは，データの他にそのデータに対する処理方法 (Method) も含みます．同じ足し算 (+) でも Scalar，Series，Snapshot，String，File オブジェクトによってその処理方法は異なるのが普通です．例えば，Scalar オブジェクトは図 2.8.「Scalar オブジェクトの足し算の概念図」 に示すように 1 点のみの足し算です．

図 2.8. Scalar オブジェクトの足し算の概念図

一方 Series オブジェクトでは図 2.9.「Series オブジェクトの足し算の概念図」 に示すように時間軸全てについて値を加えなければなりません．

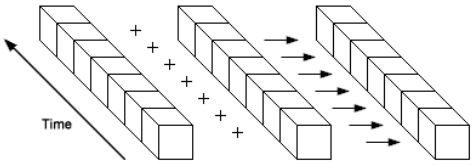


図 2.9. Series オブジェクトの足し算の概念図

このような処理方法の違いをオブジェクト自身に持たせておき，"+" というメッセージのみ送れば，オブジェクト自身がメッセージを解釈し処理を行います．以下に SATELLITE 言語で用いられる 5 種類のオブジェクトクラスについて説明します．

●Scalar クラス

制御変数や Series 要素の操作等を行うためにスカラ値を扱うオブジェクトクラスです (図 2.10.「Scalar オブジェクト」)．処理系内部では，倍精度の浮動小数点数として表現されます．Scalar オブジェクトのデータ操作は以下のように行います．

図 2.10. Scalar オブジェクト

```
[]SATELLITE[]~/home/demo:[1]% k=0.8
[]SATELLITE[]~/home/demo:[2]% k
0.8
```

●Snapshot クラス

Series が空間的な次元に加え，陰に時間軸方向としてもう一つ次元を持つのに対し，Snapshot は空間的な次元しか持たないオブジェクトクラスです (図 2.7.「時系列オブジェクトの構造及びその関連図」)．つまり，時間的

に静止したデータを取り扱うためのオブジェクトクラスであり，Series のサブセットや多次元のマトリックスとして使用することができます．演算は，同じ大きさの Snapshot 同士のみ行えます．Snapshot との混合演算では，Snapshot の全要素について同じ演算が繰り返し行われます．Snapshot オブジェクトのデータ操作の例を示します (図 2.11.「2 次元 Snapshot オブジェクトのデータ操作の例」).

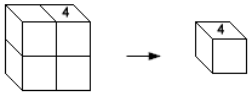


図 2.11. 2 次元 Snapshot オブジェクトのデータ操作の例

まず，以下に示すようにオブジェクトクラスの演算を行います (項 2.4.2.「クラス宣言」 参照).

```
[]SATELLITE[]~/home/demo:[3]% snapshot z[2][2]
```

これより，以下に示すように 2 次元の Snapshot オブジェクトが確保されます.

```
[]SATELLITE[]~/home/demo:[4]% z
[0]:[1]:%    0    0
[0]:[0]:%    0    0
```

このオブジェクトの一部に値を代入するには,

```
[]SATELLITE[]~/home/demo:[5]% z[0][1]=4
```

とします．これより，以下に示すように値が代入されます.

```
[]SATELLITE[]~/home/demo:[6]% z
[0]:[1]:%    0    4
[0]:[0]:%    0    0
```

以下のように入力することで，ある空間のデータを参照することができます.

```
[]SATELLITE[]~/home/demo:[7]% z[0][1]
[0]:[1]:%4
```

●Series クラス

空間的な多次元データの集合が時間軸方向に伸びている様子を表現するオブジェクトクラスです (図 2.7.「時系列オブジェクトの構造及びその関連図」). 空間的に 1 点しかないような Series は，汎用言語では 1 次元配列などで表現される単純な時系列です (図 2.12.「1 次元 Series オブジェクトの概念図」). Series 同士の演算は，空間的なデータ集合 (Snapshot) の大きさが同じときのみ可能です.

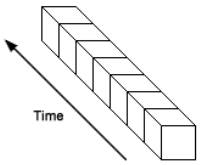


図 2.12. 1 次元 Series オブジェクトの概念図

すなわち，2 次元 Series オブジェクトと 1 次元 Series オブジェクトの演算はできません．また，時間軸方向の長さが異なる場合には，短い方の範囲内で演算が行われ，残りはそのままコピーされます．また，Series と Scalar の混合演算においては，Series 全体に Scalar を巡回させて計算が行われます．Series の大きな特徴は，各要素ごとに繰り返される隠れた反復演算と，"[]"，":[]" 演算子による時系列，空間データの切り出し，書き込みなどの操作機能を持つことです．ここで，Series オブジェクトのデータ操作の例を示します.

```
[]SATELLITE[]~/home/demo:[8]% x=1~7
```

これより，1 次元 Series オブジェクトに 1 から 7 が格納されます (図 2.13.「1 次元 Series オブジェクトのデータ操作の例」). ここで，"~" は交差 1 の数列を生成する数列生成演算子です (項 2.5.「式と演算子」 参照).

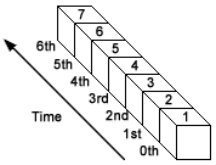


図 2.13. 1 次元 Series オブジェクトのデータ操作の例

```
[]SATELLITE[]~/home/demo:[9] x
[0]:%  1  2  3  4  5
[5]:%  6  7
[]SATELLITE[]~/home/demo:[10]% x:[3]
4
```

により時間軸方向の 3 番目のデータを参照することができます．次に，2 次元オブジェクトのデータ操作の例を示します．以下のようにオブジェクトクラスの宣言を行います (項 2.4.2.「クラス宣言」 参照).

```
[]SATELLITE[]~/home/demo:[11]% series y[2][2]
```

以下に示すように y に値を格納します.

```
[]SATELLITE[]~/home/demo:[12]% y[0][1]=x
```

以下のように入力することで，ある空間の時系列方向のデータを参照することができます (図 2.14.「2 次元 Series オブジェクトのデータ操作の例 1」).

```
[]SATELLITE[]~/home/demo:[13]% y[0][1]
[0]:%  1  2  3  4  5
[5]:%  6  7
```

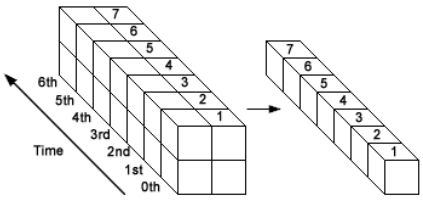


図 2.14. 2 次元 Series オブジェクトのデータ操作の例 1

また，以下のように入力することで，ある時刻の空間的なデータを参照することができます (図 2.15.「2 次元 Series オブジェクトのデータ操作の例 2」).

```
[]SATELLITE[]~/home/demo:[14]% y:[3]
[0][1]:%  0  4
[0][0]:%  0  0
```

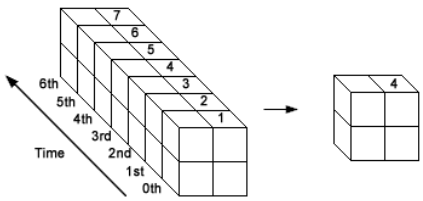


図 2.15. 2 次元 Series オブジェクトのデータ操作の例 2

●File クラス

実験データ，シミュレーション結果等をハードディスク内にファイル形式で，半永久的に保存したいときに使用します．ファイルへのロードおよびストアは，オブジェクトの評価，代入で操作できます．従って，ファイルのフォーマットやデータ型などを気にすることなく他のオブジェクトと同じように操作できます．また，series オブジェクトとの混合演算も可能です．

File クラスは Series とほぼ同じ構造を持ち，多次元データ (Series，Snapshot) をレコード方向に複数格納することができます (図 2.16.「データファイル構造」)．レコードとは Series の時刻に相当し，レコード方向は伸縮自在の長さを持ちます．各レコードに格納するデータは同じサイズでなければなりません．また，File オブジェクトの次元数，インデックスは最初にストアされたオブジェクトに依存し，異なる次元数，インデックスを持つオブジェクトは強制変換された後ストアされます．

データのストアは，File オブジェクトへの代入によって行えます．例えば，

```
[]SATELLITE[]~/home/demo:[15]% $"data.dat":[0] = y
```

は y (Serries または Snapshot オブジェクト) を data.dat のレコード 0 にストアします．

データをロードする場合には，式中に File オブジェクトをそのまま記述すればよく，処理系により自動的に評価されて Series に変換されます．例えば，

```
[]SATELLITE[]~/home/demo:[16]% x = $"data.dat":[0]
```

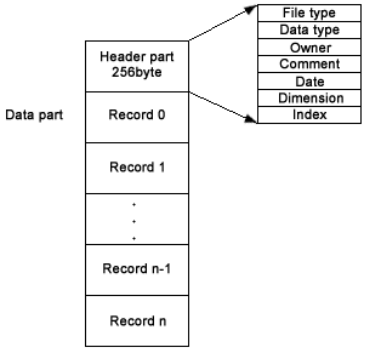


図 2.16. データファイル構造

は x に data.dat のレコード 0 のデータをロードします．また，

```
[]SATELLITE[]~/home/demo:[17]% y = $"data.dat"
```

は y に data.dat の全てのレコードをロードします．x，y は共に Series オブジェクトであり，次元，インデックスは data.dat に依存します．例えばレコードに 2 次元データが格納されている場合には，x は 2 次元，y は 3 次元の Series オブジェクトとなります．

●String クラス

実験データ，シミュレーション結果等は，最終的にデータファイルとして保存されます．ファイル名には，データの属性や通し番号等を含む場合が多く，データを管理する上で基本となる情報です．String オブジェクトは単にファイル名を記憶するだけでなく，作図の際のラベル，プログラムからのメッセージ出力として使用する．String オブジェクトの，連結，削除，繰り返し，分割などは "+"，"-"，"*"，"/" などのメッセージを送ることによって行うことができます．また，文字列はダブルクウォート (") で囲まれます．例えば，文字列と文字列の連結は以下のように "+" を用います．

```
[]SATELLITE[]~/home/demo:[18]% y = "test" + ".dat"
[]SATELLITE[]~/home/demo:[19]% y
test.dat
```

また，文字列から文字列の削除は以下に示すように "-" を用います．

```
[]SATELLITE[]~/home/demo:[20]% y = "test.dat" - ".dat"
[]SATELLITE[]~/home/demo:[21]% y
test
```

文字列を繰り返すには以下に示すように "*" を用います.

```
[]SATELLITE[]~/home/demo:[22]% "ABC" * 4
ABCABCABCABC
```

文字列を分割するには以下に示すように "/" を用います.

```
[]SATELLITE[]~/home/demo:[23]% "A,BC,D,EFG,H,IJK" / ","
[0]%  A  BC  D  EFG  H
[5]%  IJK
```

ここで，"[0]%"，"[5]%" は複数の要素を表示するときのインデックスの先頭を表します.

2.4.2. クラス宣言

本処理系における変数のクラス宣言は，永久的な型の束縛を意味するものではなく，値の定まらないオブジェクトを生成するものです.

Series の宣言では，以下に示すように添え字として，空間的な大きさが指定されなければ 1 次元の時系列を扱うことを表します.

```
[]SATELLITE[]~/home/demo:[1]% series x
```

また，64×64 の空間的な大きさを持つ時系列は以下のように宣言します.

```
[]SATELLITE[]~/home/demo:[2]% series y[64][64]
```

Snapshot の宣言は，以下のように行います.

```
[]SATELLITE[]~/home/demo:[3]% snapshot a[10], b[20][20]
```

この Snapshot の宣言では，空間的な大きさを省略することは出来ません．また，Scalar の宣言は，以下のように行います.

```
[]SATELLITE[]~/home/demo:[4]% scalar i, j, k
```

Scalar は，空間的な大きさを持つことは許されず 1 つの値のみ扱います．また，String の宣言は，以下のように行います.

```
[]SATELLITE[]~/home/demo:[5]% string str, mstr[10]
```

この String の宣言においては，空間的な大きさを指定することも可能です.

2.4.3. オブジェクトクラスの変換

本処理系における変数のオブジェクトクラス (型) の決定は代入時に行われ，右辺のオブジェクトクラスが用いられます．従って，前述のクラス宣言は，関数の引数として値を受け取る場合や，多次元オブジェクトの要素にオブジェクトを代入する場合のみ用いられています．すなわち,

```
[]SATELLITE[]~/home/demo:[1]% a = 1
```

のように代入文によって，a のオブジェクトクラスが決定する場合はクラス宣言する必要はありません．しかし，クラス宣言されていない変数を関数および手続きの引数として指定できない場合があります．例えば，以下に示す ISPP モジュールの fftc 関数です.

```
[]SATELLITE[]~/home/demo:[2]% fftc("P", x, y, u, v)
```

ここで，"P" は計算方式フラグ，x，y は入力時系列，u，v は出力時系列です．この場合は fftc 関数を実行する前に u，v のクラス宣言を行う必要があります．

また，オブジェクトクラスの変換は演算により自動的に行われます．演算は演算子の左側にあるオブジェクト (左被演算オブジェクト) に，演算子と右側のオブジェクト (右被演算オブジェクト) をメッセージとして送ると解釈されるので，演算結果は左被演算オブジェクトのクラスになります．

これを利用するとオブジェクトクラスの変換を行うことができます．例えば，String と Scalar の演算においては Scalar が String に強制変換された後，文字列の処理が施されます．

```
[]SATELLITE[]~/home/demo:[3]% "test" + 3
test3
[]SATELLITE[]~/home/demo:[4]% "" + 3.1415926
3.14159
```

ここで，" " +3.1415926 の結果が 3.14159 になるのは数値の表示精度によるものです．また，Scalar と String の演算においては String が Scalar に変換された後，数値演算が施されます．例えば，

```
[]SATELLITE[]~/home/demo:[5]% 6 + "3.1415926"
9.1415926
[]SATELLITE[]~/home/demo:[6]% 0 + "1.08e-2"
0.0108
```

です．この操作は，その他のオブジェクトでもよく，Series から String への変換，または Series から String への変換も行うことができます．さらに，複数の要素を持つオブジェクト同士などでも同様に行うことができます．

2.4.4. オブジェクトの情報

オブジェクトクラスがわからない変数のオブジェクトクラスを得るには，typeof 関数を使用します．今，変数 x が series オブジェクト，変数 y が Snapshot オブジェクトとします．これらの変数のオブジェクトクラスを得る場合には，

```
[]SATELLITE[]~/home/demo:[1]% typeof(x)
series
[]SATELLITE[]~/home/demo:[2]% typeof(y)
snapshot
```

このように，オブジェクトクラスを得ることができます．

また，オブジェクトのインデックスを得るには，index 関数を使用します．例えば，

```
[]SATELLITE[]~/home/demo:[3]% a = 1~10
```

とした場合，

```
[]SATELLITE[]~/home/demo:[4]% index(a)
10
```

このように，オブジェクトのインデックスを得ることができます．また，データが b[10][50] のように多次元データの場合は，

```
[]SATELLITE[]~/home/demo:[5]% index(b)
[0]:%   10   50
```

のように出力されます．

2.5. 式と演算子

単純な算術演算だけでなく代入文，関数なども式として評価します．式が評価されると自動的にその値が表示されます．ただし，代入文だけは例外的に値を表示しません．本言語処理系においては，演算子と組み込み

関数は表記法が異なるだけで内部では同様に扱われます．式中に現れる演算子はその直前にあるオブジェクトに, 組み込み関数は第一引数のオブジェクトにメッセージとして送られます．従って，同じ演算子，組込み関数でもオブジェクトクラスの違いにより，その処理は異なるものとなります．演算子には，算術演算子，関係演算子，インクリメント・ディクリメント演算子，代入演算子に加えて，公差 1 の数列を生成する "~" 演算子，時系列の結合を行う "()" 演算子があります．数列生成演算子 "~" と時系列結合演算子 "()" の使用例を挙げます．

```
[]SATELLITE[]~/home/demo:[1]% x = -3~-1
[]SATELLITE[]~/home/demo:[2]% x
[0]:%   -3   -2   -1
[]SATELLITE[]~/home/demo:[3]% y = 1~3
[]SATELLITE[]~/home/demo:[4]% y
[0]:%    1    2    3
[]SATELLITE[]~/home/demo:[5]% z = (x, 0, y)
[]SATELLITE[]~/home/demo:[6]% z
[0]:%   -3   -2   -1    0    1
[5]:%    2    3
```

演算子には優先順位があり，表 2.2.「演算子の優先度 (高い順)」 に従って解釈されます．

表 2.2. 演算子の優先度 (高い順)

演算子	優先順位
() [] : []	
^	右結合
! - ++ --	左結合
~	左結合
* / %	左結合
+ -	左結合
> >= < <= == !=	左結合
&&	左結合
||	左結合
= += -= *= /= ^=	右結合

Series に対する比較演算子の使用例として，(z > 0) * z を示します．

```
[]SATELLITE[]~/home/demo:[7]% (z > 0) * z
[0]:%    0    0    0    0    1
[5]:%    2    3
```

オブジェクトは演算が終了すると消えてしまいます．演算結果を残すためには変数に代入しておかなければなりません．ここで扱う変数は，束縛されたデータ型を持たない単なるオブジェクトの入れものとして考えます．従って，同じ名前の変数でも代入されるごとにオブジェクトが変化する可能性があり，代入される以前のオブジェクトはゴミとなります．ゴミとなったオブジェクトはある期間保管されますが，ある一定時間に達すると回収された後，破壊されます．

2.6. 組み込み定数

インタラクティブに処理するときに，ユーザからの入力を軽減するため，または組み込み関数を容易に操作できるように組み込み定数を用意しています．その種類には

(1) 円周率を表す PI や自然対数の底 E などの組み込み定数 (表 2.3.「定数一覧」)

(2) const 文によるユーザ定義の定数，例えば，

```
[]SATELLITE[]~/home/demo:[1]% const Degree = PI/180
```

があります．定数には，オブジェクトを代入することはできません．また，本言語処理系では組み込み定数とユーザ定義の定数は同等に扱われ，組み込みの定数を const 文で変更することも可能です．const 文は右辺が計算式，また Series などのオブジェクトでもよく，その時点で評価された結果が設定されます．組み込み定数，ユーザ定義の定数が変数と異なるのは代入が禁止されている点だけです．

表 2.3. 定数一覧

定数名	値	意味
DEG	57.2957	$\frac{180}{\pi}$
E	2.7182	e，自然対数の底
GAMMA	0.5772	γ，Euler-Mascheroni の定数
PHI	1.6180	$\frac{\sqrt{5}+1}{2}$，黄金分割比
PI	3.1415	π，円周率

2.7. 制御構造

制御構造は，C 言語と同様な if 文や while 文，do-while 文，for 文，{ … } を使用した文のグループ化が使用できます．

- if (expr) stmt
- if (expr) stmt else stmt
- while (expr) stmt
- do stmt while (expr)
- for (expr ; expr ; expr) stmt

ここで，expr は，代入式，関係も含めた一般的な式です．また，stmt は単文であり，{ , } で囲みグループ化した文も stmt とみなします．expr において，論理演算子には AND 演算子 "&&" や OR 演算子 "||"，及び全ての関係演算子が使用できます．論理演算の結果が 0 ならば偽，それ以外は真と扱われます．また，論理演算の結果が複数の要素を持つ場合 (Series 同士の比較など) は，すべての要素が 0 でないとき真になります．

2.7.1. if 文

選択文は制御の幾つかの流れの一つを選ぶためのものです．以下に示すように二つの if 文のどちらも，条件式の結果が偽でなければ，最初のサブ文が実行されます．第 2 の形 (else 付き) の場合は，式が偽であれば，第 2 のサブ文に実行が移ります．

- if 文第 1 形式：

```
if ( 条件式 ) {
  サブ文;
}
```

- if 文第 2 形式：

```
if ( 条件式 ) {
  第 1 サブ文;
} else {
  第 2 サブ文;
}
```

例えば，「x が n より小さい場合，s に x を加算する」 という処理は，

```
[]SATELLITE[]~/home/demo:[1]% if (x < n){
+ s = s + x
+ }
```

と記述します．また，「x が n より小さい場合，s に x を加算し，それ以外なら s から x を減算する」という処理は，

```
[]SATELLITE[]~/home/demo:[2]% if (x < n){
+ s = s + x
+ } else {
+ s = s - x
+ }
```

と記述します．

2.7.2. while 文，do-while 文

繰り返し文はループを指定するのに使用されます．while 文と do-while 文では，条件式の値が偽でない限り，サブ文が繰り返し実行されます．while 文では，その式のすべての副作用も含めて，条件式の評価 (テスト) は文の各実行の前に行われます．これに対し do 文では，テストが行われるのは各繰り返しの後で行われます．

- while 文：

```
while ( 条件式 ) {
  サブ文;
}
```

- do-while 文：

```
do {
  サブ文;
} while ( 条件式 ) ;
```

例えば，while 文において，「x が n より小さい間，s に x を加算する」 という処理は，

```
[]SATELLITE[]~/home/demo:[1]% while (x < n) {
+ s = s + x
+ n++
+ }
```

と記述します．また，do-while 文においては，

```
[]SATELLITE[]~/home/demo:[2]% do {
+ s = s + x
+ n++
+ } while( x < n )
```

と記述します．

2.7.3. for 文

for 文では，最初の式が一度だけ評価されます．従って，これはループの初期化の指定にあたります．第 2 の式は各繰り返しの前に評価され，結果が偽になると for は終了します．一方，第 3 の式は繰り返しの後に評価され，ループの再初期化を指定します．

- for 文：

```
for ( 初期化 ; 条件式 ; 再初期化 ) {
  サブ文;
}
```

例えば,「s に x を n 回加算する」 という処理は

```
[]SATELLITE[]~/home/demo:[1]% for( i = 1; i <= n; i++) {
+ s = s + x
+ }
```

で記述されます.

これらの繰り返し文においては，break 文を使用すると，while や for 文の中から即座に抜けることができ，continue 文によりループの開始点に戻ることができます.

2.8. 関数と手続き

2.8.1. 変数・定数のスコープと関数・手続きの引数

SATELLITE における変数は，その関数・手続きの中でのみ有効なローカル変数として扱われます．関数・手続きの定義部以外で宣言された変数の有効範囲は関数・手続きの外のみであり，一種のローカル変数とみなせます.

関数・手続き内において，その外で宣言された変数を参照するには予約語 external により外部参照を明示する必要があります．ただし，ある関数・手続き内で宣言されている変数を別の関数・手続きにおいて参照することはできません.

組み込み定数および予約語 const で定義される定数は，その定義以後，いずれの関数・手続き内でも有効です．関数・手続き内で定数を定義することも可能ですが，関数・手続きの定義時には有効にならず，それらを実行した時点で有効になるため，ある関数内で定義した定数をその関数以外で参照する場合には，それ以前にその関数が実行されている必要があります.

SATELLITE では，関数・手続きの引数は，すべて変数渡しであるため，関数・手続き内で引数変数の値を操作した場合，呼び出し側にその操作した結果が反映されることになります.

2.8.2. 組み込み関数

数学ライブラリ，その他システム内の状態を制御する関数は，あらかじめ組み込み関数として定義されています．数学ライブラリは，series，snapshot，scalar のどれにも適用できます．また，各組み込み関数の優先順位は同じです．表 2.4.「数学ライブラリ一覧」 に数学ライブラリ，表 2.5.「システム関数一覧」 にシステム内部の状態を参照，変更するためのシステム関数を示します (詳しくはリファレンスマニュアル参照).

表 2.4. 数学ライブラリ一覧

関数名	内容
abs(x)	x の絶対値
acos(x)	x の逆余弦
asin(x)	x の逆正弦
atan(x)	x の逆正接
atan2(x,y)	x／y の逆正接，atan(x／y) に同じ
cos(x)	x の余弦
exp(x)	e^x
int(x)	x の整数部，0 に近い向きに切り捨て
mod(x)	x／y のあまり，x%y に同じ
log(x)	e を底とする x の対数 (自然対数)
log2(x)	2 を底とする x の対数
log10(x)	10 を底とする x の対数 (常用対数)
pow(x,y)	x^y，x^y に同じ
sgn(x)	x の符号
sin(x)	x の正弦
sqrt(x)	$\sqrt{x}$
tan(x)	x の正接

表 2.5. システム関数一覧

関数名	内容
history()	ヒストリを表示
index(x)	変数のインデックス (大きさ) を取得
inline(x)	プログラムのインライン展開
length(x)	データポイント／要素数の取得
printf(x,...)	変数の表示
strlen(x)	文字列の長さの取得
typeof(x)	変数型の取得
undef(x)	変数の登録抹消
symbols()	変数名の表示

2.8.3. ユーザ定義関数

ユーザは関数および手続きを定義することができます．例えば，「引数 n に 10 を加える」 という処理をする関数 plusten は，

```
[]SATELLITE[]~/home/demo:[1]% func plusten (n) {
+ f = n + 10
+ return f
+ }
```

で定義することができます．この関数は例えば，以下に示すように呼び出すことで目的とする処理が行えます．

```
[]SATELLITE[]~/home/demo:[2]% num = 8
[]SATELLITE[]~/home/demo:[3]% plusten (num)
18
```

また，関数は再帰呼び出しが可能です．以下に 「引数 x の階乗を求める」 という処理をする関数 fac を示します．

```
[]SATELLITE[]~/home/demo:[4]% func fac(x) {
+ if(x<=0) return 1 else return x*fac(x-1)
+ }
```

さらに，手続きを用いて，「引数 x に引数 n の値，引数 y に引数 n+1 の値を代入する」 という処理をする手続きを以下に示します．まず，項 2.4.3. 「オブジェクトクラスの変換」 で述べたように，クラス宣言されていない変数を関数および手続きの引数として指定できないため，手続きを呼び出す前に x，y をクラス宣言します．

```
[]SATELLITE[]~/home/demo:[5]% scalar x,y
[]SATELLITE[]~/home/demo:[6]% proc plusten(n, x, y) {
+ x = n
+ y = n + 1
+ }
```

例えば，この手続きは以下に示すように呼びだすことで目的とする処理が行えます．

```
[]SATELLITE[]~/home/demo:[7]% plusten(14, x, y)
[]SATELLITE[]~/home/demo:[8]% x
14
[]SATELLITE[]~/home/demo:[9]% y
15
```

また，関数，手続きの中で使用される変数は，外部参照を明示しない場合は，すべてローカル変数 (すなわち，その関数・手続き中でのみ有効) です．グローバル変数の宣言 (外部参照の明示) は予約語 external を用いて行います．グローバル変数を使用したい場合は使用したい関数の中で，その都度宣言する必要があります．前述の例と同じような処理を行う関数を予約語 external を用いて示します．

```
[]SATELLITE[]~/home/demo:[10]% proc subplusone(n) {
+ external x,y
+ x = n
+ y = n + 1
+ }
[]SATELLITE[]~/home/demo:[11]% func glplusone (gn) {
+ external x,y
+ subplusone(gn)
+ z = x + y
+ return z
+ }
```

このように使用する関数を定義します．定義した関数を以下のように呼び出します．

```
[]SATELLITE[]~/home/demo:[12]% scalar x,y,z
[]SATELLITE[]~/home/demo:[13]% glplusone (4)
9
[]SATELLITE[]~/home/demo:[14]% x
4
[]SATELLITE[]~/home/demo:[15]% y
5
[]SATELLITE[]~/home/demo:[16]% z
0
```

関数において引数の型をチェックしないため，多重定義された演算子，数学関数だけからなる関数は，実引数のオブジェクトクラスに関係なく正常な動作が得られます (多態関数)．ただし，関数内の演算子がサポート

されていないオブジェクトクラスは除きます．以下にシグモイド関数を用いた例を示します．まず，シグモイド関数を以下に示すように定義します．

```
[]SATELLITE[]~/home/demo:[17]% func sigmoid(t){
+ r=1/(1+exp(-t))
+ return r
+ }
```

引数 t を Scalar オブジェクトとして，関数を呼び出した場合は，

```
[]SATELLITE[]~/home/demo:[18]% sigmoid(0)
0.5
```

また，引数 t を Series オブジェクトとして，関数を呼び出した場合は，

```
[]SATELLITE[]~/home/demo:[19]% sigmoid(-10~10)
[ 0]:%  0.0000  0.0001  0.0003  0.0009  0.0025
[ 5]:%  0.0067  0.0180  0.0474  0.1192  0.2689
[10]:%  0.5000  0.7311  0.8808  0.9526  0.9820
[15]:%  0.9933  0.9975  0.9991  0.9997  0.9999
[20]:%  1.0000
```

また，引数 t を String オブジェクトとして，関数を呼び出した場合は，対応していない演算子の処理を実行できずにエラーとなります．

```
[]SATELLITE[]~/home/demo:[20]% sigmoid("test")
sl:   string not supported method
```

このように，Series オブジェクトを使用する例では，"~" 演算子により -10 から 10 までの時系列 (21 データポイント) を sigmoid 関数に渡すと，21 個の要素をもつ Series が結果として得られます．従って，時系列を扱うプログラムが数式に従って容易に記述できることがわかります．また，この関数例では，実引数として Scalar, Snapshot, Series, File オブジェクトクラスのいずれかを与えた場合には期待した処理結果が得られます．

2.8.4. 入出力

プログラム中でオブジェクトの表示や入力などは，組み込み関数や外部関数の他に組み込みコマンドでもいくつか提供されています．

printf 文は，表示するときの精度などが指定できます．使用方法は，C 言語の printf 関数に類似していますが，時系列の要素全てが自動的に表示される点が異なります．

```
[]SATELLITE[]~/home/demo:[1]% x = 3
[]SATELLITE[]~/home/demo:[2]% printf("x = %d\n",x)
x=3
[]SATELLITE[]~/home/demo:[3]% printf("%1.7f\n",(1,2,3,4,5))
[0]:%  1.000000  2.000000  3.000000
[3]:%  4.000000  5.000000
```

また，オブジェクトを端末からの入力で読み込むときには，read 関数があります．read 関数は読み込むオブジェクトクラスを引数として受け取り，入力された文字列をそのオブジェクトクラスに強制変換します．読み込まれたオブジェクトは read の戻り値となります．時系列のように複数の要素からなるオブジェクトは，コンマを区切り記号として分解され読み込まれます．例えば，

```
[]SATELLITE[]~/home/demo:[4]% y = read(series)
```

と入力した後，

```
1, 2, 3, 4, 5, 6, 7, 8
```

と入力すると，以下に示すように y には入力した数値が格納されます．

```
[]SATELLITE[]~/home/demo:[5]% y
[0]:%  1  2  3  4  5
[3]:%  6  7  8
```

2.9. プログラミング

2.9.1. パラメータ補完機能

SATELLITE 言語の特徴であるインタラクティブにパラメータの説明を表示しながら入力を催促する機能です．パラメータの区切り記号は，"," で表されます．例えば，グラフを表示する graph 関数を扱う場合には，

```
[]SATELLITE[]~/home/demo:[1]% graph
[]SATELLITE[]~/home/demo:[2]% graph(x
·····  Y-AXIS DATA ( Object or "T","F","D" )
```

ここで，Y 軸に表示するオブジェクトを入力します．例えば，"volt" を入力すると，

```
[]SATELLITE[]~/home/demo:[3]% graph(volt
[]SATELLITE[]~/home/demo:[4]% graph(volt,"T"
·····  X-AXIS DATA ( Object or "T","F","D" )
```

と表示され，X 軸に表示するオブジェクトを入力します．このように，オブジェクト入力を表示されるパラメータ全てについて行います．もし，デフォルトのパラメータで構わない場合についてはリターン入力で次の表示に移ります．

SATELLITE コマンドは，全て関数として構文解析をされるのでパラメータ列を丸括弧で括らなければなりません．また，入力されたパラメータは，エディットバッファに格納されているので自由に編集ができます．

2.9.2. インライン機能

ファイルに記述されているプログラムをインタプリタに読み込ませるためには，inline 関数を使用します．inline 関数は，ファイル名を引数にとり，そのファイルを標準入力として展開します．従って，端末からの入力と同様にプログラムが次々に評価されます．ファイル中に他のファイルをインライン展開することも可能です．inline 関数は実行の段階で処理されます．また，関数定義中で使用されたときも同様です．また，ファイルから読み込んでいる途中でエラーが生じたときには，その後のプログラムはそれ以降評価されません．作成したファイル (ファイル名は testsum.sl とします) の例を以下に示します．

```
sum = 0;
for( i = 1; i <= 10; i++){
  sum = sum + i;
}
printf("sum = %d\n", sum);
```

このファイルをインタプリタに読み込み実行させるには，

```
[]SATELLITE[]~/home/demo:[1]% inline("testsum.sl")
```

と入力することで，

```
[]SATELLITE[]~/home/demo:[2]% inline("testsum.sl")
sum = 55
```

目的とする処理が行われます．また，ファイルの中で，別ファイルを読み込むときは，以下のように inline 関数を用いてプログラミングします．

```
sum = 0;
for( i = 1; i <= 10; i++) {
  sum = sum + i;
```

```
}
printf("sum = %d\n",sum);
inline("別のファイル名");
```

第 3 章 システム関連モジュール

3.1. SYSTEM モジュールの概要

SYSTEM モジュールは，データの切り出しをはじめとするデータの基本的な編集機能を集めたシステム・モジュールです．データの最大・最小値の検索や，データの形式の変更，データファイルのヘッダ情報を表示するなどの関数も収められています．

3.2. コマンドの使用例

ここでは，主なコマンドの使い方を説明します．

3.2.1. コマンドマニュアルを表示する (help)

コマンドのマニュアルをオンライン表示します．この関数の形式を以下に示します．

```
help( "funcname" )
```

ここで，funcname は関数名です．このとき，引数は String 型でなければなりません． 例えば，

```
[]SATELLITE[]~/home/demo:[1]% help("help")
```

とすると，help 関数のマニュアルが表示されます．

3.2.2. データファイルのヘッダ情報の表示・変更をする (header)

データファイルのヘッダ情報 (データ形式，インデックスなど) を表示，もしくは変更します．この関数の形式を以下に示します．

```
header( "file_name", mode )
```

ここで，file_name はデータファイル名，mode はモード (0 : 画面表示，1 : 変更) です．例えば，

```
[]SATELLITE[]~/home/demo:[1]% header("test.dat", 0)
```

とすると，データファイル (test.dat) の情報を画面表示します．

3.2.3. バッチ処理の一時停止をする (wait)

リターンキーが押されるまで待つ関数です．この関数の形式を以下に示します．

```
wait()
```

また，引数に Scalar 型の数字を伴うことで，指定秒間処理を停止することもできます．

```
[]SATELLITE[]~/home/demo:[1]% wait(3)
```

3.2.4. データサイズ，インデックスの変更をする (reform)

オブジェクトのフォーマットを変更する関数です．この関数の形式を以下に示します．

```
y = reform(x, index)
```

x は入力オブジェクト，y は出力オブジェクト，index はデータサイズです．例えば，図 3.1.「reform 関数の使用例 1」 に示すように 1 次元 Series オブジェクト a を 2 次元 Series オブジェクト b に変更する場合は，

```
[]SATELLITE[]~/home/demo:[1]% a = 1~14
[]SATELLITE[]~/home/demo:[2]% a
```

```
[ 0]:%   1   2   3   4   5
[ 5]:%   6   7   8   9  10
[10]:%  11  12  13  14
[]SATELLITE[]~/home/demo:[3]% b = reform(a, (7, 2))
[]SATELLITE[]~/home/demo:[4]% b
[0]:[0]%   1   2
[1]:[0]%   3   4
[2]:[0]%   5   6
[3]:[0]%   7   8
[4]:[0]%   9  10
[5]:[0]%  11  12
[6]:[0]%  13  14
```

とします.

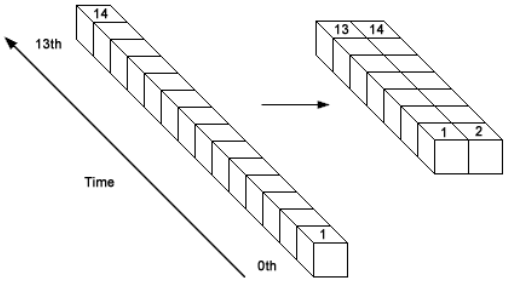


図 3.1. reform 関数の使用例 1

また，図 3.2.「reform 関数の使用例 2」 に示すように 2 次元オブジェクト b を 3 次元オブジェクト c に変更する場合は,

```
[]SATELLITE[]~/home/demo:[5]% c = reform(b, (3,2,4))
[]SATELLITE[]~/home/demo:[6]% c
[0]:[0][0]%   1   2   3   4
[0]:[1][0]%   5   6   7   8
[1]:[0][0]%   9  10  11  12
[1]:[1][0]%  13  14   0   0
[2]:[0][0]%   0   0   0   0
[2]:[1][0]%   0   0   0   0
```

とします．このとき，指定したサイズが入力オブジェクトより大きい場合には，データ後部に 0 詰めをします.

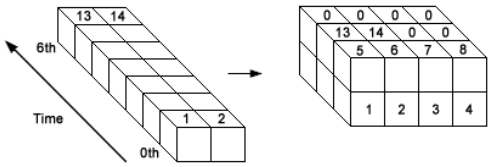


図 3.2. reform 関数の使用例 2

この関数は Snapshot オブジェクトについても同様に扱うことができます.

3.2.5. データバッファのモニター (bm)

ユーザから起動されるプロセスと平行して，リアルタイムにデータバッファをグラフに表示するモニタを起動させる関数です．この関数の形式を以下に示します.

```
bm( x )
```

ここで，x はモニタリング対象のオブジェクトです．バッファ x をモニタしたい場合は，

```
[]SATELLITE[]~/home/demo:[3]% bm(x)
```

により，バッファモニタを起動させることができます．図 3.3.「バッファモニタの起動画面 (全景)」 はバッファモニタの起動例です．ここで，x には以下に示す処理によりデータが事前に格納されています．

```
[]SATELLITE[]~/home/demo:[1]% t = 0~99
[]SATELLITE[]~/home/demo:[2]% x = sin(2*PI*t/100)
```

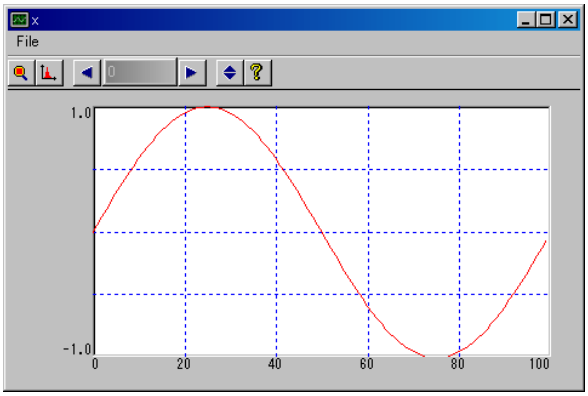


図 3.3. バッファモニタの起動画面 (全景)

また，「SCALE ボタン」  をクリックすることにより，新たにウィンドウが開かれ，スケールを自由に設定することが可能です．図 3.4.「スケール設定画面」 にスケール設定画面を示します．

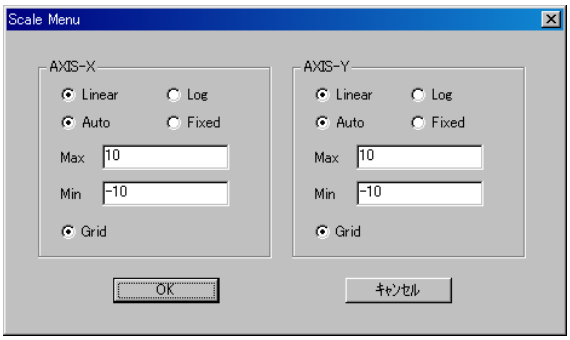


図 3.4. スケール設定画面

前例では，モニタリング対象のバッファ x は 1 次元 Series オブジェクトでした．ここでは，2 次元 Series オブジェクトのバッファモニタについて説明します．まず，reform 関数を用いて，以下に示すように，1 次元 Series オブジェクト x を 2 次元 Series オブジェクト y にします．

```
[]SATELLITE[]~/home/demo:[4]% y = reform(x, (2,50))
```

この 2 次元 Series オブジェクト y をモニタリングする場合，前例と同様に，

```
[]SATELLITE[]~/home/demo:[5]% bm(y)
```

とします．この場合のバッファモニタ画面を 図 3.5.「バッファモニタの画面 1 (2 次元 Series オブジェクト)」 に示します．

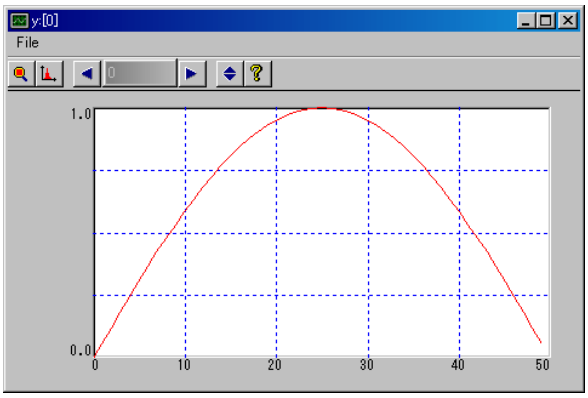


図 3.5. バッファモニタの画面 1 (2 次元 Series オブジェクト)

ここで,「インデックスボタン」 をクリックすることにより，図 3.6. 「バッファモニタの画面 2 (2 次元 Series オブジェクト)」 に示すようなモニタ画面に変わります.

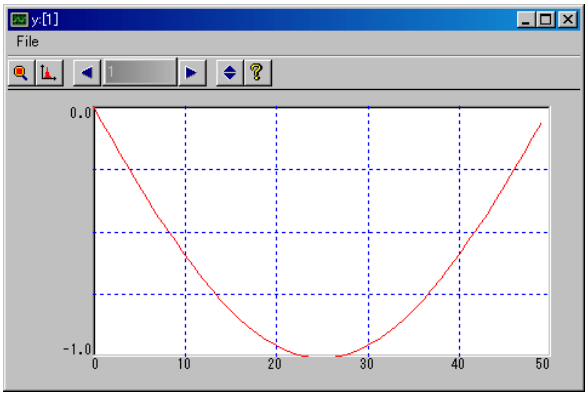


図 3.6. バッファモニタの画面 2 (2 次元 Series オブジェクト)

図 3.5. 「バッファモニタの画面 1 (2 次元 Series オブジェクト)」 と 図 3.6. 「バッファモニタの画面 2 (2 次元 Series オブジェクト)」 は同じ変数名の時間的に異なるデータが表示されています．この例では，図 3.5. 「バッファモニタの画面 1 (2 次元 Series オブジェクト)」 の横軸は 図 3.3. 「バッファモニタの起動画面 (全景)」 に示す空間 1 から 50 までのデータを表し，図 3.6. 「バッファモニタの画面 2 (2 次元 Series オブジェクト)」 は 図 3.3. 「バッファモニタの起動画面 (全景)」 に示す空間 51 から 100 までのデータを表します.

3.2.6. サンプリング周波数を変更する (sampling)

サンプリング周波数を変更する関数です．この関数の形式を以下に示します.

```
sampling( frequency )
```

ここで，frequency はサンプリング周波数です．ここで設定するサンプリング周波数は ISPP モジュールの関数や，GPM の graph 関数から参照されます．例えば,

```
[]SATELLITE[]~/home/demo:[1]% a = 1~10
```

としたときのグラフを 図 3.7. 「sampling 関数の使用例 1」 に示します．この場合，frequency のデフォルトは 1000.0 となっています.

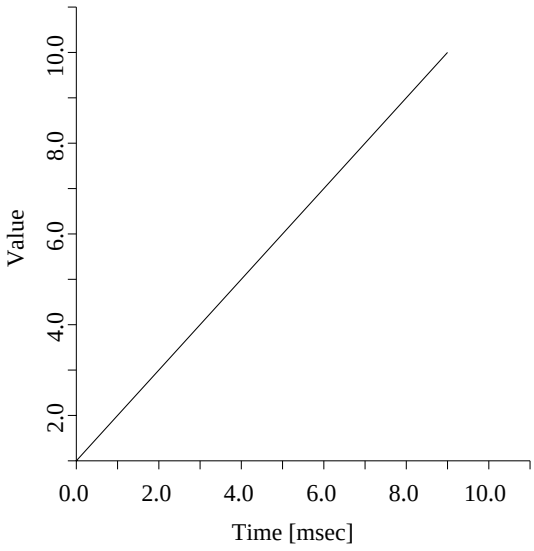


図 3.7. sampling 関数の使用例 1

また，サンプリング周波数を 100 とした後の a のグラフを 図 3.8.「sampling 関数の使用例 2」 に示します.

```
[]SATELLITE[]~/home/demo:[2]% sampling(100)
```

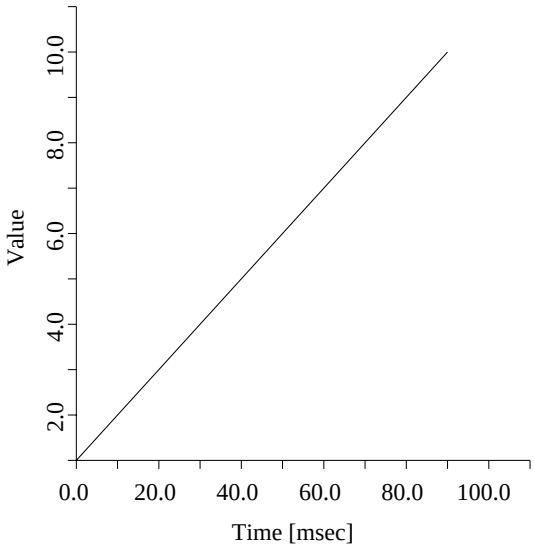


図 3.8. sampling 関数の使用例 2

3.2.7. データの一部を切り出す (cut)

オブジェクトの一部を取り出す関数です．この関数の形式を以下に示します.

```
y = cut(x, start, end)
```

ここで，x は入力オブジェクト，y は出力オブジェクト，start は切り出すデータの始点座標，end は切り出すデータの終点座標です．例えば，図 3.9.「cut 関数の使用例 1」 に示すように 1 次元 Series オブジェクト a の一部を切り出すには,

```
[]SATELLITE[]~/home/demo:[1]% a = 1~7
[]SATELLITE[]~/home/demo:[2]% a
```

```
[0]:%   1   2   3   4   5
[5]:%   6   7
[]SATELLITE[]~/home/demo:[3]% b = cut(a, 3, 5)
[]SATELLITE[]~/home/demo:[4]% b
[0]:%   4   5   6
```

とします．ここで，

```
[]SATELLITE[]~/home/demo:[5]% b = cut(a, 3, 5)
```

を

```
[]SATELLITE[]~/home/demo:[6]% b = cut(a, 5, 3)
sl4: Error [<SYSTEM:cut> No.7] Illegal index near line 6
```

とした場合は，正しく実行されません．

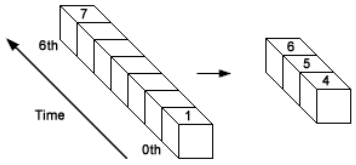


図 3.9. cut 関数の使用例 1

また，図 3.10.「cut 関数の使用例 2」 に示すように，2 次元 Series オブジェクト b の一部を切り出すには，

```
[]SATELLITE[]~/home/demo:[7]% a = 1~14
[]SATELLITE[]~/home/demo:[8]% b = reform(a, (7,2))
[]SATELLITE[]~/home/demo:[9]% c = cut(b,(3,0),(5,0))
[]SATELLITE[]~/home/demo:[10]% c
[0]:[0]%   7
[1]:[0]%   9
[2]:[0]%  11
```

とします．

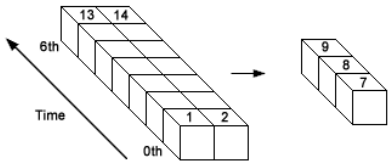


図 3.10. cut 関数の使用例 2

この関数は Snapshot オブジェクトについても同様に扱うことができます．

3.2.8. データの一部に別のデータを上書きする (put)

オブジェクトの一部に，別のデータを入れる関数です．この関数の形式を以下に示します．

```
z = put(x, y, index)
```

ここで，x は入力オブジェクト，y は挿入オブジェクト，z は出力オブジェクト，index は y を書き込む座標です．出力オブジェクト z は入力オブジェクト x の時間軸方向の長さに依存します．例えば，図 3.11.「put 関数の使用例 (1 次元 Series オブジェクト)」 に示すように 1 次元 Series オブジェクト a に 1 次元 Series オブジェクト b を上書きするには，

```
[]SATELLITE[]~/home/demo:[1]% a = 1~7
[]SATELLITE[]~/home/demo:[2]% b = 11~17
[]SATELLITE[]~/home/demo:[3]% c = put(a,b,3)
[]SATELLITE[]~/home/demo:[4]% c
[0]:%   1   2   3  11  12
[5]:%  13  14
```

とします.

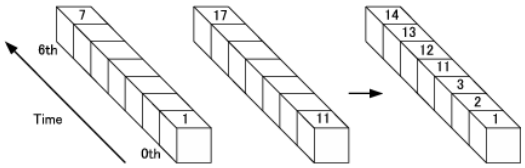


図 3.11. put 関数の使用例 (1 次元 Series オブジェクト)

また，図 3.12.「put 関数の使用例 (2 次元 Series オブジェクト)」 に示すように，2 次元 Series オブジェクト ar に br を上書きするには,

```
[]SATELLITE[]~/home/demo:[5]% a = 1~14
[]SATELLITE[]~/home/demo:[6]% b = 30~35
[]SATELLITE[]~/home/demo:[7]% ar = reform(a,(7,2))
[]SATELLITE[]~/home/demo:[8]% br = reform(b,(3,2))
[]SATELLITE[]~/home/demo:[9]% c = put(ar,br,(2,0))
[]SATELLITE[]~/home/demo:[10]% c
[0]:[0]%   1   2
[1]:[0]%   3   4
[2]:[0]%  30  31
[3]:[0]%  32  33
[4]:[0]%  34  35
[5]:[0]%  11  12
[6]:[0]%  13  14
```

とします.

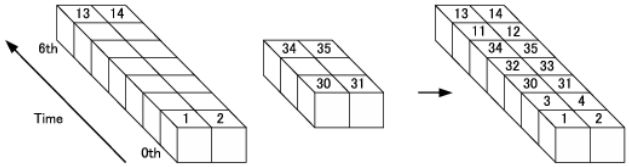


図 3.12. put 関数の使用例 (2 次元 Series オブジェクト)

この関数は Snapshot オブジェクトについても同様に扱うことができます.

3.2.9. データとデータを連結する (merge)

2 つのオブジェクトのデータを連結します．この関数の形式を以下に示します.

```
z = merge(x, y)
```

ここで，x は入力オブジェクト 1，y は入力オブジェクト 2，z は出力オブジェクトです．出力オブジェクト z には入力オブジェクト x の終端に入力オブジェクト y を連結したデータが出力されます．また，多次元データの場合には，サブインデックスが一致しなければなりません．例えば，図 3.13.「merge 関数の使用例 (2 次元 Series オブジェクト)」 に示すように 2 次元 Series オブジェクト ar と br を連結させるには,

```
[]SATELLITE[]~/home/demo:[1]% a = 1~8
[]SATELLITE[]~/home/demo:[2]% b = 30~35
[]SATELLITE[]~/home/demo:[3]% ar = reform(a,(4,2))
[]SATELLITE[]~/home/demo:[4]% br = reform(b,(3,2))
[]SATELLITE[]~/home/demo:[5]% c = merge(ar, br)
[]SATELLITE[]~/home/demo:[6]% c
[0]:[0]%   1   2
[1]:[0]%   3   4
[2]:[0]%   5   6
[3]:[0]%   7   8
[4]:[0]%  30  31
[5]:[0]%  32  33
[6]:[0]%  34  35
```

とします.

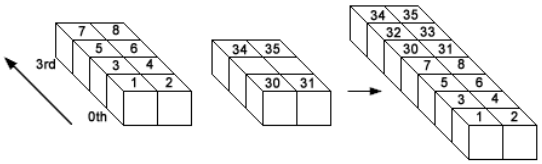


図 3.13. merge 関数の使用例 (2 次元 Series オブジェクト)

この関数は Snapshot オブジェクトについても同様に扱うことができます.

3.2.10. データの一部を指定した値で埋める (fill)

これはオブジェクトの指定した範囲を指定した値で埋める関数です．この関数の形式を以下に示します.

```
y = fill(x, start, end, value)
```

ここで，x は入力オブジェクト，y は出力オブジェクト，start は始点座標，end は終点座標，value は埋める数値です．すなわち，入力オブジェクト x の start から end の位置までの範囲を value の値で埋めます．また，データのサイズより大きい位置を指定した場合には，最大範囲までの値で埋めます．例えば，図 3.14.「fill 関数の使用例 1」 に示すように 1 次元 Series オブジェクト a の指定した範囲を 20 で埋めるには,

```
[]SATELLITE[]~/home/demo:[1]% a = 1~7
[]SATELLITE[]~/home/demo:[2]% b = fill(a,3,5,20)
[]SATELLITE[]~/home/demo:[3]% b
[0]: %   1   2   3  20  20
[5]: %  20   7
```

とします.

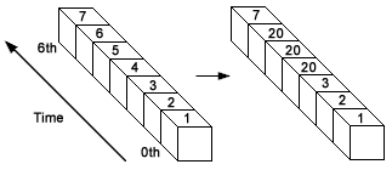


図 3.14. fill 関数の使用例 1

また，図 3.15.「fill 関数の使用例 2」 に示すように，2 次元 Series オブジェクト b の指定した範囲を 30 で埋めるには,

```
[]SATELLITE[]~/home/demo:[4]% a = 1~14
[]SATELLITE[]~/home/demo:[5]% b = reform(a,(7,2))
```

```
[]SATELLITE[]~/home/demo:[6]% c = fill(b,(3,0),(5,0),30)
[]SATELLITE[]~/home/demo:[7]% c
[0]:[0]%   1   2
[1]:[0]%   3   4
[2]:[0]%   5   6
[3]:[0]%  30   8
[4]:[0]%  30  10
[5]:[0]%  30  12
[6]:[0]%  13  14
```

とします.

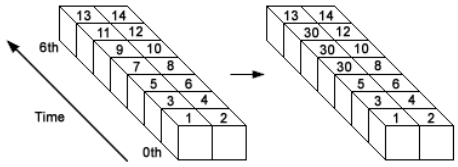


図 3.15. fill 関数の使用例 2

この関数は Snapshot オブジェクトについても同様に扱うことができます.

3.2.11. データの順番を逆にする (reverse)

入力データの配列を逆にしたデータを得る関数です．この関数の形式を以下に示します.

```
y = reverse( x )
```

ここで，x は入力オブジェクト，y は出力オブジェクトです．例えば，以下に示すように，1 次元 Seires オブジェクト a を逆にしたデータを得るには,

```
[]SATELLITE[]~/home/demo:[1]% a = 1~7
[]SATELLITE[]~/home/demo:[2]% b = reverse(a)
[]SATELLITE[]~/home/demo:[3]% a
[0]: %   1   2   3   4   5
[5]: %   6   7
[]SATELLITE[]~/home/demo:[4]% b
[0]: %   7   6   5   4   3
[5]: %   2   1
```

とします.

また，図 3.16.「reverse 関数の使用例 (2 次元 Series オブジェクト)」 に示すように 2 次元 Series オブジェクト a を逆にしたデータを得るには,

```
[]SATELLITE[]~/home/demo:[5]% a = 1~14
[]SATELLITE[]~/home/demo:[6]% b = reform(a,(7,2))
[]SATELLITE[]~/home/demo:[7]% b
[0]:[0]%   1   2
[1]:[0]%   3   4
[2]:[0]%   5   6
[3]:[0]%   7   8
[4]:[0]%   9  10
[5]:[0]%  11  12
[6]:[0]%  13  14
[]SATELLITE[]~/home/demo:[8]% c = reverse( b )
[]SATELLITE[]~/home/demo:[9]% c
[0]:[0]%  14  13
```

```
[1]:[0]%  12  11
[2]:[0]%  10   9
[3]:[0]%   8   7
[4]:[0]%   6   5
[5]:[0]%   4   3
[6]:[0]%   2   1
```

とします.

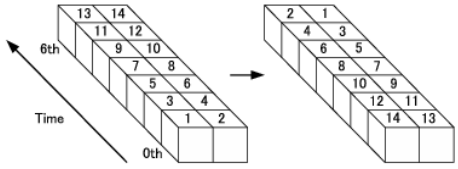


図 3.16. reverse 関数の使用例 (2 次元 Series オブジェクト)

この関数は Snapshot オブジェクトについても同様に扱うことができます.

3.2.12. 指定した位置をデータの先頭にする (rotate)

オブジェクトの内容を，指定した位置が先頭となるように移動する関数です．この関数の形式を以下に示します.

```
y = rotate(x, index)
```

ここで，x は入力オブジェクト，y は出力オブジェクト，index は先頭にする座標です．例えば，図 3.17.「rotate 関数の使用例 (1 次元 Series オブジェクト)」 に示すように，1 次元 Series オブジェクト a を指定した位置が先頭となるように移動するには,

```
[]SATELLITE[]~/home/demo:[1]% a = 1~7
[]SATELLITE[]~/home/demo:[2]% a
[0]: %   1   2   3   4   5
[5]: %   6   7
[]SATELLITE[]~/home/demo:[3]% b = rotate(a, 3)
[]SATELLITE[]~/home/demo:[4]% b
[0]: %   4   5   6   7   1
[5]: %   2   3
```

とします.

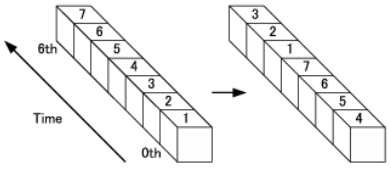


図 3.17. rotate 関数の使用例 (1 次元 Series オブジェクト)

また，図 3.18.「rotate 関数の使用例 (2 次元 Series オブジェクト)」 に示すように 2 次元 Series オブジェクト b を指定した位置が先頭となるように移動するには,

```
[]SATELLITE[]~/home/demo:[5]% a = 1~14
[]SATELLITE[]~/home/demo:[6]% b = reform(a,(7,2))
[]SATELLITE[]~/home/demo:[7]% b
[0]:[0]%   1   2
[1]:[0]%   3   4
```

```
[2]:[0]%   5   6
[3]:[0]%   7   8
[4]:[0]%   9  10
[5]:[0]%  11  12
[6]:[0]%  13  14
[]SATELLITE[]~/home/demo:[8]% c = rotate(b,(3,0))
[]SATELLITE[]~/home/demo:[9]% c
[0]:[0]%   7   8
[1]:[0]%   9  10
[2]:[0]%  11  12
[3]:[0]%  13  14
[4]:[0]%   1   2
[5]:[0]%   3   4
[6]:[0]%   5   6
```

とします.

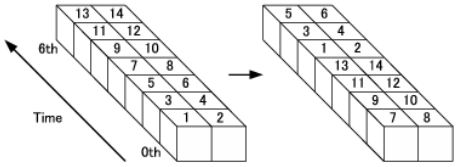


図 3.18. rotate 関数の使用例 (2 次元 Series オブジェクト)

この関数は Snapshot オブジェクトについても同様に扱うことができます.

3.2.13. データを間引く (thin)

データを指定したステップ数で間引く関数です．この関数の形式を以下に示します.

```
y = thin(x, step)
```

ここで，x は入力時系列，y は出力時系列，step は間引きの間隔 (step-1 点分のデータを間引く) です．従って，step=0 および 1 の場合はデータに変化はありません．例えば，図 3.19.「thin 関数の使用例 1」 に示すように，1 次元 Series オブジェクト a を間引く場合には,

```
[]SATELLITE[]~/home/demo:[1]% a = 1~7
[]SATELLITE[]~/home/demo:[2]% b = thin(a, 2)
[7]->[4]
[]SATELLITE[]~/home/demo:[3]% b
[0]: %   1   3   5   7
```

とします.

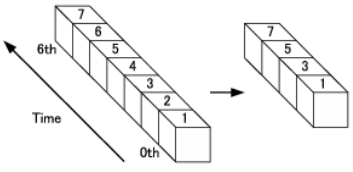


図 3.19. thin 関数の使用例 1

また，図 3.20.「thin 関数の使用例 2」 に示すように 2 次元 Series オブジェクト b を間引いたデータを得るには,

```
[]SATELLITE[]~/home/demo:[4]% a = 1~21
[]SATELLITE[]~/home/demo:[5]% b = reform(a,(7,3))
```

```
[]SATELLITE[]~/home/demo:[6]% b
[0]:[0]%   1   2   3
[1]:[0]%   4   5   6
[2]:[0]%   7   8   9
[3]:[0]%  10  11  12
[4]:[0]%  13  14  15
[5]:[0]%  16  17  18
[6]:[0]%  19  20  21
[]SATELLITE[]~/home/demo:[7]% c = thin(b,(3,2))
[7][3]->[3][2]
[]SATELLITE[]~/home/demo:[8]% c
[0]:[0]%   1   3
[1]:[0]%  10  12
[2]:[0]%  19  21
```

とします.

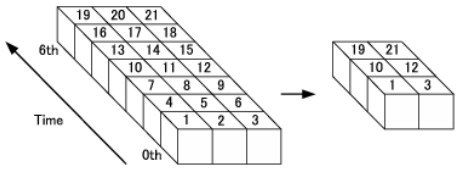


図 3.20. thin 関数の使用例 2

この関数は Snapshot オブジェクトについても同様に扱うことができます.

3.2.14. データの指定した位置の値を得る (get)

オブジェクトの指定した位置の値を得る関数です．この関数の形式を以下に示します.

```
y = get(x, position)
```

ここで，x は入力オブジェクト，y は指定した位置の値のデータ値，position はデータの座標です．例えば，図 3.21.「get 関数の使用例 1」 に示すように 1 次元 Series オブジェクト a の指定した位置の値を得るには,

```
[]SATELLITE[]~/home/demo:[1]% a = 1~7
[]SATELLITE[]~/home/demo:[2]% b = get(a, 3)
[]SATELLITE[]~/home/demo:[3]% b
4
```

とします.

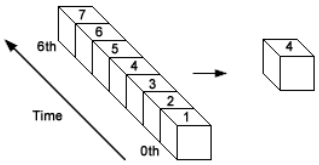


図 3.21. get 関数の使用例 1

また，図 3.22.「get 関数の使用例 2」 に示すように，2 次元 Seires オブジェクト b の指定した位置を得るには,

```
[]SATELLITE[]~/home/demo:[4]% a = 1~14
[]SATELLITE[]~/home/demo:[5]% b = reform(a,(7,2))
[]SATELLITE[]~/home/demo:[6]% c = get(b,(3,0))
```

```
[]SATELLITE[]~/home/demo:[7]% c
7
```

とします.

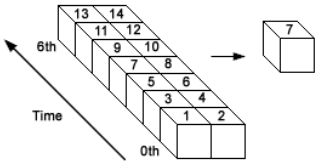


図 3.22. get 関数の使用例 2

この関数は Snapshot オブジェクトについても同様に扱うことができます.

3.2.15. データの最大値を得る (max)

データの最大値を得る関数です．この関数の形式を以下に示します.

```
y = max(x)
```

ここで，x は入力オブジェクト，y は最大値です．例えば，以下に示すように 1 次元 Series オブジェクト a の最大値を得るには,

```
[]SATELLITE[]~/home/demo:[1]% a = (3,7,5,1,6,2,4)
[]SATELLITE[]~/home/demo:[2]% c = max(a)
[]SATELLITE[]~/home/demo:[3]% c
7
```

とします．また，以下に示すように，2 次元 Series オブジェクト b の最大値を得るには,

```
[]SATELLITE[]~/home/demo:[4]% a = (7,13,1,3,12,6,11,4,14,2,9,10,5,8)
[]SATELLITE[]~/home/demo:[5]% b = reform(a,(7,2))
[]SATELLITE[]~/home/demo:[6]% c = max(b)
[]SATELLITE[]~/home/demo:[7]% c
14
```

この関数は Snapshot オブジェクトについても同様に扱うことができます.

3.2.16. データの最小値を得る (min)

データの最小値を得る関数です．この関数の形式を以下に示します.

```
y = min(x)
```

ここで，x は入力オブジェクト，y は最小値です．例えば，以下に示すように 1 次元 Series オブジェクト a の最小値を得るには,

```
[]SATELLITE[]~/home/demo:[1]% a = (3,7,5,1,6,2,4)
[]SATELLITE[]~/home/demo:[2]% c = min(a)
[]SATELLITE[]~/home/demo:[3]% c
7
```

とします．また，以下に示すように，2 次元 Series オブジェクト b の最小値を得るには,

```
[]SATELLITE[]~/home/demo:[4]% a = (7,13,1,3,12,6,11,4,14,2,9,10,5,8)
[]SATELLITE[]~/home/demo:[5]% b = reform(a,(7,2))
[]SATELLITE[]~/home/demo:[6]% c = min(b)
```

```
[]SATELLITE[]~/home/demo:[7]% c
1
```

この関数は Snapshot オブジェクトについても同様に扱うことができます.

3.2.17. データの最大値のデータ位置を得る (maxpos)

データの最大値のデータ位置を得る関数です．この関数の形式を以下に示します.

```
y = maxpos(x, num)
```

ここで，x は入力オブジェクト，y は位置格納オブジェクト，num は取得するデータ位置の数です．例えば，図 3.23.「maxpos 関数の使用例 1」 に示すように 1 次元 Series オブジェクト a の最大値のデータ位置を得るには,

```
[]SATELLITE[]~/home/demo:[1]% a = (3,7,5,1,6,2,4)
[]SATELLITE[]~/home/demo:[2]% c = maxpos(a,1)
[]SATELLITE[]~/home/demo:[3]% c
1
```

とします.

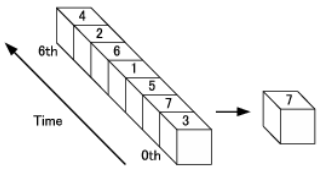


図 3.23. maxpos 関数の使用例 1

また，図 3.24.「maxpos 関数の使用例 2」 に示すようにオブジェクト a の最大値のデータ位置とその次に大きい値を持つデータの位置を得るには,

```
[]SATELLITE[]~/home/demo:[4]% c = maxpos(a, 2)
[]SATELLITE[]~/home/demo:[5]% c
[0]:[0]%1
[1]:[0]%4
```

とします.

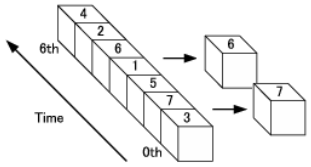


図 3.24. maxpos 関数の使用例 2

また，図 3.25.「maxpos 関数の使用例 3」 に示すように，2 次元 Series オブジェクト b の最大値のデータの位置を得るには,

```
[]SATELLITE[]~/home/demo:[6]% a = (7,13,1,3,12,6,11,4,14,2,9,10,5,8)
[]SATELLITE[]~/home/demo:[7]% b = reform(a,(7,2))
[]SATELLITE[]~/home/demo:[8]% c = maxpos(b,1)
[]SATELLITE[]~/home/demo:[9]% c
[0]:[0]%   4   0
```

とします.

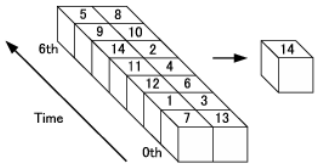


図 3.25. maxpos 関数の使用例 3

この関数は Snapshot オブジェクトについても同様に扱うことができます.

3.2.18. データの最小値のデータ位置を得る (minpos)

データの最大値のデータ位置を得る関数です．この関数の形式を以下に示します.

```
y = minpos(x, num)
```

ここで，x は入力オブジェクト，y は位置格納オブジェクト，num は取得するデータ位置の数です．例えば，図 3.26.「minpos 関数の使用例 1」 に示すように 1 次元 Series オブジェクト a の最小値のデータ位置を得るには,

```
[]SATELLITE[]~/home/demo:[1]% a = (3,7,5,1,6,2,4)
[]SATELLITE[]~/home/demo:[2]% c = minpos(a,1)
[]SATELLITE[]~/home/demo:[3]% c
3
```

とします.

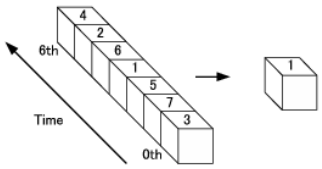


図 3.26. minpos 関数の使用例 1

また，図 3.27.「minpos 関数の使用例 2」 に示すようにオブジェクト a の最小値のデータ位置とその次に大きい値を持つデータの位置を得るには,

```
[]SATELLITE[]~/home/demo:[4]% b = minpos(a, 2)
[]SATELLITE[]~/home/demo:[5]% b
[0]:[0]%   3
[1]:[0]%   5
```

とします.

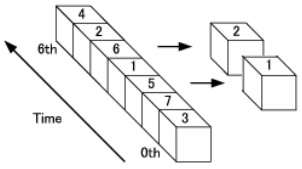


図 3.27. minpos 関数の使用例 2

また，図 3.28.「minpos 関数の使用例 3」 に示すように，2 次元 Series オブジェクト b の最小値のデータの位置を得るには,

```
[]SATELLITE[]~/home/demo:[6]% a = (7,13,1,3,12,6,11,4,14,2,9,10,5,8)
[]SATELLITE[]~/home/demo:[7]% b = reform(a,(7,2))
```

```
[]SATELLITE[]~/home/demo:[8]% c = minpos(b,1)
[]SATELLITE[]~/home/demo:[9]% c
[0]:[0]%   1   0
```

とします.

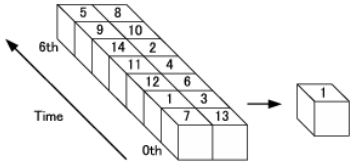


図 3.28. minpos 関数の使用例 3

この関数は Snapshot オブジェクトについても同様に扱うことができます.

3.2.19. 指定した値に近い値をデータから見つける (find)

入力オブジェクトから指定した値に最も近いデータを見つけ，値とその位置を表示し，位置を Series オブジェクトとして返す関数です．この関数の形式を以下に示します.

```
ip = find(x, val, num)
```

ここで，x は入力オブジェクト，ip はデータの位置，val は捜し出す値，num は捜し出すデータの個数です. 例えば，図 3.29.「find 関数の使用例 1 (1 次元 Series オブジェクト)」 に示すように 1 次元 Series オブジェクト a の中で，指定した値に最も近いデータとその次に近いデータの値と位置を得るには,

```
[]SATELLITE[]~/home/demo:[1]% a = 1~7
[]SATELLITE[]~/home/demo:[2]% c = find(a, 5.8, 2)
DATA[6]  --  POINT:[5]
DATA[5]  --  POINT:[4]
[]SATELLITE[]~/home/demo:[3]% c
[0]:[0]%   5
[1]:[0]%   4
```

とします.

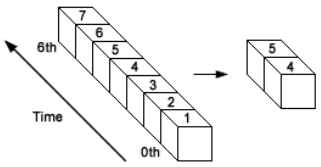


図 3.29. find 関数の使用例 1 (1 次元 Series オブジェクト)

また，図 3.30.「find 関数使用例 2 (2 次元 Series オブジェクト)」 に示すように 2 次元オブジェクト a の中で，指定した値に最も近いデータとその次に近いデータの値と位置を得るには,

```
[]SATELLITE[]~/home/demo:[4]% a = 1~14
[]SATELLITE[]~/home/demo:[5]% b = reform(a,(7,2))
[]SATELLITE[]~/home/demo:[6]% c = find(b, 6.8, 2)
DATA[7]  --  POINT:[3][0]
DATA[6]  --  POINT:[2][1]
[]SATELLITE[]~/home/demo:[7]% c
[0]:[0]%   3   0
[1]:[0]%   2   1
```

とします.

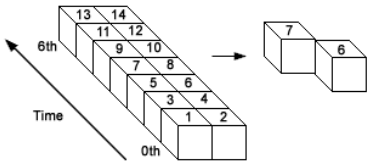


図 3.30. find 関数使用例 2 (2 次元 Series オブジェクト)

この関数は Snapshot オブジェクトについても同様に扱うことができます.

3.2.20. 現在のディレクトリを取得する (pwd)

現在作業を行っているディレクトリ名を取得する関数です．この関数の形式を以下に示します.

```
pwd()
```

3.2.21. ディレクトリを作成する (mkdir)

新たにディレクトリを作成する関数です．この関数の形式を以下に示します.

```
mkdir("file_name")
```

ここで，file_name は新たに作成するディレクトリ名です．ファイル名は，" " で囲む必要があります．例えば,

```
[]SATELLITE[]~/home/demo:[1]% mkdir("My Directly")
```

とすると，現在作業を行っているディレクトリ（~/home/demo）に My Directly というディレクトリが作成されます.

3.2.22. 一時ディレクトリの場所を取得する (tempdir)

システムで利用されている一時ディレクトリの場所を取得する関数です．この関数の形式を以下に示します.

```
tempdir()
```

3.2.23. 指定した文字列の環境変数の値を得る (getenv)

指定した文字列の環境変数の値を得る関数です．この関数の形式を以下に示します.

```
getenv("env_name")
```

ここで，env_name は環境変数名です．コマンド名は，" " で囲む必要があります.

第 4 章 ディジタル信号処理パッケージ

4.1. ISPP の概要

ISPP (Interactive Signal Processing Package) は，SATELLITE の中核をなすモジュールで，ウィンドウ等の前処理，FFT，線形予測によるスペクトル推定，フィルタリング，ケプストラム解析等，ディジタル信号処理の方法論を網羅する処理機能が関数として実装されています．こうした様々な関数は，全て複数のデータ格納領域 (バッファ) またはデータファイルを対象として処理を実行することができ，関数を組み合わせて処理を記述することによって，計測されたデータの信号処理，統計処理をはじめ多角的なデータ解析を行うことができます．

4.2. ISPP のコマンド体系

ISPP の関数は 表 4.1.「ISPP 関数一覧」 に示すように，データの生成，操作，補間，算術演算，解析に分類されます．こうした基本的な機能を有した関数を組み合わせることによって，複雑なデータ解析も行うことができます．

表 4.1. ISPP 関数一覧

データ生成	
argen	AR モデルによるデータを生成する
gauss2	2 次元ガウス関数を生成する
arand	任意の確率分布に従う乱数データを生成する
mnrand	多次元正規乱数データを生成する
nrand	正規乱数データを生成する
urand	一様乱数データを生成する
データ操作	
dccut	バッファの直流成分を除去する
norm	指定したバッファの内容を正規化する
データ補間	
interp2	2 次元データの補間を行う
算術演算	
det	2 次元データを行列とみなし，行列式を求める
eigen	2 次元データを行列とみなし，固有値・固有ベクトルを求める
sum	バッファの内容を積分する
inv	2 次元データを行列とみなし，逆行列を求める
mul	2 つの 2 次元データを行列とみなし，積を求める
trans	2 次元データを行列とみなし，転置行列を求める
nmeq	正規方程式を解く
データ解析	
burg	Burg 法よりパワースペクトルと AIC を求める
cep	データの複素ケプストラムを求める
fftc	複素数高速 (逆) フーリエ変換を求める
fftn	多次元データに対しての複素数高速 (逆) フーリエ変換を求める
fir	データに対して FIR 型フィルタ処理を行う
firmake	FIR 型フィルタを設計する
hil	ヒルベルト変換を行う
butwmake	バタワース特性の IIR 型フィルタを設計する
icep	逆ケプストラム解析を行う
iir	データに対して IIR 型フィルタ処理を行う
iirmake	零点，極から IIR 型フィルタの係数を求める
levin	Levinson-Durbin 法によりパワースペクトルと AIC を求める
phase	2 つのバッファの内容により位相を求める
pole	AR 係数から極の位置を求める
power	2 つのバッファの内容を 2 乗して加算する
rank	データの度数分布，ガウス分布を求める
spcf	データのパワースペクトル，位相を求める
window	データに対してウィンドウ処理を行う

4.3. 使用例

ISPP の基本的な使用例として，主にフーリエ変換，フィルタ処理，行列演算について，以下にその例を示します．

4.3.1. フーリエ変換

ここでは，2 つの正弦波を合成した信号に雑音を重畳させた信号をフーリエ変換する一連の手順を，実際の計算結果とともに示します．

データの生成

正弦波の生成

```
[]SATELLITE[]~/home/demo:[1]% sampling(4000);
[]SATELLITE[]~/home/demo:[2]% t = (0~1999)/2000;
```

サンプリング周波数，時刻データを設定し，series 変数 (t) に格納します．

```
[]SATELLITE[]~/home/demo:[3]% a = 5*sin(2*PI*20*t) + 3;
[]SATELLITE[]~/home/demo:[4]% b = 3*sin(2*PI*50*t) + 3;
```

周波数，振幅が異なり，直流成分を持つ正弦波を series 変数 (a, b) に格納します．

データ切り出し生成された正弦波の 0 点目から 1023 点目までの，計 1024 点のデータを切り出します．

```
acut =
bcut =
```

切り出したデータを格納する series 変数を決めます．

```
acut = cut(a,
bcut = cut(b,
```

切り出すデータを設定します．

```
acut = cut(a,0
bcut = cut(b,0
```

切り出すデータの始点を設定します．

```
[]SATELLITE[]~/home/demo:[5]% acut = cut(a,0,1023);
[]SATELLITE[]~/home/demo:[6]% bcut = cut(b,0,1023);
```

切り出すデータの終点を設定します．

以上で切り出したデータが series 変数 (acut, bcut) に格納されます．

乱数の生成

正規乱数を発生させる場合には，nrand 関数を用います．

```
nois =
```

発生させた乱数を格納する series 変数を決めます．

```
nois = nrand(1024,
```

発生させるデータ点数を設定します．

```
nois = nrand(1024,1
```

乱数の初期値 (奇数) を設定します.

```
nois = nrand(1024,1,0
```

乱数の平均値を設定します.

```
[]SATELLITE[]~/home/demo:[7]% nois = nrand(1024,1,0,1);
```

乱数の分散を設定します.

以上でデータ点数 1024 点，平均 0，分散 1 の正規乱数データが series 変数 (nois) に格納されます．なお，一様乱数を発生させる場合には，urand 関数を用います.

信号の合成

以上の作業により得られた信号を合成します.

```
data =
```

合成信号を格納する series 変数を決めます.

```
[]SATELLITE[]~/home/demo:[8]% data = acut + bcut + nois;
```

各信号を合成します.

以上で 2 つの正弦波データと正規乱数データを合成したデータが series 変数 (data) に格納されます.

```
[]SATELLITE[]~/home/demo:[9]% wopen(1,"A4",0,0);
[]SATELLITE[]~/home/demo:[10]% x = 0~1023;
[]SATELLITE[]~/home/demo:[11]% size(100,50);
[]SATELLITE[]~/home/demo:[12]% scale("N","F","N","F",0,1000,-20,20);
[]SATELLITE[]~/home/demo:[13]% graph(data,x,0,0,0,0,0);
[]SATELLITE[]~/home/demo:[14]% title(1,"Time[sec]","Amplitude");
[]SATELLITE[]~/home/demo:[15]% axis(1,1,"XY","XY",5,0,3,200,20,1);
[]SATELLITE[]~/home/demo:[16]% frame();
```

合成した信号の波形を表示します．これらの関数はグラフィック・システム (GPM) で詳しく説明します.

合成した信号の波形を 図 4.1.「2 つの正弦波および雑音を足し合わせた信号の波形」 に示します.

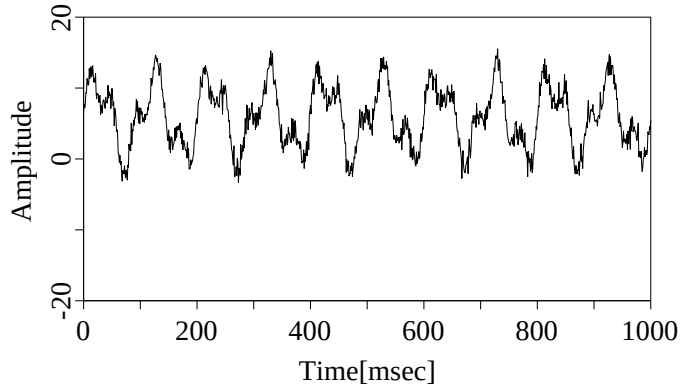


図 4.1. 2 つの正弦波および雑音を足し合わせた信号の波形

データの前処理

ある信号に FFT を施す前に行うべき前処理として，以下に直流分除去とウィンドウ処理の方法を示します.

直流成分除去

データの直流成分を除去する場合には，dccut 関数を用います.

```
data1 =
```

直流成分を除去したデータを格納する series 変数を決めます.

```
[]SATELLITE[]~/home/demo:[17]% data1 = dccut(data);
```

直流成分除去対象データを設定します.

以上で直流成分が除去されたデータが series 変数 (data1) に格納されます.

直流成分を除去した信号の表示

直流成分を除去した信号波形を表示します.

```
[]SATELLITE[]~/home/demo:[18]% wopen(1,"A4",0,0);
[]SATELLITE[]~/home/demo:[19]% size(100,50);
[]SATELLITE[]~/home/demo:[20]% scale("N","F","N","F",0,1000,-20,20);
[]SATELLITE[]~/home/demo:[21]% graph(data1,x,0,0,0,0,0);
[]SATELLITE[]~/home/demo:[22]% title(1,"Time[sec]","Amplitude");
[]SATELLITE[]~/home/demo:[23]% axis(1,1,"XY","XY",5,0,3,200,20,1);
[]SATELLITE[]~/home/demo:[24]% frame();
```

直流成分を除去した信号を表示します．直流成分を除去した信号波形を 図 4.2.「直流成分除去後の波形」 に示します.

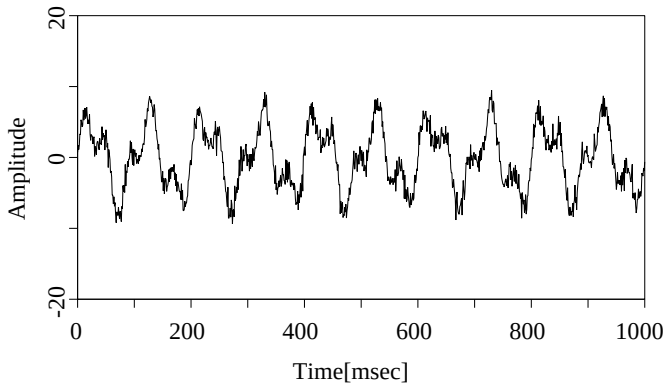


図 4.2. 直流成分除去後の波形

ウィンドウ処理

データにウィンドウ処理を施す場合には，window 関数を用います.

```
data2 =
```

ウィンドウ処理後のデータを格納する series 変数を決めます.

```
data2 = window(data1,
```

ウィンドウ処理対象データを設定します.

```
data2 = window(data1,1,
```

ウィンドウの種類 (1 : ハミング窓，2 : ハニング窓，3 : ブラックマン窓，4 : トライアングル窓) を設定します.

```
[]SATELLITE[]~/home/demo:[25]% data2 = window(data1,1,0);
```

データの補正 (補正を行うと，ウィンドウ処理前後の積分値が等しくなります) をしない場合は 0，する場合は 1 と設定します.

以上でウィンドウ処理されたデータが series 変数 (data2) に格納されます.

ウィンドウ処理した信号の表示

ウィンドウ処理をした信号波形を表示します.

```
[]SATELLITE[]~/home/demo:[26]% wopen(1,"A4",0,0);
[]SATELLITE[]~/home/demo:[27]% size(100,50);
[]SATELLITE[]~/home/demo:[28]% scale("N","F","N","F",0,1000,-20,20);
[]SATELLITE[]~/home/demo:[29]% graph(data2,x,0,0,0,0,0);
[]SATELLITE[]~/home/demo:[30]% title(1,"Time[sec]","Amplitude");
[]SATELLITE[]~/home/demo:[31]% axis(1,1,"XY","XY",5,0,3,200,20,1);
[]SATELLITE[]~/home/demo:[32]% frame();
```

ウィンドウ処理後の信号波形を 図 4.3.「ウィンドウ処理後の波形」 に示します.

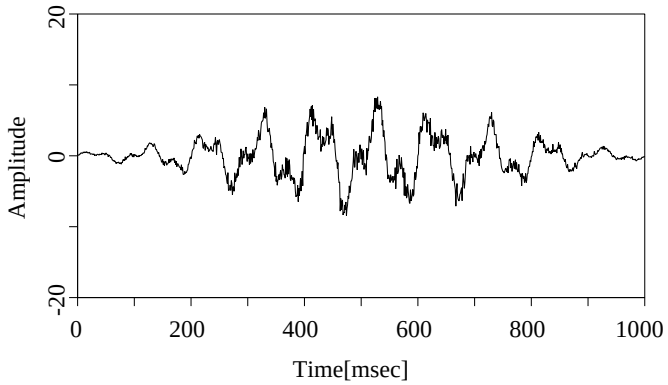


図 4.3. ウィンドウ処理後の波形

フーリエ変換

fftc 関数により，複素数データのフーリエ変換および，逆フーリエ変換を行うことができます．なお，フーリエ変換には FFT 法を用いるため，変換対象データの点数は，2 のべき乗である必要があります.

```
[]SATELLITE[]~/home/demo:[33]% series Rout,Iout;
```

出力時系列の実部と虚部を格納する series 変数を宣言します.

```
[]SATELLITE[]~/home/demo:[34]% rei = (0~1023)*0;
```

入力信号に虚部データが存在しない場合 0 を入力するため，あらかじめ series 変数 (rei) に 1024 点の 0 を格納します.

```
fftc("P",
```

計算方式フラグ (P : フーリエ変換，I : 逆フーリエ変換) を設定します.

```
fftc("P",data2,
```

変換対象データの実部を入力します.

```
fftc("P",data2,rei,
```

変換対象データの虚部を入力します.

```
fftc("P",data2,rei,Rout,
```

フーリエ変換後の実部データを格納する series 変数を設定します.

```
[]SATELLITE[]~/home/demo:[35]% fftc("P",data2,rei,Rout,Iout);
```

フーリエ変換後の虚部データを格納する series 変数を設定します.

以上でフーリエ変換されたデータが series 変数 (Rout, Iout) に格納されます.

```
[]SATELLITE[]~/home/demo:[36]% wopen(1,"A4",0,0);
[]SATELLITE[]~/home/demo:[37]% size(100,50);
[]SATELLITE[]~/home/demo:[38]% scale("N","F","N","F",0,100,-1000,1000);
[]SATELLITE[]~/home/demo:[39]% graph(Rout,"F",0,0,0,0,0);
[]SATELLITE[]~/home/demo:[40]% ltype(5,1);
[]SATELLITE[]~/home/demo:[41]% graph(Rout,"F",0,0,0,0,0);
[]SATELLITE[]~/home/demo:[42]% title(1,"Frequency[sec]","Amplitude");
[]SATELLITE[]~/home/demo:[43]% axis(1,1,"XY","XY",5,0,3,20,1000,1);
[]SATELLITE[]~/home/demo:[44]% frame();
```

フーリエ変換後の信号を表示します.

フーリエ変換後の信号波形を 図 4.4. 「FFT 処理後の信号波形 (実部 : 実線，虚部 : 破線)」 に示します.

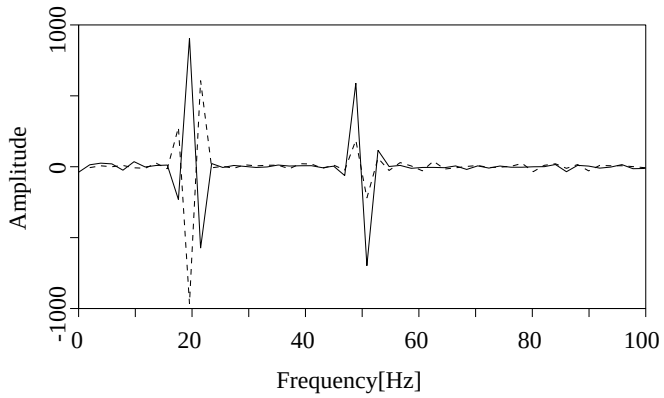


図 4.4. FFT 処理後の信号波形 (実部 : 実線，虚部 : 破線)

パワースペクトル

2 つの変数の内容を 2 乗して加算する場合には，power 関数を用います．この関数により，fftc 関数により得られたフーリエ変換後の実部データおよび虚部データから，入力時系列のパワースペクトルを求めることができます．また，入力時系列から直接パワースペクトルを求めることのできる，spcf 関数もあります．spcf 関数については，後に説明します.

```
pw =
```

求めたパワースペクトルを格納する series 変数を決めます.

```
pw = power(Rout,
```

フーリエ変換後の実部データを入力します.

```
[]SATELLITE[]~/home/demo:[45]% pw = power(Rout,Iout);
```

フーリエ変換後の虚部データを入力します.

以上で，パワースペクトルデータが series 変数 (pw) に格納されます.

```
[]SATELLITE[]~/home/demo:[46]% wopen(1,"A4",0,0);
[]SATELLITE[]~/home/demo:[47]% size(100,50);
[]SATELLITE[]~/home/demo:[48]% scale("N","F","N","F",0,100,-200000,2000000);
[]SATELLITE[]~/home/demo:[49]% ltype(1,1);
```

```
[]SATELLITE[]~/home/demo:[50]% graph(pw,"F",0,0,0,0,0);
[]SATELLITE[]~/home/demo:[51]% title(1,"Frequency[sec]","Power");
[]SATELLITE[]~/home/demo:[52]% axis(1,1,"XY","XY",5,0,3,20,1000000,0);
[]SATELLITE[]~/home/demo:[53]% frame();
```

求めたパワースペクトルを表示します.

求めたパワースペクトルを 図 4.5.「パワースペクトル」 に示します.

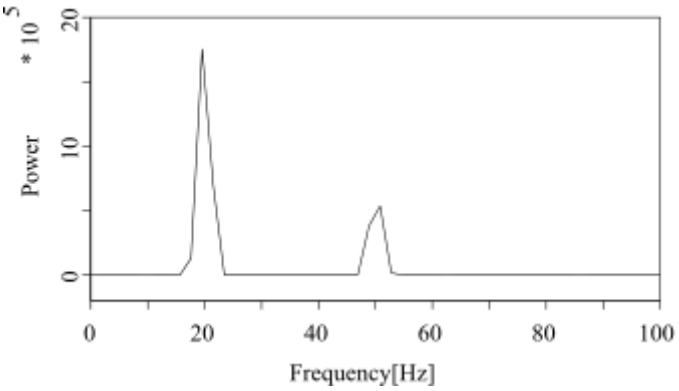


図 4.5. パワースペクトル

位相特性

2 つの変数の内容により位相を求める場合には，phase 関数を用います．この関数により，fftc 関数により得られたフーリエ変換後の実部データおよび虚部データから，その位相特性を求めることができます.

```
phs =
```

求めた位相を格納する series 変数を決めます.

```
phs = phase(Rout,
```

フーリエ変換後の実部データを入力します.

```
phs = phase(Rout,Iout
```

フーリエ変換後の虚部データを入力します.

```
phs = phase(Rout,Iout,"D",
```

出力データ形式 ("D" : degree，"O" : radian) を設定します.

```
[]SATELLITE[]~/home/demo:[54]% phs = phase(Rout,Iout,"D","U");
```

位相戻しをする場合は "U"，しない場合は "O" と設定します.

以上で位相データが series 変数 (phs) に格納されます.

入力系列からパワースペクトルと位相を 1 つの関数で求める方法

spcf 関数により，ある入力時系列から，そのパワースペクトルおよび位相を，同時に求めることができます．以下にその具体的な手順を示します.

```
[]SATELLITE[]~/home/demo:[55]% series pw,phs;
```

パワースペクトルおよび位相を格納する series 変数を宣言します.

```
spcf(data2,
```

対象時系列を設定します.

```
spcf(data2,pw,
```

求めたパワースペクトルを格納する series 変数を設定します.

```
[]SATELLITE[]~/home/demo:[56]% spcf(data2,pw,phs);
```

求めた位相を格納する series 変数を設定します.

以上でパワースペクトル，位相データが series 変数 (pw，phs) に格納されます.

4.3.2. フィルタ処理

MA フィルタ

移動平均法を用いたフィルタにより，信号を平滑化する手順を以下に示します．なおフィルタの次数は 5 次とします.

```
[]SATELLITE[]~/home/demo:[1]% coef = (1/5,1/5,1/5,1/5,1/5);
```

フィルタ係数を series 変数に格納します.

```
output =
```

平滑化された信号データを格納する変数を決めます.

```
output = fir(coef,
```

fir 関数により，時系列信号を平滑化します．まず，フィルタの係数を設定します.

```
[]SATELLITE[]~/home/demo:[2]% output = fir(coef, input);
```

平滑化の対象となる時系列信号 (input) を設定します.

以上で平滑化されたデータが series 変数 (output) に格納されます．なお，fir 関数は，図 4.6.「fir 関数のフィルタ処理概念図」(a_1, a_2, a_3, a_4, a_5 は，フィルタ係数) の様な構成で処理を行います．したがって，そのフィルタ係数となるパラメータ系列の長さは，奇数値 (以下，2n+1 とする) でなければなりません．また，処理対象となる時系列データの最初から n 点までのデータについては，適切な処理を行うことができないため，n+1 点目のデータを採用します．時系列データの最後から n+1 点までのデータについても同様で，最後から n+1 点目のデータを採用します.

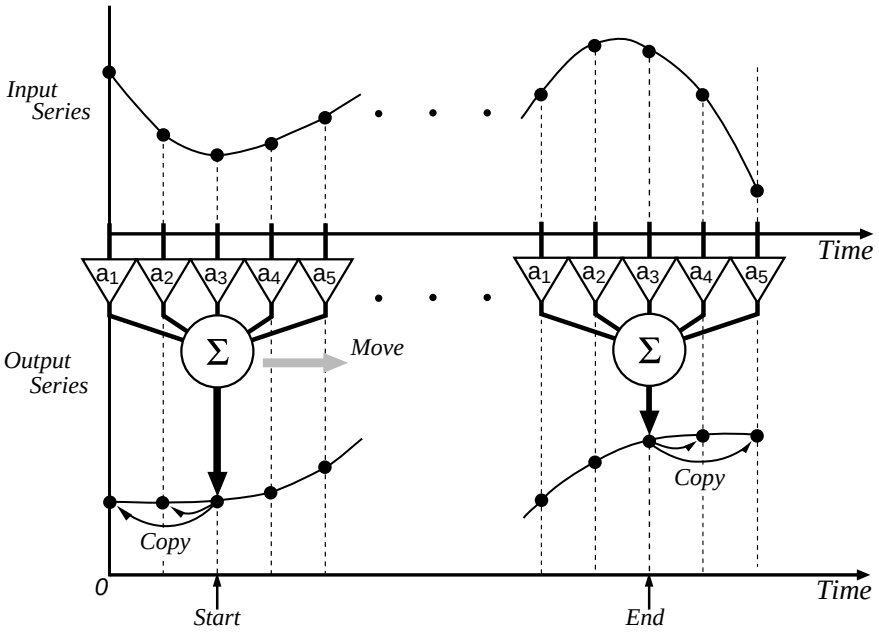


図 4.6. fir 関数のフィルタ処理概念図

FIR フィルタ

ローパス，ハイパス，バンドパスの FIR ファイルを，窓関数法により設計する場合には，まず firmake 関数によりフィルタ係数を求め，fir 関数によりフィルタリングを行います．以下に，ローパスフィルタの場合について，その具体的な手順を示します．

```
[]SATELLITE[]~/home/demo:[3]% sampling(1024);
```

サンプリング周波数を設定します．

```
coef =
```

求めたフィルタ係数を格納する series 変数を入力します．

```
coef = firmake(1,
```

設計するフィルタ特性をローパスフィルタに設定します．(1 : LowPass，2 : HighPass，3 : BandPass)

```
coef = firmake(1,11
```

フィルタの次数を設定します．ただし，実際のフィルタリングには fir 関数を用いるため，次数は奇数値でなければなりません．

```
coef = firmake(1,11,100
```

遮断周波数を設定します．

```
[]SATELLITE[]~/home/demo:[4]% coef = firmake(1,11,100,3);
```

使用する窓関数の種類を設定します．(0 : 矩形窓，1 : ハニング窓，2 : ハミング窓，3 : ブラックマン窓，4 : カイザー窓)

以上で，遮断周波数 100Hz の 11 次 FIR 型ローパスフィルタのフィルタ係数が series 変数 (coef) に格納されます．なお，ハイパスの場合には，同様に，

```
coef = firmake(2,11,400,3);
```

と設定できます．この例では，遮断周波数 400Hz の 11 次 FIR 型ハイパスフィルタが設計されます．また，バンドパスの場合には，遮断周波数を 2 つ設定する必要があるので，series で遮断周波数を設定します．例えば，遮断周波数が 100Hz，400Hz で，フィルタ次数が 11 次の場合には，

```
coef = firmake(3,11,(100,400),3);
```

とします．

次に，coef を用いて fir 関数によりフィルタリングします．

```
[]SATELLITE[]~/home/demo:[5]% output = fir(coef,input);
```

IIR フィルタ

バタワース特性を持った，ローパス，ハイパス，バンドパスに IIR フィルタを設計する場合には，まず，butwmake 関数により伝達関数の零点，極，利得を求めます．次に，iirmake 関数により零点，極をフィルタ係数に変換し，fir，iir 関数によりフィルタリングを行います．以下に，ローパスフィルタの場合について，その具体的な手順を示します．

```
[]SATELLITE[]~/home/demo:[6]% sampling(1024);
```

サンプリング周波数を設定します．

```
[]SATELLITE[]~/home/demo:[7]% series zr,zi,pr,pi;
```

伝達関数の零点，極の座標を格納する series 変数を宣言します．

```
gain =
```

設計したフィルタの利得を格納する scalar 変数を入力します．

```
gain = butwmake(1,100,
```

ローパスフィルタの指定と遮断周波数を設定します．

```
gain = butwmake(1,100,13
```

フィルタの次数を設定します．ただし，FIR フィルタの場合と同様に，奇数値を設定しなければなりません．また，最大次数は 101 次です．

```
gain = butwmake(1,100,13,zr,zi,
```

伝達関数の零点の座標 (実部と虚部) を設定します．

```
[]SATELLITE[]~/home/demo:[8]% gain = butwmake(1,100,13,zr,zi,pr,pi);
```

伝達関数の極の座標 (実部と虚部) を設定します．

以上でバタワース特性を持った遮断周波数 100Hz の 13 次 IIR 型ローパスフィルタにおいて，その伝達関数の零点 (zr,zi)，極 (pr,pi)，利得 (gain) が series 変数に格納されます．なお，ハイパスの場合には，同様に，

```
gain = butwmake(2,400,13,zr,zi,pr,pi);
```

設計されます．また，バンドパスの場合は遮断周波数を 2 つ設定する必要があるので，例えば遮断周波数が 100Hz，400Hz で，フィルタ次数が 13 次の場合には，

```
gain = butwmake(3,(100,400),13,zr,zi,pr,pi);
```

とします．

次に，zr，zi，pr，pi から，iirmake 関数によりフィルタ係数を算出します．

```
[]SATELLITE[]~/home/demo:[9]% series a,b;
```

伝達関数の分母の係数，及び分子の係数を格納する series 変数を宣言します．

```
iirmake(zr,zi,pr,pi,
```

伝達関数の零点，極の座標を設定します．

```
iirmake(zr,zi,pr,pi,a,
```

伝達関数の分母の係数を設定します．

```
[]SATELLITE[]~/home/demo:[10]% iirmake(zr,zi,pr,pi,a,b);
```

伝達関数の分子の係数を設定します．

以上でフィルタ係数が fir，iir 関数にそのまま使用できるフォーマットで出力されます．後は，実際にフィルタリングを行います．

```
[]SATELLITE[]~/home/demo:[11]% temp = fir(b,input);
```

伝達関数の分子の計算を行います．

```
[]SATELLITE[]~/home/demo:[12]% output = iir(a,temp)*gain;
```

伝達関数の分母の計算を行い，利得を乗ずると出力が求まります．

以上で，フィルタリングされた結果が，series 変数 (output) に格納されます．

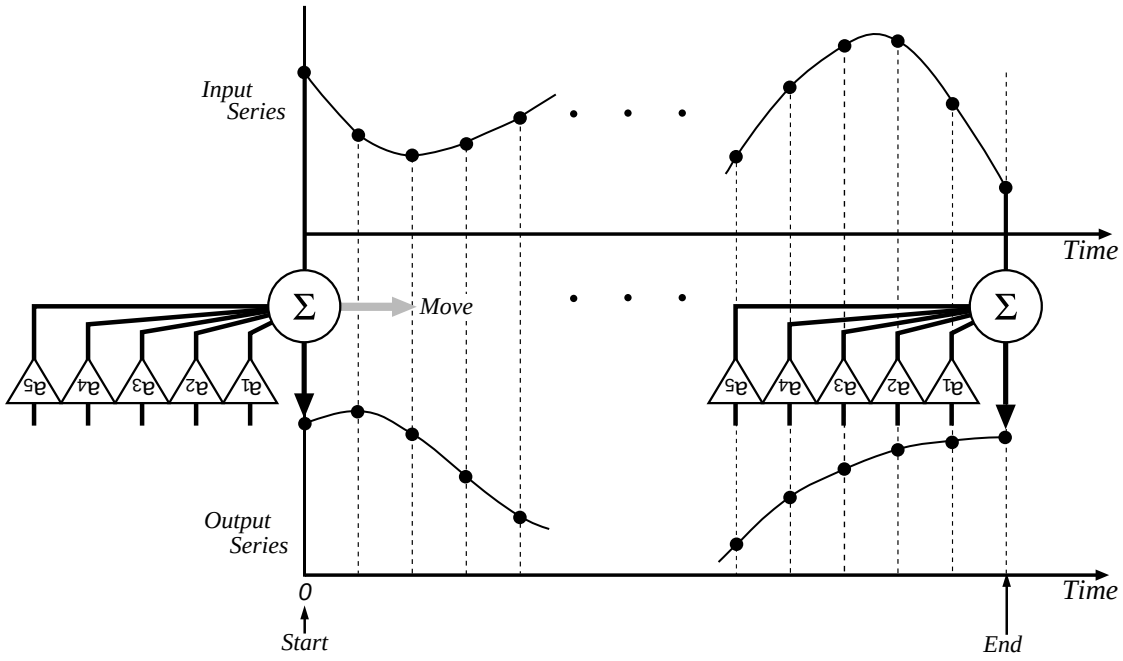


図 4.7. iir 関数のフィルタ処理概念図

実際の計算例

実際に，バタワース特性を持った遮断周波数 100Hz の 13 次 IIR 型ローパスフィルタを設計し，インパルス応答を求めることでフィルタ特性を求めてみます．以下に，その手順を示します．

```
[]SATELLITE[]~/home/demo:[13]% sampling(512);
```

サンプリング周波数を設定します．

```
[]SATELLITE[]~/home/demo:[14]% series zr,zi,pr,pi;
[]SATELLITE[]~/home/demo:[15]% series a,b;
[]SATELLITE[]~/home/demo:[16]% series u,v;
```

butwmake 関数，iirmake 関数，spcf 関数で必要な変数をそれぞれ宣言します．

```
[]SATELLITE[]~/home/demo:[17]% delay = 50;
[]SATELLITE[]~/home/demo:[18]% datp = 511;
[]SATELLITE[]~/home/demo:[19]% impulse = (1,(1~datp)*0);
[]SATELLITE[]~/home/demo:[20]% d_impulse = ((0~delay)*0,impulse);
```

インパルス信号を生成します．

```
[]SATELLITE[]~/home/demo:[21]% wopen(1,"A4",0,0);
[]SATELLITE[]~/home/demo:[22]% x = 0~205;
[]SATELLITE[]~/home/demo:[23]% size(100,50);
[]SATELLITE[]~/home/demo:[24]% scale("N","F","N","A",0,205);
[]SATELLITE[]~/home/demo:[25]% graph(impulse,x,0,0,0,0,0);
[]SATELLITE[]~/home/demo:[26]% title(1,"Data Point","Amplitude");
[]SATELLITE[]~/home/demo:[27]% axis(1,1,"XY","XY",5,0,1,50,0.5,0);
[]SATELLITE[]~/home/demo:[28]% origin(120,20);
[]SATELLITE[]~/home/demo:[29]% graph(d_impulse,x,0,0,0,0,0);
[]SATELLITE[]~/home/demo:[30]% title(1,"Data Point","Amplitude");
[]SATELLITE[]~/home/demo:[31]% axis(1,1,"XY","XY",5,0,1,50,0.5,0);
```

生成したインパルス信号を出力します．

ここで，impulse (図 4.8.「impulse 信号」) の前に delay 個の 0 を詰め，d_impulse (図 4.9.「d_impulse 信号」)を生成した理由は，fir 関数の仕様上，フィルタの次数以内の出力が決定できないためです．つまり，インパルス応答を求めるためには，d_impulse をフィルタリング (図 4.10.「firtemp 信号」) した後に，delay 分だけシフト (図 4.11.「firinp 信号」) することにより求めます．

また，図 4.6.「fir 関数のフィルタ処理概念図」 に示した様に，fir 関数は現在の出力を求めるのに未来の入力を用いています．これでは，因果性を持たないので，iirmake 関数は伝達関数の分子の係数に，(フィルタの次数−1) 個の 0 を詰めた形でその係数を出力します．このため，フィルタリングの対象になるのは，((2×フィルタの次数)−1) 個目より後のデータとなります．この場合 2×13−1 ＝ 25 ですから，25 よりも大きな値 (50) を delay として定義し，前述の方法でフィルタリングすることにより，その正確な出力を求めることができます．

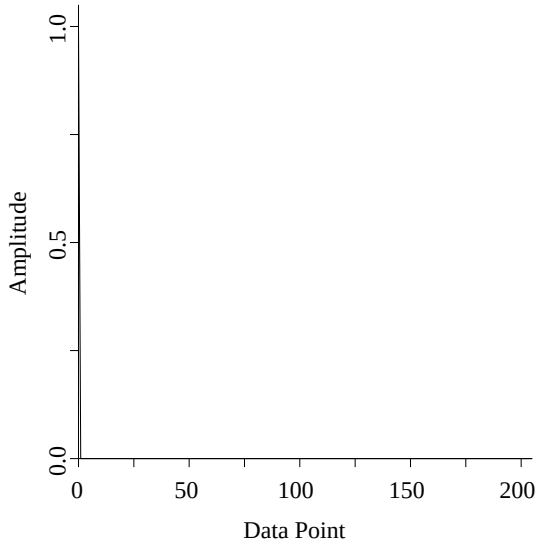


図 4.8. impulse 信号

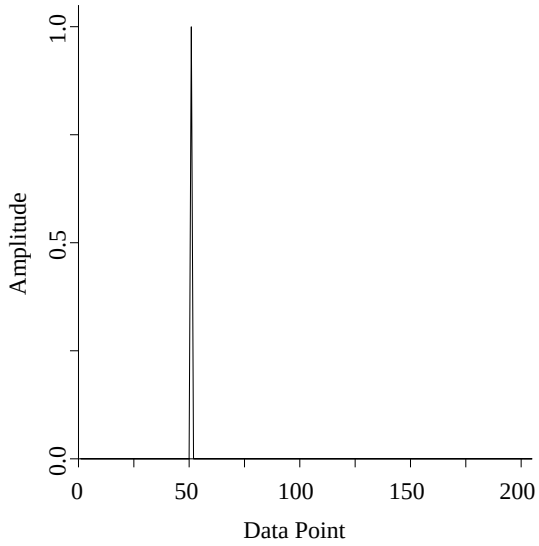


図 4.9. d_impulse 信号

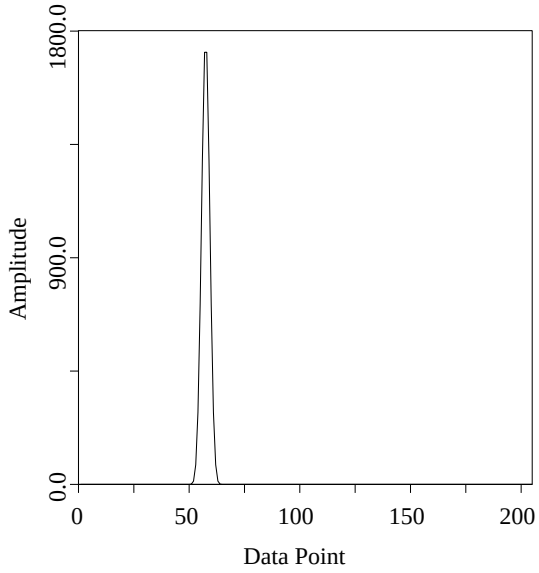


図 4.10. firtemp 信号

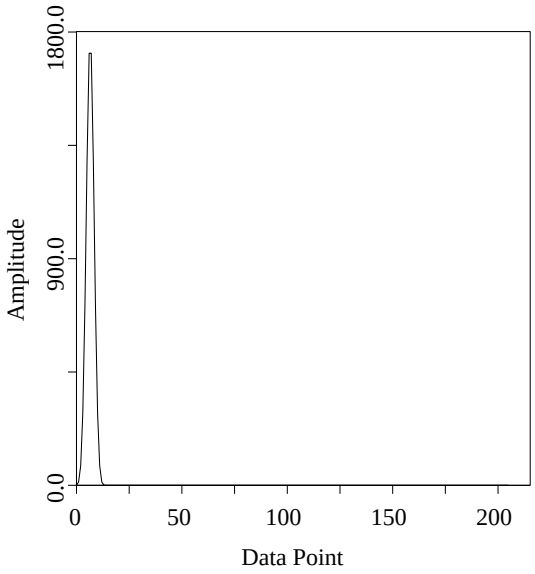


図 4.11. firinp 信号

```
[]SATELLITE[]~/home/demo:[32]% gain = butwmake(1,100,13,zr,zi,pr,pi);
[]SATELLITE[]~/home/demo:[33]% iirmake(zr,zi,pr,pi,a,b);
```

フィルタの伝達関数を求め，フィルタを設計します.

```
[]SATELLITE[]~/home/demo:[34]% firtemp = fir(b,d_impulse);
[]SATELLITE[]~/home/demo:[35]% firinp = cut(firtemp,delay+1,datp+delay+1);
```

伝達関数の分子の部分の計算を行い，impulse の前に詰めた 0 を取り除くことにより，delay 分だけシフトします.

```
[]SATELLITE[]~/home/demo:[36]% wopen(1,"A4",0,0);
[]SATELLITE[]~/home/demo:[37]% x = 0~205;
[]SATELLITE[]~/home/demo:[38]% size(100,50);
[]SATELLITE[]~/home/demo:[39]% scale("N","F","N","A",0,205);
[]SATELLITE[]~/home/demo:[40]% graph(firtemp,x,0,0,0,0,0);
[]SATELLITE[]~/home/demo:[41]% title(1,"Data Point","Amplitude");
[]SATELLITE[]~/home/demo:[42]% axis(1,1,"XY","XY",5,0,1,50,900,0);
[]SATELLITE[]~/home/demo:[43]% origin(120,20);
[]SATELLITE[]~/home/demo:[44]% graph(firinp,x,0,0,0,0,0);
[]SATELLITE[]~/home/demo:[45]% title(1,"Data Point","Amplitude");
[]SATELLITE[]~/home/demo:[46]% axis(1,1,"XY","XY",5,0,1,50,900,0);
```

iir 関数によるインパルス応答を表示します.

```
[]SATELLITE[]~/home/demo:[47]% output = iir(a,firinp)*gain;
```

伝達関数の分母の部分の計算を行い，利得を乗じ，設計したフィルタのインパルス応答を求めます.

```
[]SATELLITE[]~/home/demo:[48]% wopen(1,"A4",0,0);
[]SATELLITE[]~/home/demo:[49]% x = 0~160;
[]SATELLITE[]~/home/demo:[50]% size(100,50);
[]SATELLITE[]~/home/demo:[51]% scale("N","F","N","A",0,160);
[]SATELLITE[]~/home/demo:[52]% graph(output,x,0,0,0,0,0);
[]SATELLITE[]~/home/demo:[53]% title(1,"Data Point","Amplitude");
[]SATELLITE[]~/home/demo:[54]% axis(1,1,"XY","XY",5,0,1,20,0.3,0);
```

フィルタのインパルス応答を表示します (図 4.12.「設計したフィルタのインパルス応答」)

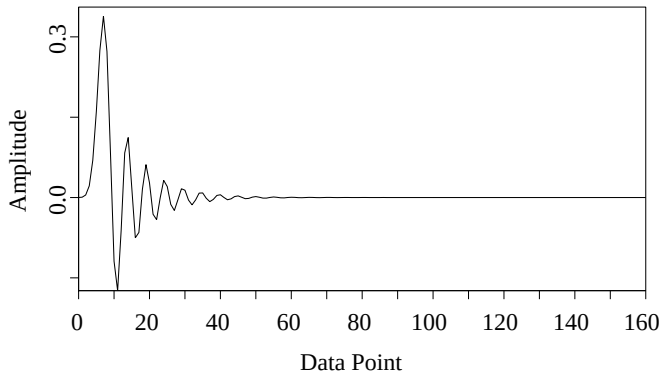


図 4.12. 設計したフィルタのインパルス応答

```
[]SATELLITE[]~/home/demo:[55]% spcf(output,u,v);
```

output をフーリエ変換し，パワースペクトル (u) を求めることにより，設計したフィルタの振幅特性を求めます.

```
[]SATELLITE[]~/home/demo:[56]% wopen(1,"A4",0,0);
[]SATELLITE[]~/home/demo:[57]% size(100,50);
[]SATELLITE[]~/home/demo:[58]% scale("N","A","N","A");
[]SATELLITE[]~/home/demo:[59]% graph(u,"F",0,0,0,0,0);
[]SATELLITE[]~/home/demo:[60]% title(1,"Frequency[Hz]","Power");
[]SATELLITE[]~/home/demo:[61]% axis(1,1,"XY","XY",5,0,1,50,0.000004,0);
```

設計したフィルタの振幅特性を表示します (図 4.13.「設計したフィルタの振幅特性」)

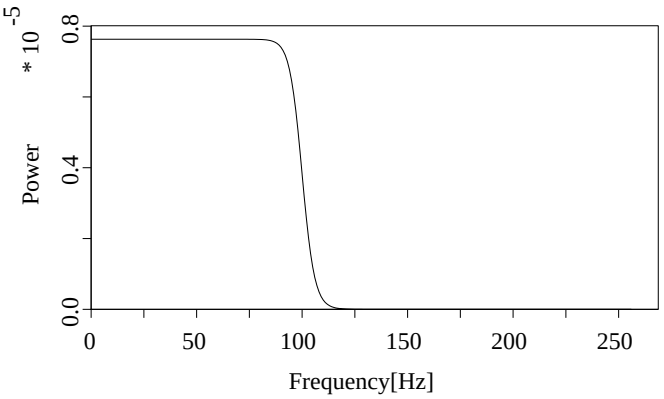


図 4.13. 設計したフィルタの振幅特性

4.3.3. 2 次元 series の扱い方

ISPP には，1 次元データの他に 2 次元データを扱うことができる，fir，fftn といった関数があります．これらの関数での 2 次元データの扱い方について説明します.

2 次元 FIR フィルタ

```
[]SATELLITE[]~/home/demo:[1]% x = (1~441)*0;
[]SATELLITE[]~/home/demo:[2]% x:[220] = 1;
[]SATELLITE[]~/home/demo:[3]% input = reform(x,(21,21));
```

まず，2 次元データを作成します.

```
[]SATELLITE[]~/home/demo:[4]% wopen(1,"A4",0,0);
[]SATELLITE[]~/home/demo:[5]% size(80,80);
[]SATELLITE[]~/home/demo:[6]% scale("N","A","N","A");
[]SATELLITE[]~/home/demo:[7]% gsolm(input,0.3,0.4,0,0,0,7,0,1,1,"X",1,0,5);
```

作成したデータを表示します (図 4.14. 「作成した 2 次元データ」).

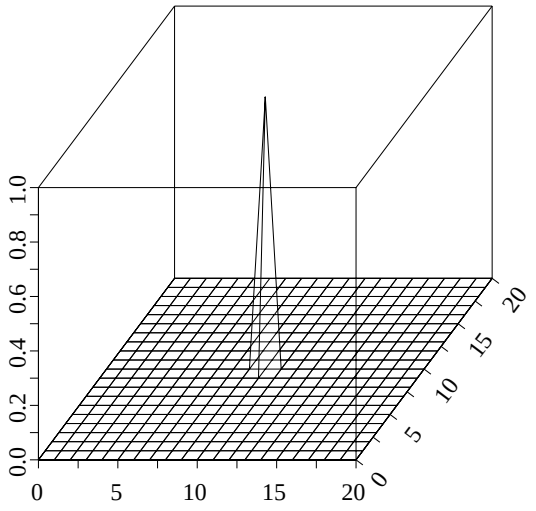


図 4.14. 作成した 2 次元データ

```
[]SATELLITE[]~/home/demo:[8]% coeftemp = (1~25)*0+1;
[]SATELLITE[]~/home/demo:[9]% coef = reform(coeftemp,(5,5));
```

次に，フィルタ係数を決定します．ここでは簡単な例として，全ての係数が 1 の場合について考えます．また，係数データは奇数×奇数である必要があります．

```
output = fir(coef,
```

fir 関数により，時系列信号を平滑化します．まず，フィルタ係数を設定します．

```
[]SATELLITE[]~/home/demo:[10]% output = fir(coef,input);
```

平滑化の対象となる時系列信号 (input) を設定します．

```
[]SATELLITE[]~/home/demo:[11]% output2 = fir(coef,input,0.0);
```

このとき，3 つ目の引数として scalar 型の値を与えた場合には，入力データからはみ出した部分はその値で補間し，2 つ目しか引数を与えなかった場合には，入力全体の平均値で補間します．

```
[]SATELLITE[]~/home/demo:[12]% wopen(1,"A4",0,0);
[]SATELLITE[]~/home/demo:[13]% size(80,80);
[]SATELLITE[]~/home/demo:[14]% scale("N","A","N","A");
[]SATELLITE[]~/home/demo:[15]% gsolm(output2,0.3,0.4,0,0,0,7,0,1,1,"X",1,0,5);
```

求めたフィルタの応答を表示します (図 4.15. 「2 次元 FIR フィルタの応答」)

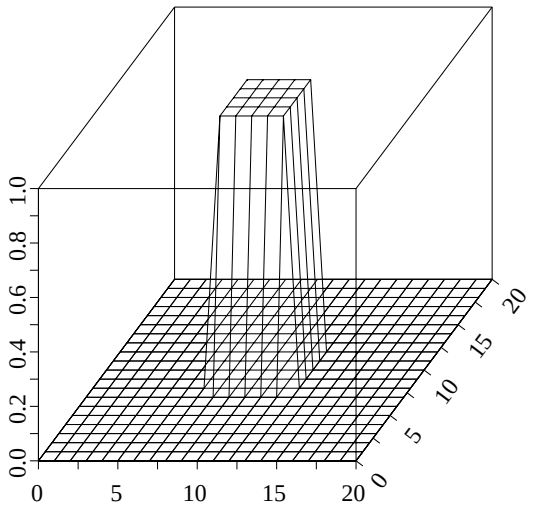


図 4.15. 2 次元 FIR フィルタの応答

2 次元フーリエ変換

```
[]SATELLITE[]~/home/demo:[16]% series u[32],v[32];
```

出力時系列の実部と虚部を格納する series 変数を宣言します.

```
[]SATELLITE[]~/home/demo:[17]% datp = 15;
[]SATELLITE[]~/home/demo:[18]% data = ((1~datp)*0,1,1,(1~datp)*0);
[]SATELLITE[]~/home/demo:[19]% x = mul(trans(data),data);
```

入力時系列の実部の 2 次元のデータを作成します.

```
[]SATELLITE[]~/home/demo:[20]% y = x*0;
```

入力信号に虚部データが存在しない場合 0 を入力するため，あらかじめ series 変数に 1024×1024 点の 0 を格納します.

```
[]SATELLITE[]~/home/demo:[21]% wopen(1,"A4",0,0);
[]SATELLITE[]~/home/demo:[22]% size(80,80);
[]SATELLITE[]~/home/demo:[23]% scale("N","A","N","A");
[]SATELLITE[]~/home/demo:[24]% gsolm(x,0.3,0.4,0,0,0,7,0,1,1,"X",1,0,5);
```

入力時系列の実部のデータを表示します (図 4.16. 「2 次元 FFT への入力波形 (実部)」)

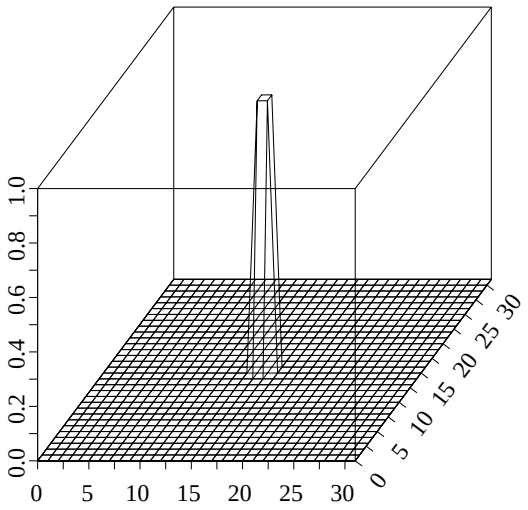


図 4.16. 2 次元 FFT への入力波形 (実部)

```
fftn("P",
```

計算方式フラグ ("P" : フーリエ変換, "I" : 逆フーリエ変換) を設定します.

```
fftn("P",x,
```

変換対象データの実部を入力します.

```
fftn("P",x,y,
```

変換対象データの虚部を入力します.

```
fftn("P",x,y,u,
```

フーリエ変換後の実部データを格納する series 変数を設定します.

```
[]SATELLITE[]~/home/demo:[25]% fftn("P",x,y,u,v);
```

フーリエ変換後の虚部データを格納する series 変数を設定します. power 関数により, fftn 関数により得られたフーリエ変換後の実部データおよび虚部データから, 入力時系列のパワースペクトルを求めます.

```
pw2 =
```

求めたパワースペクトルを格納する series 変数を決めます.

```
pw2 = power(u,
```

フーリエ変換後の実部データを入力します.

```
[]SATELLITE[]~/home/demo:[26]% pw2 = power(u,v);
```

フーリエ変換後の虚部データを入力します.

```
[]SATELLITE[]~/home/demo:[27]% wopen(1,"A4",0,0);
[]SATELLITE[]~/home/demo:[28]% size(80,80);
[]SATELLITE[]~/home/demo:[29]% scale("N","A","N","A");
[]SATELLITE[]~/home/demo:[30]% gsolm(pw2,0.3,0.4,0,0,0,7,0,1,1,"X",1,0,5);
```

求めたパワースペクトルを表示します (図 4.17. 「パワースペクトル」)

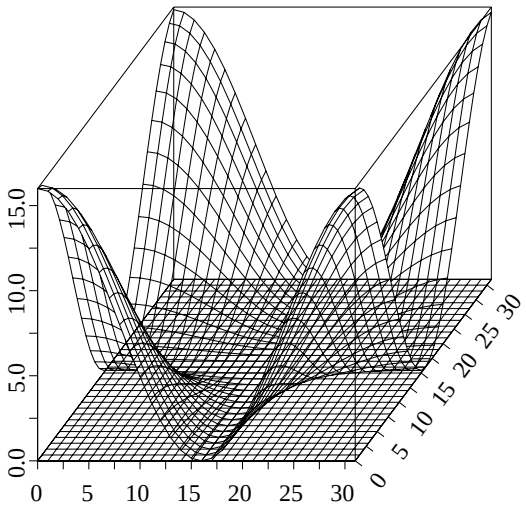


図 4.17. パワースペクトル

4.3.4. 行列演算

ISPP の特徴の一つとして，行列演算が挙げられます．n 次元 1 次方程式の解を，正規方程式を導くことで求めてみます．

$$x_{11}\theta_1 + \cdots + x_{1n}\theta_n = y_1$$

$$x_{m1}\theta_1 + \cdots + x_{mn}\theta_n = y_m$$

一般に連立方程式は，行列の積により表すことができ，n 次元 1 次方程式の場合は，次式のように表すことができます．

$$\begin{pmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \cdots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{pmatrix} \begin{pmatrix} \theta_1 \\ \vdots \\ \theta_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \qquad (4.1)$$

式 (4.1) は，以下のような行列演算を行うことにより，解くことができます

$$X\Theta = Y \qquad (4.2)$$

正規方程式とは，式 (4.2) の両辺に左側から X^T をかけた，次式で示される式でした．

$$X^T X\Theta = X^T Y$$

ここで，行列 $X^T X$ は正則行列と仮定すると，

$$\Theta = (X^T X)^{-1} X^T Y \qquad (4.3)$$

となり，式 (4.3) を解くことで，式 (4.1) の解を求めることができます．では，行列 X, Y に，次の様な数値を代入して，実際に ISPP により式 (4.3) の解を求める手順を以下に示します．

$$X = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 2 & 1 & 3 \\ 3 & 3 & 1 \\ 2 & 3 & 2 \end{pmatrix} \quad Y = \begin{pmatrix} 10 \\ 20 \\ 25 \\ 27 \\ 21 \end{pmatrix} \quad \Theta = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix}$$

ISPP には，行列演算用の関数として，積を求める mul 関数，逆行列を求める inv 関数，転置行列を求める trans 関数などがあり，これらの関数を組み合わせることによって行列演算を行います．

```
[]SATELLITE[]~/home/demo:[1]% tempx = (1,1,1,1,2,2,2,1,3,3,3,1,2,3,2);
[]SATELLITE[]~/home/demo:[2]% x = reform(tempx,(5,3));
```

まず行列 X を生成します．

```
[]SATELLITE[]~/home/demo:[3]% tempy = (10,20,25,27,21);
[]SATELLITE[]~/home/demo:[4]% y = reform(tempy,(5,1));
```

同様にして，行列 Y を生成します．

```
[]SATELLITE[]~/home/demo:[5]% xt = trans(x);
```

X の転置行列を求め，演算結果を xt に格納します．

```
[]SATELLITE[]~/home/demo:[6]% xtx = mul(xt,x);
```

X^T と X の積を求め，演算結果を xtx に格納します．

```
[]SATELLITE[]~/home/demo:[7]% ixtx = inv(xtx);
```

X^TX の逆行列 X^TX^{-1} を求め，演算結果を ixtx に格納します．

```
[]SATELLITE[]~/home/demo:[8]% ixtxxt = mul(ixtx, xt);
```

X^TX^{-1} と X^T の積を求め，演算結果を ixtxxt に格納します．

```
[]SATELLITE[]~/home/demo:[9]% theta = mul(ixtxxt,y);
```

$X^TX^{-1}X^T$ と Y の積を求めれば，解が求まります．

なお，次のように記述すれば，行列 X と行列 Y から直接に Θ を求めることができます．

```
theta = mul(mul(inv(mul(trans(x),x)),trans(x)),y);
```

以上の演算の結果，

$$\Theta = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix} = \begin{pmatrix} 5.544 \\ 1.583 \\ 4.126 \end{pmatrix}$$

と求まります．

4.3.5. データの統計処理

ここで，正規乱数を発生する ISPP の nrand 関数で生成されたデータの検定を，rank 関数を用いて行います．

まず，データを用意します(ここでは正規乱数)．

```
[]SATELLITE[]~/home/demo:[1]% x = nrand(4096,1,0,1.0);
```

これより，平均 0 分散 1.0 の正規分布に従う乱数が 4096 点生成されます．次に，rank 関数の結果を出力する変数を宣言します．

```
[]SATELLITE[]~/home/demo:[2]% series u, v, w;
```

これより，series 型の変数 u，v，w が定義されます．ここで，rank 関数を用います．

```
rank(x,
```

入力するデータ系列を関数に与えます.

```
rank(x,u,
```

Y 軸の出力時系列 : 度数を格納する変数を設定します.

```
rank(x,u,v,
```

X 軸の出力時系列 : 小分割されたデータの範囲を格納する変数を設定します.

```
rank(x,u,v,w,
```

Y 軸の出力時系列 : フィットした正規分布の密度関数値を格納する変数を設定します.

```
rank(x,u,v,w,min(x),
```

階級の最小値を設定します.

```
rank(x,u,v,w,min(x),max(x),
```

階級の最大値を設定します.

```
[]SATELLITE[]~/home/demo:[3]% rank(x,u,v.w,min(x),max(x),100);
```

階級の個数を設定します.

この関数を実行すると，u，v，w に計算結果が格納され

```
 ** MEAN VALUE    =    -0.018528
    VARIANCE      =     0.99103
    STANDARD D    =     0.9955
    3-ORDER M     =     0.0070914
    SKEWNESS      =     0.0071879
    4-ORDER M     =     3.0125
    KURTOSIS      =     0.067269
-0.01852805
```

という出力が返ってきます．上から，データの平均値，データの分散，データの標準偏差，3 次のモーメント，skewness (歪度)，4 次のモーメント，kurtosis (尖度) を表しています．skewness は分布の左右対称性の目安としてよく使われます．これは，零に近いほど平均を基準に分布の左右が対象であることを示しています．また，kurtosis は確率分布がどの程度尖ったものとなるかを示す尺度で，データが正規分布に従う場合には 3 に近くなります．rank 関数の仕様では，主に正規性の検定に用いることを想定しているため，定義式から 3 を減じた値を kurtosis として表示していますので注意して下さい.

次に，計算した結果をグラフに表示する例を示します.

```
[]SATELLITE[]~/home/demo:[4]% wopen(1,"A4",0,0);
[]SATELLITE[]~/home/demo:[5]% size(80,80);
[]SATELLITE[]~/home/demo:[6]% origin(30,30);
[]SATELLITE[]~/home/demo:[7]% title(1,"Data Point","histogram");
[]SATELLITE[]~/home/demo:[8]% scale("N","F","N","F",min(v),max(v),min(u),max(u));
[]SATELLITE[]~/home/demo:[9]% graph(u,v,0,0,0,1,3.5);
[]SATELLITE[]~/home/demo:[10]% graph(w,v,0,0,0,1,3.5);
[]SATELLITE[]~/home/demo:[11]% axis(1,1,"XY","XY",5,0,0,0,0,0);
```

これらの関数を実行すると，図 4.18.「正規乱数のヒストグラムとその正規分布」 のように正規分布およびヒストグラムが表示されます.

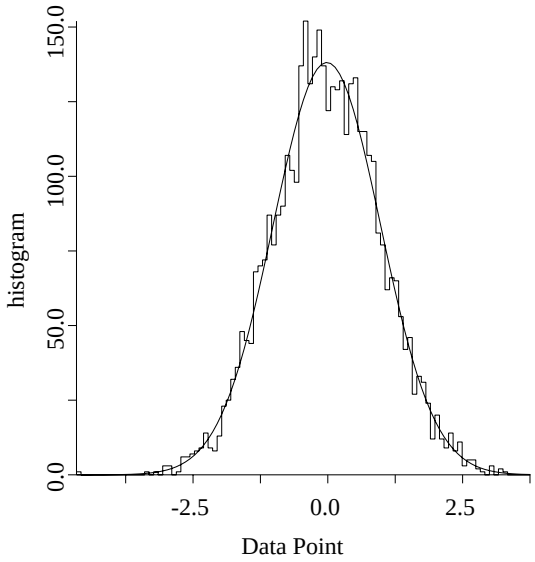


図 4.18. 正規乱数のヒストグラムとその正規分布

第 5 章 グラフィックシステム

5.1. GPM の概要

GPM は，SATELLITE において，NCS，ISPP など他のモジュールを用いて処理，解析されたデータの可視化を担当するモジュールです．データ解析においては，データを単に推知の羅列として出力するだけでなく，可視化を行い多角的に検討する必要があります．そのために多様なグラフィック機能を提供し，効率的なグラフ作成を可能とするモジュールが GPM です．GPM は論文，レポート，OHP などのグラフ作成で大いに威力を発揮します．

GPM コマンドを機能で分類すると

- グラフィックを描画する関数
- 描画パラメータを設定する関数

の 2 つに大きくわけられ，約 30 種類の関数から構成されます．主な関数を 表 5.1.「GPM コマンド」 に示します．描画関数には，

- 1 次元データを表示するもの
- 鳥瞰図，等高線図のように 2 次元データを表示するもの

など多くの関数があり，用途に応じて多様な表示が可能です．このようなグラフィックスを描画するためには，線種，線幅，描画色などの多くのパラメータが必要ですが，これらのパラメータは，" ltype() "，" lwidth() "，" color() "といったパラメータ設定関数で設定します．描画パラメータは，コマンドラインから会話的に設定でき，パラメータの種類，順序を知らなくても効率的にグラフを作成できます．描画色 (0 から 7) は，数値以外に 表 5.2.「描画色」 のように数値に対応するカラー名を指定してもかまいません (全て大文字もしくは小文字で記述してください).

また，SATELLITE 起動時に描画パラメータは初期化され，デフォルト値が与えられます．

表 5.1. GPM コマンド

ウィンドウ関係	
wopen	ウィンドウを開く
wclose	ウィンドウを閉じる
we	画面を消去する
newpage	改ページする
chwin	描画先のウィンドウを設定する
グラフ関係	
graph	グラフを描く
axis	座標軸を描く
draw	指定したレベルのラインを描く
frame	グラフ枠を描く
label	文字列を表示する
グラフィックス描画関数	
line	線分，矩形を描く
2 次元データ描画関数	
cont	2 次元データを等高線で表示する
gsolm	2 次元データを鳥瞰図で表示する
map	2 次元データをマップ表示する
パラメータ設定関数	
color	グラフ，フレームの描画色を設定する
factor	描画するグラフィックスの倍率を設定する
font	文字のフォントを設定する
ltype	線の種類を設定する
lwidth	線の幅を設定する
origin	グラフの原点を設定する
scale	グラフの表示範囲を設定する
size	グラフの大きさを設定する
title	グラフ座標軸のタイトルを設定する
その他の関数	
ginit	描画パラメータを初期化する
gstat	現在の描画パラメータを確認する

表 5.2. 描画色

数値	カラー名
0	black
1	blue
2	red
3	mazenta
4	green
5	cyan
6	yellow
7	white

5.2. GPM の使用方法

GPM を使用する上で必要なこととして,

- ウィンドウの操作方法
- プリンタへの出力方法

が挙げられます．図形，グラフを描画するためには，ウィンドウを開く必要があり，"wopen" を用います．反対にウィンドウを閉じる関数は "wclose" です．ウィンドウは複数開くことが可能で，書き込み先のウィンドウを設定する関数は "chwin" です．ただし，複数ウィンドウに同時に書き込むことはできません.

さて，GPM コマンドにより作成したグラフは視覚的に確認するだけでなく，ファイルとして保存し，プリンタに出力できることが望まれます．ウィンドウ上のグラフをプリンタに出力する場合は,

```
% gpm2ps > filename
% lpr -Pps filename
```

としてください．"gpm2ps" は GPM が出力した中間ファイル (グラフィックデータ) を PS ファイル (グラフィックデータをプリンタが解釈できる形式) に変換するコマンドです．PS ファイル変換後は，Tgif などで読み込むことができます．"lpr" はプリンタにファイルの内容を送るコマンドです．作成したグラフを Tex などで eps ファイルとして利用したい場合は,

```
% gpm2eps > filename
```

としてください.

次に，簡単な例題により主な GPM コマンドの使用方法を説明します．詳しい使用方法についてはリファレンスマニュアルをご覧ください.

5.2.1. 1 次元データの表示

●例題 1 : 正弦波 (sin 関数) の表示

```
[]SATELLITE[]~/home/demo:[1]% wopen(1,"A4",0,1)
[]SATELLITE[]~/home/demo:[2]% t = (0~100) / 100
[]SATELLITE[]~/home/demo:[3]% y = sin(2 * PI * t)
[]SATELLITE[]~/home/demo:[4]% scale("N","A","N","A")
[]SATELLITE[]~/home/demo:[5]% graph(y,t,0,0,0,0,0)
[]SATELLITE[]~/home/demo:[6]% frame()
[]SATELLITE[]~/home/demo:[7]% axis(1,1,"XY","XY",5,0,0,0,0,0)
```

上述のリストは，直接コマンドラインより打ち込んでもよく，ファイルにしてインライン展開させても実行可能です．まず，グラフを描画するためには，ウィンドウを開く必要があります．1 行目の "wopen" がウィンド

ウを開く関数です．表示した図をプリンタに表示したい場合，またはファイルに保存したい場合は，最後の引数を 1 に設定します．プリンタに出力しない場合には 0 に設定してください (デフォルト値は 0 です).

注意 : 最後の引数を 2 に設定した場合は，ファイルのみの出力になります.

4 行目でグラフ軸の種類とスケールを決定しています．グラフ軸の種類は N (linear) と L (log) の 2 つがあり，X 軸，Y 軸ともに N を選択しています．引数の A (auto) はスケールの自動設定です．特に "scale" で設定しない場合は自動設定となります．スケールを固定してグラフ表示したい場合には，引数を F (fix) とし，関数の引数の 5，6 番目に X 軸の最小値，最大値もしくは Y 軸の最小値，最大値を記述するか，記述を省略した場合には値の入力を促しますので，直接，X 軸，Y 軸の最小，最大値を入力してください．5 行目で x と y の関係をグラフ表示，6 行目でグラフ枠，7 行目でグラフ軸を表示しています．この手順により作成したグラフを 図 5.1.「例題 1 の出力結果」 に示します．このグラフでは，特に色の設定を行っていないため，初期設定値の黒色で表示されます．色を変更する場合は，"graph" 関数実行前に "color" 関数を実行してください．"color" 関数では，描画色とフレーム色が指定できます.

なお，ウィンドウ画面を消去する関数は，"we" です.

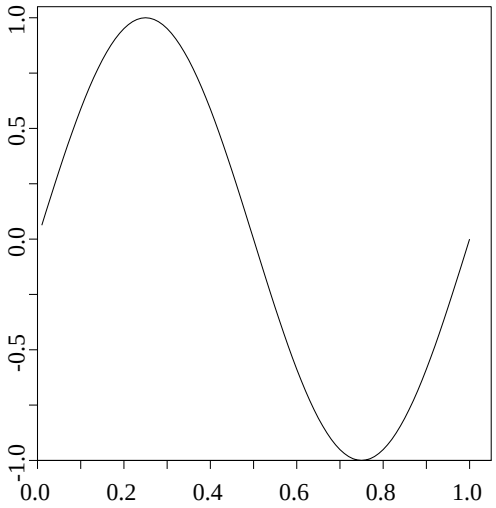


図 5.1. 例題 1 の出力結果

●例題 2 : 振幅，周波数が異なる 2 つの正弦波の表示

```
[]SATELLITE[]~/home/demo:[1]% wopen(1,"A4",0,1)
[]SATELLITE[]~/home/demo:[2]% sx = 80
[]SATELLITE[]~/home/demo:[3]% sy = 80
[]SATELLITE[]~/home/demo:[4]% origin(40,40)
[]SATELLITE[]~/home/demo:[5]% size(sx,sy)
[]SATELLITE[]~/home/demo:[6]% title(1,"time","f(t)")
[]SATELLITE[]~/home/demo:[7]% t = (0~100) / 100
[]SATELLITE[]~/home/demo:[8]% y1 = sin(PI * 5 * t)
[]SATELLITE[]~/home/demo:[9]% y2 = 0.5 * sin(2 * PI * 5 * t)
[]SATELLITE[]~/home/demo:[10]% scale("N","F","N","F",0,1.0,-1.2,1.2)
[]SATELLITE[]~/home/demo:[11]% lwidth(1,2)
[]SATELLITE[]~/home/demo:[12]% graph(y1,t,0,0,0,0,0)
[]SATELLITE[]~/home/demo:[13]% lwidth(2,2)
[]SATELLITE[]~/home/demo:[14]% graph(y2,t,0,0,0,0,0)
[]SATELLITE[]~/home/demo:[15]% axis(1,1,"XY","XY",5,0,0,0,0,0)
[]SATELLITE[]~/home/demo:[16]% lwidth(1,2)
[]SATELLITE[]~/home/demo:[17]% ltype(1,2)
[]SATELLITE[]~/home/demo:[18]% line(0,sy / 2,sx,sy / 2)
```

```
[]SATELLITE[]~/home/demo:[19]% ltype(1,1)
[]SATELLITE[]~/home/demo:[20]% frame()
```

例題 2 では，グラフの重ね合わせ，X 軸，Y 軸タイトルの表示，原点座標の指定，グラフサイズの設定，グラフ曲線の太さを指定しています．具体的には，4 行目で原点座標の指定，5 行目でグラフサイズの設定，6 行目でタイトル名を設定しています．ここでは，振幅の異なる波形を重ねるために，"scale" の設定を F (fix) としています．1 番目の波形を A (auto) で表示し，そこに 2 番目の波形を同じスケールで描画する場合は，D (default) として下さい．11，13 行目では線幅の設定を行っています．18 行目の "line" 関数の座標は相対座標で，基準位置は，"origin" で設定した場所です．"origin" には，ウィンドウの左下を原点とした絶対座標の値 (ウィンドウのマウスカーソルを移動すると表示される数値) を入力します．ここでは，17 行目の "ltype" により線種を破線とし，y = 0 の直線を引いています．例題 2 の出力結果を 図 5.2.「例題 2 の出力結果」 に示します．

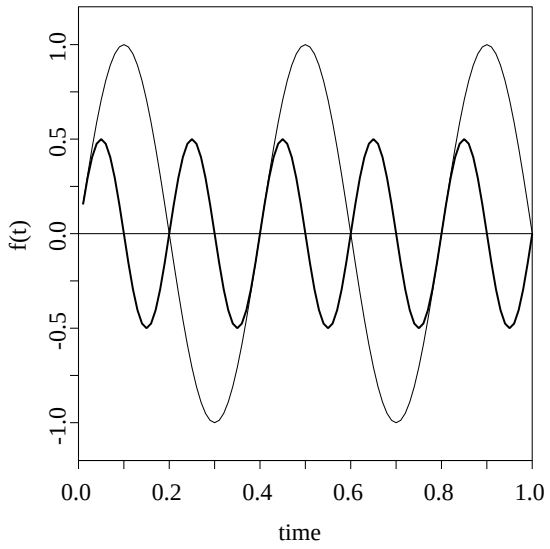


図 5.2. 例題 2 の出力結果

5.2.2. 2 次元データの描画

●例題 3 : 乱数データの 2 次元表示

```
[]SATELLITE[]~/home/demo:[1]% x = nrand(128,1,0,1)
[]SATELLITE[]~/home/demo:[2]% y = reform(x,(16,8))
[]SATELLITE[]~/home/demo:[3]% wopen(1,"A4",0,1)
[]SATELLITE[]~/home/demo:[4]% size(80,80)
[]SATELLITE[]~/home/demo:[5]% origin(20,200)
[]SATELLITE[]~/home/demo:[6]% gsolm(y,0.3,0.4,0,0,0,0,-1,1,1,"Y",1,1,5)
[]SATELLITE[]~/home/demo:[7]% origin(20,100)
[]SATELLITE[]~/home/demo:[8]% cont(y,0.3,"X",1,1)
[]SATELLITE[]~/home/demo:[9]% frame()
[]SATELLITE[]~/home/demo:[10]% origin(120,200)
[]SATELLITE[]~/home/demo:[11]% color("green","red")
[]SATELLITE[]~/home/demo:[12]% map(y,"X",1,0,1)
[]SATELLITE[]~/home/demo:[13]% color("black","black")
[]SATELLITE[]~/home/demo:[14]% frame()
[]SATELLITE[]~/home/demo:[15]% origin(120,100)
[]SATELLITE[]~/home/demo:[16]% map(y,"X",0,0,1)
[]SATELLITE[]~/home/demo:[17]% frame()
```

例題 3 では，1 次元の正規乱数データを 2 次元データに変換し，鳥瞰図表示，等高線表示，MAP 表示，カラー MAP 表示を行います．データを変換する関数は 4 行目の "reform" で 128 点の 1 次元データを 16×8 の 2 次元データとしています．6 行目では 2 次元データを鳥瞰図表示しています．この例では，グラフに隠線処理を行っ

ています．詳細はリファレンスマニュアルを参照して下さい．8 行目では等高線表示をしています．12，16 行目では，MAP 表示を行っています．"map" 関数には，データ値を最小値から最大値の間に正規化し，個々のデータ値をパターンのサイズに対応させ表示する方法と色に対応させる方法の 2 つがあります．例題 3 の出力結果を 図 5.3.「例題 3 の出力結果」 に示します．

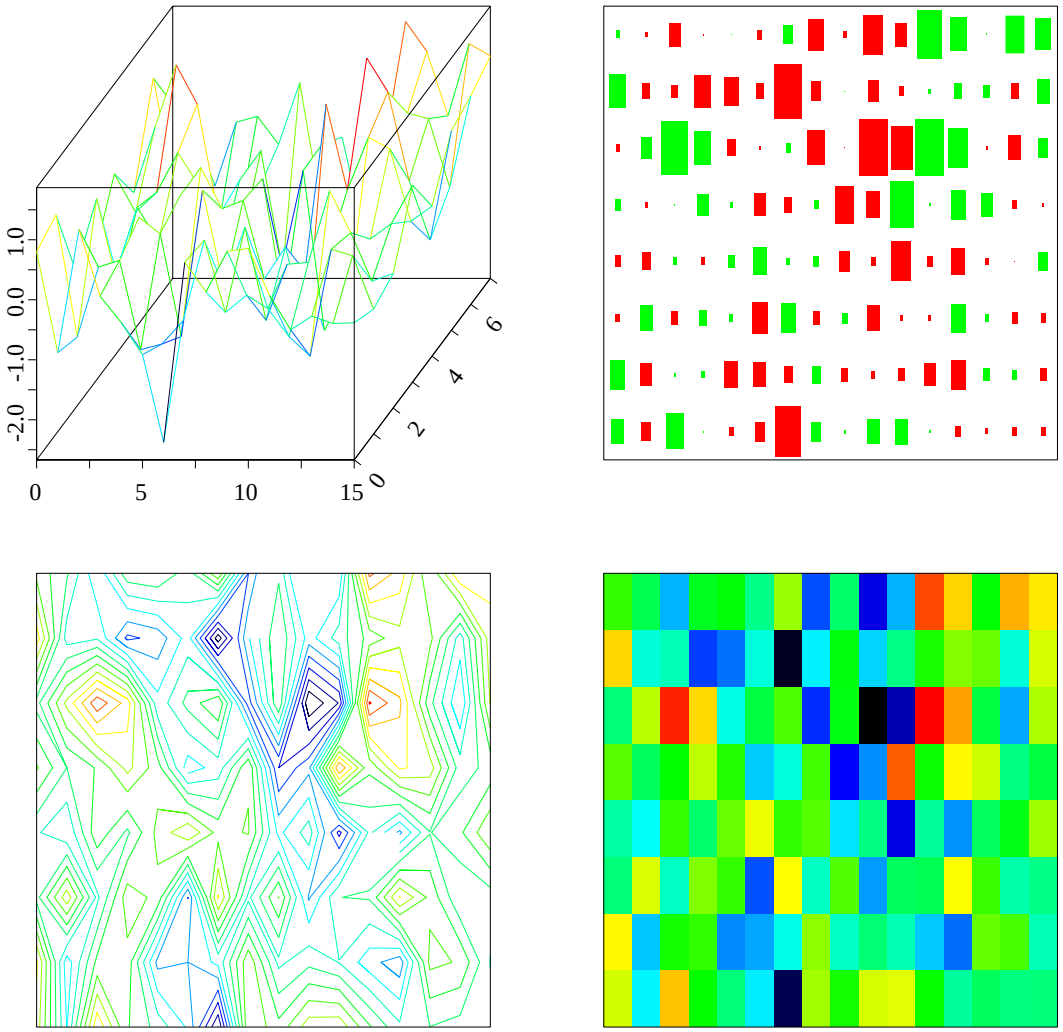


図 5.3. 例題 3 の出力結果

●例題 4 : 時系列データの表示

```
[]SATELLITE[]~/home/demo:[1]% wopen(1,"A4",0,1)
[]SATELLITE[]~/home/demo:[2]% sampling(1000*10)
[]SATELLITE[]~/home/demo:[3]% size(80,80)
[]SATELLITE[]~/home/demo:[4]% x = nrand(1000,1,0,1)
[]SATELLITE[]~/home/demo:[5]% origin(20,200)
[]SATELLITE[]~/home/demo:[6]% title(1,"time[msec]","value")
[]SATELLITE[]~/home/demo:[7]% scale("N","A","N","A")
[]SATELLITE[]~/home/demo:[8]% graph(x,"T",0,0,0,0,0)
[]SATELLITE[]~/home/demo:[9]% axis(1,1,"XY","XY",5,0,0,0,0,0)
[]SATELLITE[]~/home/demo:[10]% frame()
[]SATELLITE[]~/home/demo:[11]% label("I",20,70,5.0,0,"example4")
```

SATELLITE で，生体信号の解析等を行う場合，扱うデータとして時系列データが多く含まれます．そこで，例題 4 では，1000 点の正規乱数データを発生させ，それを 100[msec] 間に得られたデータと仮定し，時間軸表示させることを考えます．ここでは，1000 点のデータを時間軸 (100[msec]) と対応させるために，2 行目の "sampling" で 10 倍のサンプリングを行っています．サンプリング周波数の初期設定値は 1000 です．8 行目では X

軸の引数を T (time) としてグラフ表示しています．データ点をそのまま表示したい場合は D (data) としてください．11 行目では，"label" 関数によって文字列の表示を行っています．表示座標の指定は，"line" 関数と同様に相対座標です．また，"label" 関数によって表示される文字種は "font" で指定します．

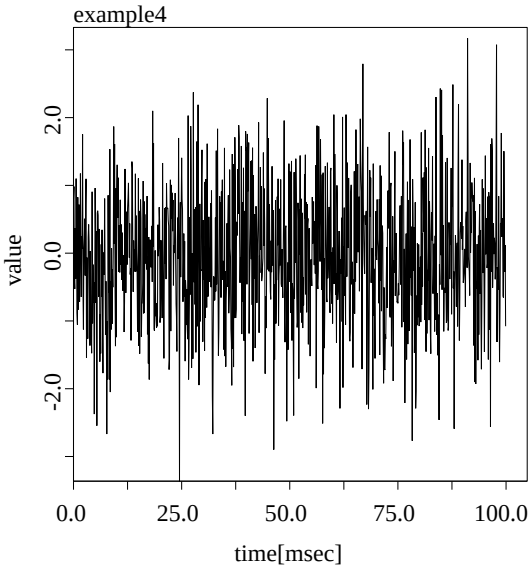


図 5.4. 例題 4 の出力結果

第 6 章 バックプロパゲーション・シミュレーター

6.1. BPS とは

BPS (**B**ack-**P**ropagation **S**imulator) は，多層階層型ニューラルネットワークをワークステーション上で実現するための関数・手続きの集合であり，SATELLITE のシステムモジュールの 1 つです．サポートしているニューラルネットワークモデルは，**多層階層型フィードフォワードニューラルネットワーク** (以下，**ニューラルネットワーク**または単に**ネットワーク**) で，その学習アルゴリズムとして**バックプロパゲーション** (**誤差逆伝播学習法**), およびその加速化法として提案されている 5 つの学習法を用いることができます．

BPS モジュールの使用にあたっては，ニューラルネットワークおよび学習に関する最小限の予備知識が必要とされています．**BPS** をはじめてお使いになる方には，例えば Rumelhart らの Parallel Distributed Processing 第 8 章などを参照し，ニューラルネットワークとその学習メカニズムについて知っておくことをお勧めします．

6.2. BPS の特徴

BPS は，以下のような特徴をもっています．

- SATELLITE の提供する会話的環境による会話形式で，ニューラルネットワークの構造を容易に定義することができます．また，ネットワークにおけるパラメータ (結合重み) の設定・変更も簡単にでき，ニューラルネットワークに関するシミュレーションをスムーズに実行できます．
- SATELLITE の inline 関数を使用することにより，ネットワーク構造およびパラメータの設定から学習，ネットワークのテストまたはその内部状態のトレースまでを自動的に一括処理することができます．
- SATELLITE におけるバッファモニタリング機能を利用し，学習時の誤差変化の様子をリアルタイムで見ることができます．また，**GPM** モジュールを用いることにより，シミュレーション結果の視覚的表示が簡単にできます．
- SATELLITE のディジタル信号処理モジュール **ISPP** を使用することにより，ニューラルネットワークの多角的かつ詳細な解析を行なうことができます．

6.3. BPS で使用するファイルについて

BPS では，ネットワークの学習およびテスト等におけるデータの受け渡しはファイルを介して行なわれます．こうしたファイルには，

1. 学習時にネットワークに提示する入力データを格納する**入力データファイル**
2. 学習時の入力データに対応する教師出力データを格納する**教師データファイル**
3. 結合重みの初期値を格納する**初期ウェイトファイル**
4. 学習時に更新されるネットワークの結合重みを格納する**ウェイトヒストリファイル**
5. 学習時の総和誤差を格納する**エラーヒストリファイル**
6. テスト時にネットワークに提示する入力データを格納する**テストデータファイル**
7. テスト時の入力テストデータに対するネットワークの出力を格納する**テスト結果ファイル**

の 7 種類があります．これらのフォーマットは，すべて SATELLITE のファイルフォーマットに準拠していますが，データの格納形式はそれぞれ異なっています．格納形式については，順次説明します．その他，ネットワーク構造や内部状態の設定，データファイル管理などに関するパラメータを保存しておくための**パラメータファイル** (テキストファイル) があります．

表 6.1.「BPS で使用するファイル」 に，各々のファイルの内容と，そのファイルを使用する関数およびそのファイルを出力する関数を示します．

表 6.1. BPS で使用するファイル

ファイルの種類	ファイルの内容	ファイル使用関数
入力データファイル	学習時の入力データ	bteach, bsetrec
教師データファイル	学習時の教師データ	bteach
初期ウェイトファイル	結合重みの初期値	bwalgo, bwinit, bweight
ウェイトヒストリファイル	学習時に更新される結合重み値	bweight, bsetrec, blearn, bwgtload, berrfunc, brvmap, bsigmoid
エラーヒストリファイル	学習時の総和誤差	berror, blearn, berrload
テストデータファイル	テスト時の入力データ	bsetrec
テスト結果ファイル	テスト時のネットワーク出力	bsetrec, brec, bactload

6.4. 使用法

ここでは，ニューラルネットワークに関するシミュレーションの具体的な例を用いて **BPS** の使用法を解説します．ここで取り上げる例は XOR (Exclusive Or) 問題で，表 6.2.「XOR」 に示すような 2 の入力と 1 つの出力関係を満たす簡単な論理式をニューラルネットワークにより実現させるという問題です．

表 6.2. XOR

入力		出力
0	0	0
0	1	1
1	0	1
1	1	0

6.4.1. 入力・教師データファイル，テストデータファイルの作成

ネットワークを学習させるためには，あらかじめ**入力・教師データファイル** を作成しておく必要があります．入力・教師データファイル，テストデータファイルの格納形式は，図 6.1.「入力・教師データファイル，テストデータファイル，テスト結果ファイルの格納形式」 に示すように，レコード方向にパターン，データポイント方向に入力 (出力) ユニットを対応させます．brec 関数実行時に生成される**テスト結果ファイル** も，この格納形式をとります．パターン数，入力・出力ユニット数には制限がありません．

XOR 問題の場合は，表 6.2.「XOR」 より，入力ユニット数および出力ユニット数がそれぞれ 2 および 1 であり，パターン数は 4 です．これらのデータは series 型あるいは snapshot 型として与えることができます．いま，入力データを in，教師データを out の変数名で表すと，それぞれの変数の宣言および値は以下のようになります．

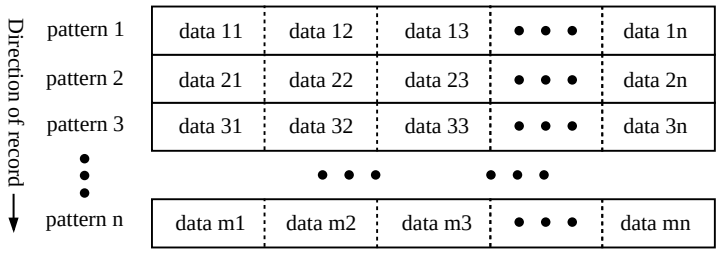


図 6.1. 入力・教師データファイル，テストデータファイル，テスト結果ファイルの格納形式

```
series 型                 snapshot 型

series in[2], out[1];     snapshot in[4][2], out[4];
#                         #
in:[0] = (0,0);           in[0][0] = 0; in[0][1] = 0;
in:[1] = (0,1);           in[1][0] = 0; in[1][1] = 1;
in:[2] = (1,0);           in[2][0] = 1; in[2][1] = 0;
in:[3] = (1,1);           in[3][0] = 1; in[3][1] = 1;
#                         #
out:[0] = 0;              out[0] = 0;
out:[1] = 1;              out[1] = 1;
out:[2] = 1;              out[2] = 1;
out:[3] = 0;              out[3] = 0;
```

これらの入力・教師データを，次のようにしてファイルへ書き込みます.

```
$"in.dat" = in;
$"out.dat" = out;
```

以上より，XOR 問題の場合，in.dat および out.dat がそれぞれ入力データファイルおよび教師データファイルとなります.

また，ネットワークのテスト時には，**テストデータファイル** が必要となります．ただし，学習時の入力データに対するネットワーク出力を見たい場合は，テストデータファイルとして入力データファイルを用います.

6.4.2. パラメータの設定

学習，ネットワークのテスト等を実行するためには，各種パラメータを設定しておく必要があります．パラメータは大きく分けて以下の 4 つに分けることができます.

1. ネットワーク構造パラメータ
2. 結合重み初期値生成パラメータ
3. 学習パラメータ
4. テストパラメータ

設定したパラメータは，bpsave 関数により**パラメータファイル** (表 6.3.「パラメータファイル」) に保存しておくことができます．これにより，次に同じパラメータでシミュレーションを行う場合には，bpload 関数によりそのファイルからパラメータを読み込むことでパラメータを設定することができます．パラメータファイルはテキストファイルですので，エディタを使用して，パラメータを変更することで，異なる条件の下でのシミュレーションが容易に行なえます.

表 6.3. パラメータファイル

コメント	内容
number of layer	層数
number of each layer's cell	各層のユニット数
status of char.func and bias	各層の特性関数とバイアス項の有無
init algorithm	結合重み初期値生成アルゴリズム
init data file name made by init command	初期ウエイトファイル名
seed	乱数の種
maximum of init data	結合重み初期値最大値
minimum of init data	結合重み初期値最小値
init weight file for learn command	学習用の初期ウエイトファイル名
weight history file name	ウェイトヒストリファイル名
weight store interval	ウェイトストアインターバル
weight history store mode	ウェイトストアモード
error history file name	エラーヒストリファイル名
error store interval	エラーストアインターバル
store direction for error	エラーストアディレクション
store mode for error	エラーストアモード
input data file name for learning	学習用入力データファイル名
teach data file name for learning	学習用教師データファイル名
first pattern No. for learning	学習用データ開始パターン番号
last pattern No.	学習用データ最終パターン番号
learning mode	学習モード
learning algorithm	学習アルゴリズム
learning rate	学習率
momentum	慣性率
increasing factor for learn. rate	Vogl 法用増加係数
reduction factor for learn. rate	Vogl 法用減少係数
threshold for Vogl method	Vogl 法用閾値
factor for Ochiai's method	落合法用ファクター
min. error for end	学習終了判定用最小誤差
max. learn count for end	学習終了判定用最大誤差
display interval (console)	ディスプレイ表示間隔
comment	コメント
weight file name for test	テスト用ウェイトヒストリファイル名
weight history number	テスト用ウェイトヒストリ番号
test data file name	テストデータファイル名
first pattern No. for test	テストデータ開始番号
last pattern No. for test	テストデータ最終番号
input layer No.	テストデータ入力層番号
output layer No.	テストデータ出力層番号
result file name	テスト結果ファイル名

ネットワーク構造パラメータ

ネットワーク構造パラメータは，結合重み初期値生成，学習，ネットワークテスト，トレース等全ての場合において設定しておく必要があります．ネットワーク構造パラメータは，**層数**，**各層のユニット数** と**特性関数** です．これらのパラメータは，blayer，bfunction 関数により設定します．なお，特性関数については，入力層 (第 **0** 層) は必ず**線形関数** となります．また，bfunction 関数は，blayer 関数で層数を設定した後に実行して下さい．

XOR 問題の場合では，ネットワークの構造は 2 入力 1 出力となります．XOR 問題では，中間層を付加し，中間層にシグモイド関数を特性関数とする 2 個のユニットを配置する必要があります (文献 [1] 参照)．また，ここでは，出力数の特性関数を線形関数とし，中間層，出力層の各ユニットにしきい値 (バイアス) を設定します．したがって，以上のネットワーク構造パラメータの設定は，次のようになります (リファレンスマニュアル参照)．

```
blayer(3,2,2,1);
bfunction("LN","SA","LA");
```

結合重み初期値生成パラメータ

学習時の結合重み初期値を格納した初期ウェイトファイルは，bwinit 関数により生成されます．bwinit 関数を実行するために設定しておかなければならないパラメータが，結合重み初期値生成パラメータです．このパラメータには，**初期値 (乱数) 生成アルゴリズム**，**初期ウェイトファイル名** (格納形式については後に説明します)，**乱数の種**，**初期値の最大値**，および，**最小値** があります．初期値生成アルゴリズムには，与えられた種から生成される乱数をそのまま用いる方法 (**R**ANDOM) と，本研究室で考案されたアルゴリズム (**J**IA) の 2 つが選択できます．(JIA のアルゴリズムを用いる場合には，各層にバイアス項を必ず付加して下さい.) これらのパラメータは，bwalgo 関数により設定します．

XOR 問題の場合は，例えば，以下のように結合重み初期値生成パラメータを設定します．

```
bwalgo("R","initwf",1,1.0,-1.0);
bwinit();
```

学習パラメータ

学習の条件を決めるパラメータは，bweight，berror，bteach，blalgo，blend，bdisp 関数により設定します．

bweight 関数では，**初期ウェイトファイル名** (学習時には，このファイルから結合重み初期値が読み込まれる)，**ウェイトヒストリファイル名** (学習時に更新される結合重みの履歴が格納されるファイル)，**ウェイト・ストアインターバル** (履歴の格納間隔)，**ウェイト・ストアモード** (格納方法 : **アペンド** と **オーバーライト** の 2 種類から選択する) を設定します．この場合の初期ウェイトファイルには，新たに生成したファイルではなく，前回のウェイトヒストリファイルを設定しても結構です．その場合には，ヒストリファイルに書き込まれている最後の履歴を初期値として用います．ストアモードは，ウェイトの履歴が必要なければオーバーライトにして下さい．なお，ウェイトヒストリファイルの格納形式，ストアモード等については，後に説明します．

berror 関数では，**エラーヒストリファイル名** (学習時の総和誤差の履歴を格納するファイル)，**エラー・ストアインターバル**，**エラー・ストアディレクション** (格納方向 : **レコード方向** と**データポイント方向** の 2 種類から選択する)，**エラー・ストアモード** を設定します．エラーストアディレクションをレコード方向とすると，各出力ユニットの誤差と総和誤差が出力されますが，データポイント方向とすると，総和誤差のみを出力します．エラーヒストリファイル，ストアディレクション等については，後に説明します．

bteach 関数では，**入力データファイル名**，**教師データファイル名**，**使用パターン番号** (入力・教師データの何番のパターンから何番のパターンまでを使用するか) を設定します．使用パターン番号は，0 から 0 番目までと設定すると，全パターンを使用することになります．

blalgo 関数では，**学習モード** (**逐次**，**一括学習の選択**)，**学習アルゴリズム** (Steep，Momentum 等合計 6 つの学習方法から選択できる)，**学習率** および各学習法において必要なパラメータを設定します．

blend 関数では，学習終了条件である**学習回数**，**最小誤差** を設定します．

bdisp 関数では，**学習回数，誤差値等をディスプレイに表示する間隔** および**コメント** を設定します．

XOR 問題での学習パラメータは，例えば以下のように設定します．

```
bweight("initwf","wgt",200,"A");
berror("err",200,"D");
bteach("in","out",0,0);
blalgo("S",6,0.005,0.6,0.0003,0.75,0.6);
blend(0.0,5000);
bdisp(200,"I'M LEARNING!");
```

テストパラメータ

ネットワークのテストを行うためのパラメータには，**テスト用ウェイトファイル名** (学習後のネットワークの結合重みを格納したウェイトヒストリファイル)，**ウェイトヒストリ番号**，**テストデータファイル名**，**使用するパターン番号**，**データの入力層,出力層，テスト結果ファイル名** があります．使用するパターン番号については，bteach 関数と同様に，0 から 0 番目までと設定すると，全パターンを使用することになります．これらのパラメータは，bsetrec により設定します．

例えば，XOR 問題の場合，

```
bsetrec("wgt",0,"in",0,0,0,2,"res");
```

により，ネットワークのテストにおけるパラメータは設定されます．

6.4.3. 結合重み初期値生成

学習を行うには，初期ウェイトファイルが必要ですが，前回の学習時に生成されたウェイトヒストリファイルに格納されている履歴を使用して学習を始める場合には，bwinit 関数を実行する必要はありません．

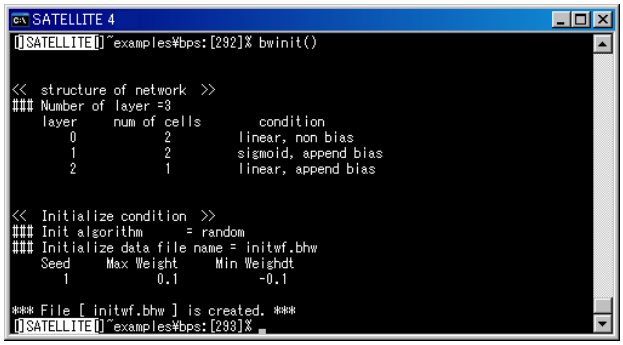


図 6.2. bwinit 関数実行の様子

図 6.2.「bwinit 関数実行の様子」 に bwinit 関数を実行した様子を示します．実行時には，まず，ネットワーク構造と結合重み初期値生成パラメータが表示されます．次に，設定した初期ウェイトファイルが存在しなっかった場合には，新たにファイルを作成した旨のメッセージが表示されます．ファイルが存在した場合には，次のようなメッセージが表示され，オーバーライトしても良いか聞いてきます．

```
*** File [ filename ] is already exist.  ***
    Overwrite ?  ( y/n ) :
```

この時に y を入力すれば，オーバーライトした旨のメッセージが表示されます．

```
File [ filename ] is overwrited.
```

n を入力すれば，エラーメッセージを表示して関数が終了し，ファイル名を設定し直すことになります．

6.4.4. 学習

学習は blearn 関数を実行することにより行われます．blearn 関数は，入力・教師データ，ネットワークの結合重み初期値をファイルから読み込み，学習時に更新される結合重みの履歴，誤差は設定した学習パラメータに従いファイルに書き込まれます．

また，総和誤差を変数 (バッファ) に書き込むこともできます．したがって，バッファモニタリング機能を利用し，誤差をリアルタイムにグラフ表示することができます．この場合には，series 型変数 (あるいは snapshot 型変数) x をバッファモニター関数 mon および blearn 関数に与えることが必要です．すなわち，

```
series x;    または snapshot x[lc] (lc=学習回数)
mon(x);
blearn(x);
```

とすればよいわけです．総和誤差を変数に書き込まない場合には，blearn(0) とします．

学習時に更新される結合重みの履歴はウェイトヒストリファイルへ，出力誤差はエラーヒストリファイルへ格納されます．ウェイトヒストリファイルの格納形式には，格納順にデータを追加していくアペンドモードと，前回のデータの上に上書きするオーバーライトモードがあります．これらの格納形式を説明するために，図 6.3.「ネットワーク構造例」 のネットワークを例として用います．

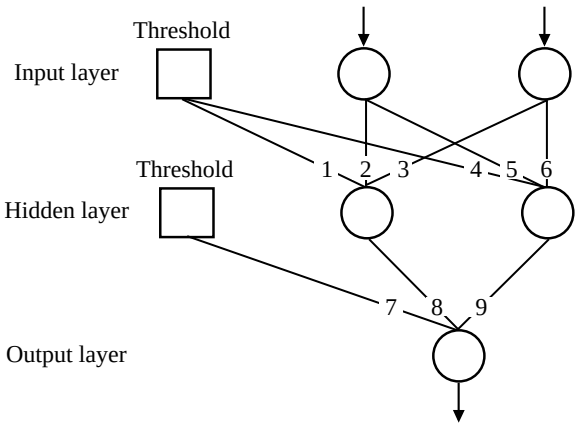


図 6.3. ネットワーク構造例

このネットワークは，3 層でユニット数は入力層から 2--2--1 であり，中間層と出力層にバイアス項が付加されています．ここで，図中の円はユニットを，正方形はバイアスユニット，また各結合線の上に書かれている数字は結合のファイルに格納される順番を示しています．このネットワークの結合重みをファイルに格納すると 図 6.4.「結合重み格納フォーマット」 のようになります．

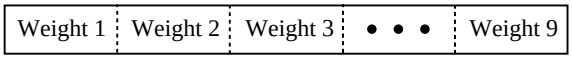


図 6.4. 結合重み格納フォーマット

学習の中断 / 再開機能を実現するため，結合重みを格納する毎に，結合重みのデータの後ろに前回の学習の結合重み補正量のデータを格納します (図 6.5.「ウェイトヒストリファイルフォーマット」). ここで，初期値データを格納する初期ウェイトファイルにおいては，補正量に相当する部分をすべて 0.0 としています．以下では，1 つの結合重み，あるいは補正量のデータの集まりをデータブロックと呼ぶことにします．図 6.5.「ウェイトヒストリファイルフォーマット」 では，レコード n からの m レコード，あるいはレコード n+m+1 からの m レコードが 1 データ 1 ブロックとなります．bwinit 関数は，結合重みの初期値と全てが 0.0 の 2 データブロックを生成し，blearn 関数は，初期ウェイトファイルの最後の 2 データブロックを読み込んで処理を行います．これにより，学習を中断した後あるいは学習終了後に，その時点から継続して学習を行うことが可能となります．

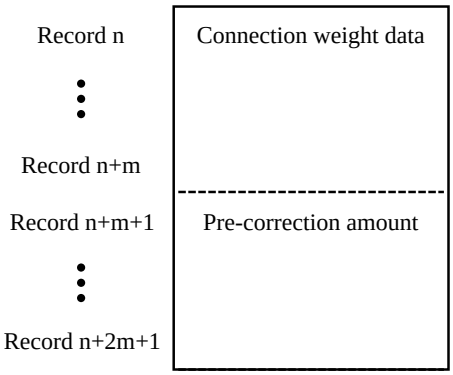


図 6.5. ウェイトヒストリファイルフォーマット

次に，アペンドとオーバーライトの 2 つの格納モードについて解説します．アペンドモードは，前回のヒストリから 1 データブロック後ろに，結合重みと前回補正量を書き込むモードです．この概念図を 図 6.6.「アペンドモード概念図」 に示します．このような格納方法をとることにより，最新のデータをファイルの最後に置くことができ，また学習開始から現時点までの履歴を格納することが可能となります．オーバーライトモードは，アペンドモードのように前回のヒストリデータから 1 データブロック後ろに格納するのではなく，ネットワークの初期化に用いたデータの上に重ね書きするモードです．これは，ネットワークの規模が大きく結合数が多い場合，あるいは結合重みの履歴を見る必要がない場合などに使用します．この概念図を 図 6.7.「オーバーライトモード概念図」 に示します．

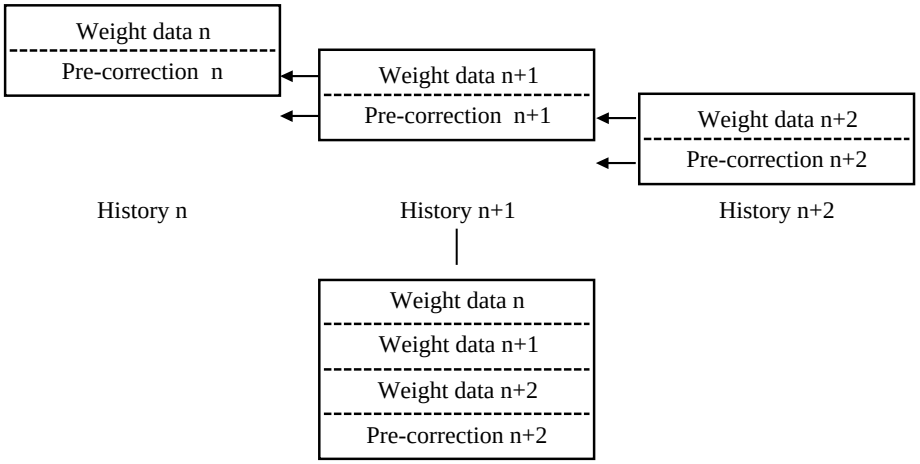


図 6.6. アペンドモード概念図

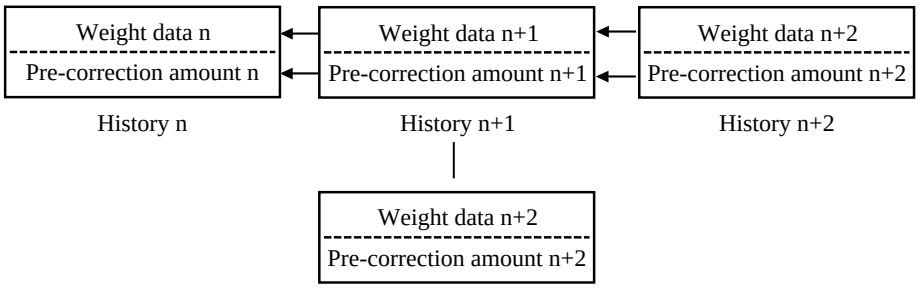


図 6.7. オーバーライトモード概念図

エラーヒストリファイルの格納形式は，レコード方向とデータポイント方向に格納する 2 種類の格納方法が用意されています．また，ウェイトヒストリファイルと同様に格納モードとしてアペンドとオーバーライトが用意されています．

レコード方向への格納は，出力層の各ユニットの誤差とそれらの総和をデータポイント方向に格納する方法です．図 6.8.「レコード方向格納フォーマット」 にその概念図を示します．図のように，出力層各ユニット毎の誤差に総和誤差を加えて格納するため，1 つのヒストリのデータ長 (data_num) は，出力層のユニット数 +1 となります．

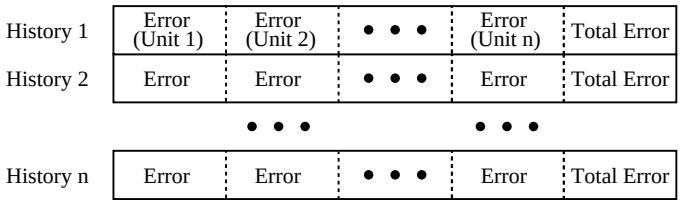


図 6.8. レコード方向格納フォーマット

図 6.9.「blearn 関数実行の様子」 に blearn 関数を実行した様子を示します．

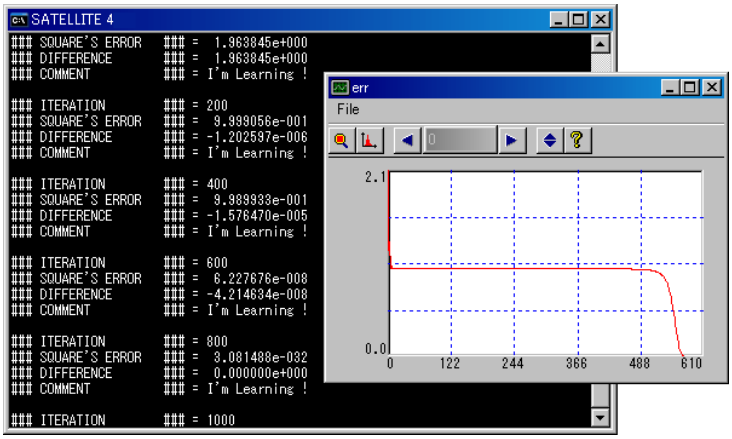


図 6.9. blearn 関数実行の様子

実行時には，まず，ネットワーク構造と学習パラメータが表示されます．次に，設定したウェイトヒストリファイル，エラーヒストリファイルが存在しなっかった場合には，新たにファイルを作成した旨のメッセージが表示されます．どちらか一方のファイルでも存在した場合には，それを伝えるメッセージが表示され，設定したストアモードに従ってオーバーライトまたはアペンドしても良いか聞いてきます．ストアモードが，オーバーライトの場合には y を入力すれば，オーバーライトしますというメッセージが表示されますが，n を入力した場合には，次のメッセージを表示して関数が終了し，ファイル名を設定しなおすことになります．ストアモードがアペンドの場合には，y を入力すればアペンドされます (メッセージは表示されない) が，n を入力した場合にはメッセージを表示し，オーバーライトされることになります．

さらに，学習が開始されると学習回数と総和誤差値，前回の誤差との差，コメントが表示されます．ここで，表示される総和誤差値は以下のように計算されています．

全パターン数を M とし，出力ユニット数を N 個，教師データを t_j，出力を o_j とすると，あるパターンを入力した時の二乗誤差 e_i は，

$$e_i = \sum_{j=1}^{N} (t_i - o_i)^2$$

です．全パターンを入力したときの総和誤差 E は，

$$E = \sum_{i=1}^{N} e_i^2$$

となります．学習は，0 回目から始まりますが，0 回目の誤差は，結合重みを全く更新しない初期状態での，ネットワーク出力と教師データとの誤差です．

学習は，総和誤差値が設定した最小誤差以下となった場合，もしくは設定した学習回数に到達した場合に終了します．終了時には，次のようなメッセージを表示します．

```
*** Learning is done !  ***
```

6.4.5. ネットワークのテスト

ネットワークのテストは，brec 関数を実行することにより行われます．brec 関数は，テストデータ，結合重みをファイルから読み込み，出力結果をファイルに書き込みます．同時に，ユニットを正方形で表し，ユニット間を線で結ぶことによりネットワーク構造を表示できます．また，ユニットの活性値を正方形の大きさの変化，または色の変化で表すことができます．この場合は，あらかじめ wopen 関数を使用し，ウィンドウを開いておくことが必要です．ネットワーク構造は，**GPM** モジュールを用いて大きさ，位置等を調節することができます．size でネットワークの大きさを，origin で位置を，color で線の色を設定することが可能です．

図 6.10.「brec 関数実行の様子」 に，brec 関数を実行している様子を示します．

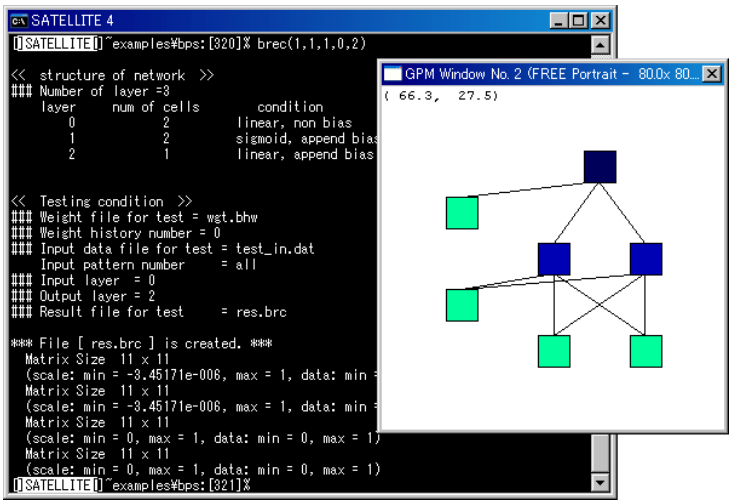


図 6.10. brec 関数実行の様子

実行時には，まず，ネットワーク構造とテストパラメータが表示されます．次に，設定したテスト結果ファイルが存在しなっかった場合には，新たにファイルを作成した旨のメッセージが表示されます．ファイルが存在した場合には，それを伝えるメッセージが表示され，オーバーライトしても良いか聞いてきます．この時に y を入力すれば，オーバーライトしましたというメッセージが表示されますが，n を入力すれば関数が終了し，ファイル名を設定しなおすことになります．

6.4.6. 誤差値，結合重み値等のトレース

BPS のロード関数と SATELLITE の **GPM** モジュールを組み合わせることにより，ネットワーク内部パラメータのトレースを実現します．具体的には，以下のような手順により行います．まず，学習時に生成されたウェイトヒストリファイル，エラーヒストリファイル，およびネットワークテスト時に生成されたテスト結果ファイルから，以下のロード関数を用い，トレースに必要なデータを，SATELLITE の変数 (バッファ) にロードします．

berrload : エラーヒストリファイルからデータをロードする
bwgtload : ウェイトヒストリファイルからデータをロードする
bactload : テスト結果ファイルからデータをロードする

さらに，グラッフィックウィンドウを使用するため，wopen 関数を実行し，バッファのデータに対して，cont, graph，gsolm，map 等の **GPM** モジュールを用いて表示を行うことができます.

6.4.7. ネットワーク内部状態の解析

ネットワーク内部状態の解析は，SATELLITE の **ISPP** 関数を用いても行うことができますが，**BPS** 専用の関数として，brvmap，bsigmoid，berrfunc，bcor 関数が用意されています.

- brvmap

  結合荷重逆投影演算を行い，その結果を SATELLITE の 2 次元バッファへ格納します．そのデータに対しては，**GPM** モジュールを用いて表示を行うことが可能です.

- bsigmoid

  この関数は，まず，テストパラメータに従ってテストを行います．次に，特性関数にシグモイド関数を使用しているある任意のユニットに関して，横軸にユニットの入力総和 (ネット値) を，縦軸に活性値をとりシグモイド曲線を描きます．さらに，その曲線上にデータを入力した時の活性値をプロットします．また，入力データ数が多い場合には，ネット値をヒストグラムとして描くこともできます.

- berrfunc

  ある任意のユニット間の結合重みを 2 カ所選択し，他の重みの値は固定しておき，選択された場所の重みの値を少しずつ変化させていきます．そのときに，ネットワークに順次データを入力し，出力の二乗誤差を計算します．その値は，SATELLITE のバッファへ格納するため，**GPM** モジュールを用いて表示することが可能となります.

- bcor

  SATELLITE のバッファデータに対し，相関マトリックスを求める関数です．ウェイトヒストリファイルより，bwgtload 関数を使用し，データをバッファへロードしておき，本関数を実行すれば，結合重みの相関値を求めることが可能です.

第 7 章 神経回路シミュレータ

7.1. NCS の概要

近年，計算機技術は，計算速度の向上や周辺機器の充実など飛躍的な発展を続けています．これに伴い，脳・神経系における情報処理機構に関する新しい研究アプローチとして，生理実験で得られた結果を元に細胞の特性を数理的に記述，再構成し，シミュレーション解析により，その機能メカニズムを探る，再構成的アプローチが注目されています．こうした数理モデルは，通常，多元の非線形連立微分方程式として記述され，数値計算によって解が求められます．しかしながら，このようなシミュレーションプログラムを，汎用プログラミング言語を用いて記述する場合，モデルパラメータの変更に伴う再プログラミングや，ファイル入出力など，モデル記述とは別な非本質的な作業が必要になり研究の効率を著しく低下させることが指摘されています．

そこで，こうした研究を支援するソフトウェアシステムとして神経回路シミュレータ NCS (Neural Circuit Simulator) が開発されました．NCS では細胞の特性や結合状態を専用のモデル記述言語である NCS 言語により記述します．NCS 言語のモデル記述には数値積分，ファイル入出力，計算結果の保存などの非本質的な処理は含まれておらず，研究者はモデル構築作業に専念することができます．また，モデルパラメータ，外部入力刺激などのシミュレーション条件は，シミュレーションプログラムとは別のファイルに格納されており，モデルを書き換えることなく，様々な条件下でのシミュレーションを効率的に行うことができます．このように NCS は神経回路モデルのシミュレーションにおいて，多くのプログラミングの知識を必要とせず，条件を変更しながらシミュレーションを効率良く行えるように設計されたシステムです．また，モデル記述の仕様等は汎用性を考え設計されていますので，神経回路モデルに限らず，連続系モデルであれば NCS を用いてモデリングシミュレーションが可能です．

以上のような NCS の特徴をまとめると，次のようになります．

- 生理学的知見に基づいた大規模神経回路モデルシミュレーションが可能．
- 生理学的知見に忠実なモデルはもちろん，連続系モデル一般を全て同様に扱うことができる．
- 数理モデル構築以外のプログラミングが不要．
- 再コンパイルをすることなく，条件を変更しながらのシミュレーションが可能．

7.1.1. 基本仕様

図 7.1.「NCS のシステム構成」に NCS システム構成を示します．NCS は 3 つの構成要素，すなわち NCS プリプロセッサ，NCS ライブラリ，条件設定関数群から構成されています．NCS 言語で記述された神経回路モデルは，NCS プリプロセッサを通して C 言語のシミュレーションプログラムとシミュレーション条件ファイル群に変換されます．変換された C 言語のシミュレーションプログラムは，C コンパイラによりコンパイルされたのち，NCS ライブラリとリンクされ，シミュレーション実行ファイルになります．NCS ライブラリは，シミュレーション開始前の処理，数値積分ルーチンなどを含む，シミュレーションを実現するための基本プログラムの集合体です．シミュレーション条件ファイル群は，外部入力条件，モデルパラメータ，信号遅延情報などシミュレーションの実行に関する情報を持ったファイルの集まりであり，条件設定関数群により，各種の条件に設定することができます．これらの条件ファイル群と実行ファイルによりシミュレーションが行われます．

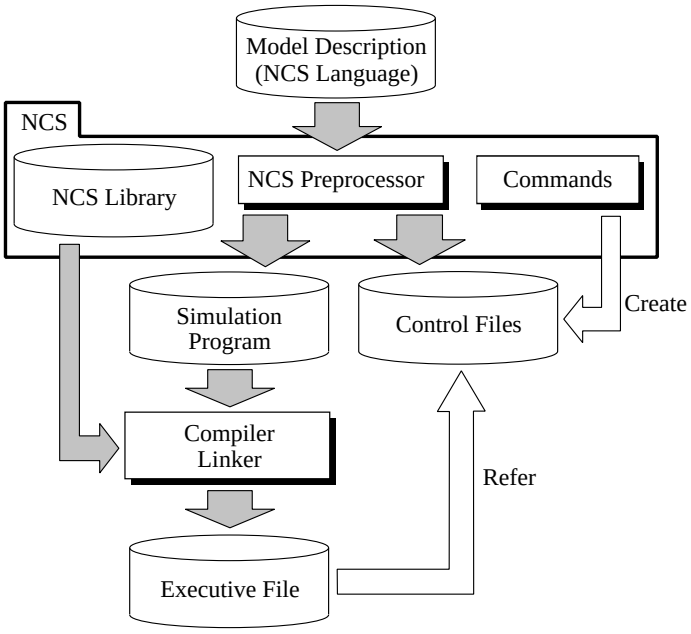


図 7.1. NCS のシステム構成

7.1.2. NCS と SATELLITE

NCS は当初 MS-DOS 上で開発され利用されてきましたが，UNIX ワークステーション上で構築された SATELLITE のモジュールの一つとして機能するように変更が加えられると共に機能の追加，改良がなされてきました．すなわち，NCS と SATELLITE の統合により，モデリングからシミュレーション結果の解析，表示，評価に至る一連の作業を円滑に進める環境が実現されました．現在 NCS は，SATELLITE の理念をさらに推し進めた SATELLITE 言語上で稼動しており，一層の利便が図られています．

7.2. NCS 言語

NCS では，モデルを記述するために，NCS 言語と呼ばれる専用の言語を用いています．NCS にはより効率よくモデル記述を行うため，いくつかの特殊記述が用意されています．

NCS 言語は，予約語，NCS ライブラリ関数，文および特殊記述で構成されています．例えば，図 7.2.「RL 直列回路」 に示す RL 直列回路は 図 7.2.「RL 直列回路」 に示す微分方程式で記述されますが，この回路の NCS による記述は，リスト 1 のようになります．

NCS では，リスト 1 に示すように "module" 文から始まり "end" で終わる，あるシステムに関する一連のモデル記述を 1 つの単位とします．NCS は通常シングルモジュールからなるモデルのみ扱いますが，その拡張として複数のモジュールからなるモデルを扱うことも可能です．このとき，モジュール記述の他に，モジュール同士の結合状態を記述するネットワーク記述が必要になります．ネットワーク記述は，NCS が神経回路を対象にしていることから，神経回路を構成しやすいよう特化した記述になっています．ネットワーク記述をしようした複数のモジュールからなるモデルを NCS 言語で記述する方法などは 項 7.4.2.「NCS 言語による結合モデル」 で詳しく説明します．

以下には NCS 言語で使用される，予約語，NCS ライブラリ関数，文，特殊記述の詳細について述べていきます．

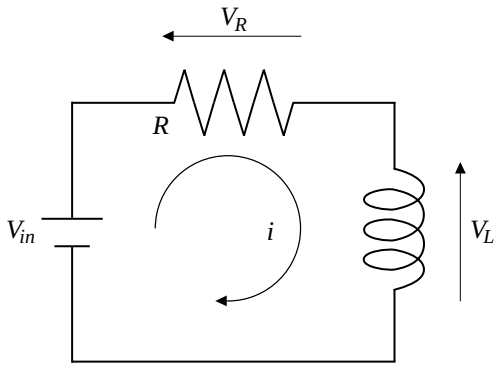


図 7.2. RL 直列回路

表 7.1. RL 直列回路モデル

電流 $\frac{di}{dt} = \frac{e - v_r}{l}$	i : 電流 [A] e : 電圧源 [V] l : インダクタンス 0.1[H]
インダクタンス電圧 $v_l = \frac{di}{dt} \cdot l$	v_l : インダクタンス電圧 [V]
抵抗電圧 $v_r = r \cdot i$	v_r : インダクタンス電圧 [V] r : 抵抗 10[Ω]

リスト 1 : NCS 言語による RL 直列回路モデルの記述

```
1   /* RL circuit module */
2   module:      circuit;
3   exinput:     E;
4   output:      i;
5   observable:  Vr, Vl;
6   constant:    L = 0.1;
7   parameter:   R = 10., i0 = 0.;
8   function:
9                di = (E - Vr) / L;
10               i = integral(i0,di);     /*  電流   */
11               Vl = di * L;             /*  インダクタンス電圧   */
12               Vr = R * i;              /*  抵抗電圧   */
13               end;
```

7.2.1. 予約語

次の 2 つが予約語としてシステムにより定義されています.

1. TIME : シミュレーション時刻を保持しています. 全モジュールにおいて適用できます.
2. CN : コンポーネント番号を保持しています. 全モジュールにおいて適用できます.

7.2.2. ライブラリ関数

次の 5 種類の関数が用意されています.

- pulse

  記述) $y = pulse(a, b, c, d, e)$

  引数)

  1. a : 入力開始時刻
  2. b : 入力初期値
  3. c : 高さ
  4. d : 時間幅
  5. e : 周期

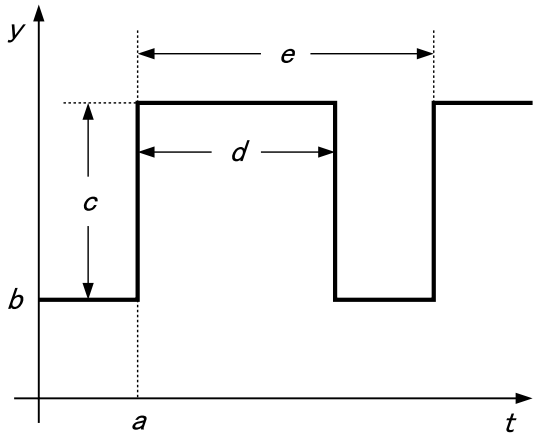


図 7.3. パルス関数

- ramp

  記述) $y = ramp(a, b, c)$

  引数)

  1. a : 入力開始時刻
  2. b : 入力初期値
  3. c : 勾配

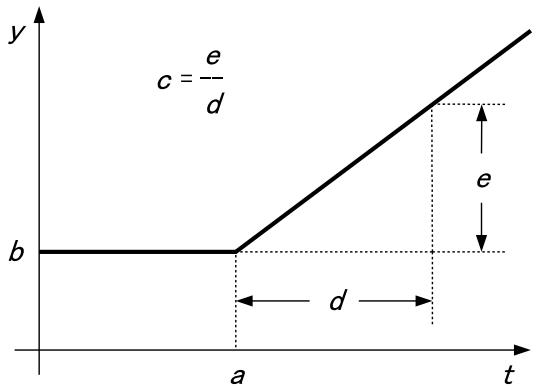


図 7.4. ランプ関数

- integral

  記述) $y = integral(a, p)$

  式) $y(t) = \int pdt, \quad y(0) = a$

- sigmoid

  記述) $y = sigmoid(-x)$

  式) $y = \frac{1}{1+\exp(x)}$

- rcasb

  記述) $y = rcasb(v, a, b, c, d, e, f, g)$

  式) $y = \frac{a \cdot \exp\{b \cdot (v+c)\} + d \cdot (v+e)}{\exp\{f \cdot (v+c)\} + g}$

NCS ライブラリ関数の他に，記述には C 言語の数学ライブラリ関数が使用できます．表 7.2.「NCS 言語で使える C 言語数学ライブラリ関数の例」 に，その代表的な例の記述と式を示します．

また，ユーザが定義した関数も使用できます．

表 7.2. NCS 言語で使える C 言語数学ライブラリ関数の例

関数名	記述	式
exp	$y = \exp(x)$	$y = e^x$
pow	$y = pow(x, a)$	$y = x^a$
sin	$y = \sin(x)$	$y = \sin(x)$
cos	$y = \cos(x)$	$y = \cos(x)$
tan	$y = \tan(x)$	$y = \tan(x)$

7.2.3. モジュール記述

あるシステムに関する一連のモデル記述をモジュールとして記述します．電気回路モデルの記述や膜モデルによるイオン電流レベルでの記述が可能です．モジュールはいくつかの文を用いて記述します．ここでは，モジュール記述に使用する文とその構成について説明します．

各文の基本的な構成は次のようになります．

宣言子　：　文の内容　；

どの文の記述を行うか宣言する宣言子で始まり，コロン ":" で区切って文の内容を記述します．各文はセミコロン ";" で終わらなければいけません．文の中には以下の，"module"，"output"，"function" は必ず記述しなければいけません．その他の文は状況に応じて使用します．

1. module [必須]

   モジュール名を定義します．先頭が英文字で始まる 8 文字以内の英数字綴りをモジュール名として宣言します．

   記述例)

   ```
   /*  "circuit"というモジュール名を宣言  */
   module  :  circuit;
   ```

2. exinput [任意]

   セルモジュールにおいて，モデルの外部入力変数名を定義します．外部変数名は 1 つです．

   記述例)

   ```
   /*  "E"を外部入力変数として定義します.   */
   exinput  :  E;
   ```

3. output [必須]

   モジュールの出力変数名を定義します．出力変数の数は 1 つです．

   記述例)

   ```
   /*  "i"を出力変数として定義  */
   output  :  i;
   ```

4. observable [任意]

   後述する function 文で使用する変数のうち，計算値を観測したい変数名を宣言します．宣言した変数はシミュレーション実行時に，nout 関数で出力指定を行うことによって値を観測できます．

   記述例)

   ```
   /*  "Vr","Vl"を観測する変数として定義  */
   observable  :  Vr, Vl;
   ```

5. constant [任意]

   function 文で使用する定数の，定数名とその値を宣言します．ここで定義された定数名は必ずしも function 文で使用される必要はありません．ただし，その分メモリを取っていますので，不必要な定数は宣言しないほうが良いでしょう．

   記述例)

   ```
   /*  "L"を定数として使用し，0.1 を代入することを宣言  */
   constant  :  L = 0.1;
   ```

6. parameter [任意]

   パラメーター変数の定義を行います．ここで定義されたパラメーターはシミュレーション条件ファイルに登録され，シミュレーション実行時に npara 関数によって値を変更することができます．ここで定義されたパラメーターは必ずしも function 文で使用される必要はありません．

   記述例)

```
/*  "R", "i0"をパラメーターとして使用し,
    それぞれ 10, 0 を代入することを宣言  */
parameter  :  R = 10, i0 = 0;
```

7. function [必須]

   数式によりモジュールの特性を記述します．変数は使用される前に必ず値が定義されていなければなりません．数式はセミコロンで区切られます．

   function 文内の記述に柔軟性を持たせるため，if 文が用意されています．

   書式)

   ```
   if ( 条件式 ) { 文 1 } [ else { 文 2 } ]
   ```

   表 7.3. 条件式の関係演算子

演算	記号
等値	=
非等値	<>
比較	<，>，<=，>=
論理和	.or.
論理積	.and.
否定	.not.

   条件式が真なら文 1 が，偽なら文 2 が実行されます．条件式に用いる関係演算子を 表 7.3.「条件式の関係演算子」 に示します．

   記述例)

   ```
   /*  "Vl", "Vr"の式を記述  */
   function  :
   Vl = di * L;
   Vr = R * i;
   ```

8. コメント

   "/*"，"*/" で囲まれた部分はコメントになります．文中のどこでも，また複数行に渡る使用も可能です．

文の並びは module 文を最初，function 文を最後に記述する以外は順不動で，module，output，function 文以外の文は省略することが可能です．なお，モジュール記述の終わりに必ず，"end;" をつけることを忘れないでください．

7.2.4. モデル記述例

ここでは，NCS 言語で記述する方法を，実際のモデルを使って説明します．

●RL 直列回路モデル

ここでは，前節の 図 7.2.「RL 直列回路」 の RL 直列回路のモデル記述について説明します．RL 直列回路モデルのインダクタンス電流は以下のように表されます．

$$V_R = R \cdot i \qquad (7.1)$$

$$V_L = L \cdot \frac{di}{dt} \qquad (7.2)$$

$$\frac{di}{dt} = \frac{1}{L}(E - V_R) \qquad (7.3)$$

式 (7.1)，(7.2) を NCS 言語によって記述すると，

```
Vr = R * i;
Vl = di * L;
```

回路を流れる電流 i は微分方程式 (7.3) で記述されていますので，次のようになります．

```
di = (E - Vr) / L;
i = integral(i0,di);
```

次に，こうした特性を持つセルタイプモジュールを記述します．モジュール名は "circuit" とします．

```
module  :   circuit;
```

外部入力に電圧源 E を，そしてインダクタンス電圧 Vl を出力するものとします．

```
exinput:   E;
output:    Vl;
```

またここで抵抗電圧 V_r，電流 i を観測することにします．

```
observable:   Vr, i;
```

インダクタンス L を定数として宣言します．

```
constant:   L = 0.1;
```

抵抗 R，初期電流 $i0$ をパラメーターとして宣言します．

```
parameter:   R = 10., i0 = 0;
```

function 文には，前述した記述を用います．

こうして，リスト 1 の NCS 言語による記述が完成します．

●振り子の運動モデル

ここでは，力学系の簡単なシミュレーションとして，図 7.5.「振り子の運動」 に示す振り子の運動について説明します．一端を固定した長さ l [m] の糸の他端に重さ m [kg] のおもりをつるし，鉛直面内で振動させます．

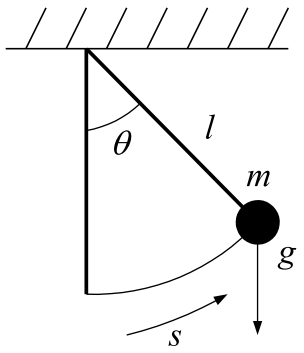


図 7.5. 振り子の運動

最下点から円弧にそって測った長さを s [m]($s = l\theta$) とすれば，その運動方程式は

$$m\frac{d^2s}{dt^2} = -mg\sin\theta \qquad (7.4)$$

と表せます．これより NCS のモデル記述は

```
module:      PENDULUM;
output:      theta;
observable:  d2theta,dtheta,theta;
parameter:   m = 1.0,g = 9.8,l = 1.0,theta0 = 3.141592 / 180.0 * 45.0;
function:
             d2theta = -g / l * sin(theta);
             dtheta = integral(0.0,d2theta);
             theta = integral(theta0,dtheta);
end;
```

となります．運動方程式は 2 階微分の形になっていますので，これを 1 階の連立微分方程式に書き換え，θ を求める記述とします．

比較のため解析的に運動方程式の解を求めます．問題を簡単にするために θ [rad] を 1 より充分小さいと仮定し，平衡点のまわりで線形近似します．このとき運動方程式は

$$m\frac{d^2s}{dt^2} = -mg\theta \qquad (7.5)$$

となります．これを解くと

$$\theta = \theta_0 \cdot \sin\left(\sqrt{\frac{g}{l}}t + \phi\right) \qquad (7.6)$$

となります．

モデルファイル名を "pendulum.mdl" とし，シミュレーションするための SATELLITE スクリプトをリスト 2 に示します．シミュレーションの実行ついては，項 7.3.「NCS の使用法」 で詳しく説明します．

θ の初期値を 10 度として振り子を運動させた (減衰なし) 場合のシミュレーション結果を 図 7.6.「振り子の運動モデルのシミュレーション結果」 に示します．実践が解析解，四角形は 10 刻み毎のモデル出力になります．

リスト 2 : 振り子の運動モデルのシミュレーション実行用バッチスクリプトの例

```
1.  npp("pendulum")      #モデルファイルの登録とプリプロセッサの起動
2.  nlink();                                        # 実行ファイルの作成
3.  ntime(10.0,0.01,0.01,0.01);                     # 時間条件の設定
4.  time = (0~(10.0 / 0.01)) * 0.01;                # 時間軸データ
5.  series theta;
6.  nout(theta,"pendulum",0,1);                     # 外部出力条件の設定
7.  theta0 = PI / 180 * 10.0;                       # 初期角度 10 度に設定
8.  npara("PENDULUM","theta0",theta0);              # パラメータの変更
9.  ncal();                                         # シミュレーションの実行
10. phi = PI / 180 * 90;                            # 解析解の計算
11. m = 1.0;
12. l = 1.0;
13. gra = 9.8;
14. true_theta = theta0 * sin(sqrt(gra / l) * time + phi);
15. theta = theta / (PI / 180.0);                   # [rad]から[deg]に変換
16. true_theta = true_theta / (PI / 180.0);
17. wopen(1,"A4",0,1);                              # グラフ表示
18. title(1,"time[s]","angle[deg]");
19. scale("N","F","N","F",0,10,-30,30);
20. axis(1,1,"XY","XY",5,0,3,2,15,0);
21. frame();
```

```
22. graph(theta,time,0,1,10,0,2);
23. graph(true_theta,time,0,0,0,0,0);
```

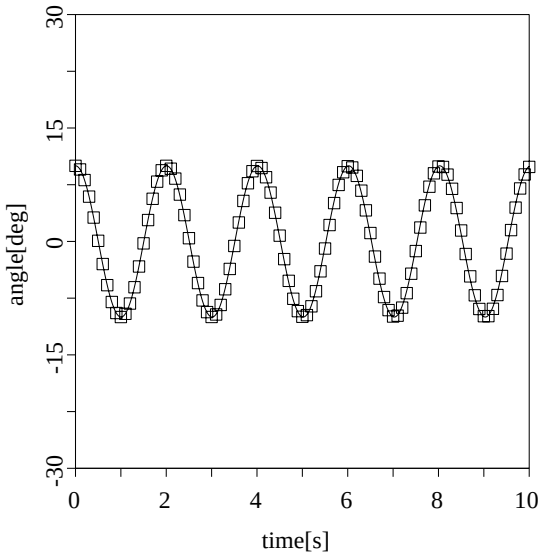


図 7.6. 振り子の運動モデルのシミュレーション結果

●Hodgkin-Huxley モデル

ここではイカの巨大神経軸索の電気的特性をモデル化した Hodkin-Huxley (以下 H-H モデル) の記述について詳しく説明します.

H-H モデルのナトリウム電流は次の式で表されます.

$$I_{na} = \bar{g}_{Na} \cdot m^3 \cdot h\,(V - E_{Na}) \qquad (7.7)$$

$$\frac{dm}{dt} = \alpha_m(1-m) - \beta_m \cdot m \qquad (7.8)$$

$$\alpha_m = \frac{0.1(25-V)}{\exp\left[\frac{25-V}{10}\right] - 1} \qquad (7.9)$$

$$\beta_m = 4\exp\left(-\frac{V}{18}\right) \qquad (7.10)$$

$$\frac{dh}{dt} = \alpha_h(1-h) - \beta_h \cdot h \qquad (7.11)$$

$$\alpha_h = 0.07\exp\left(-\frac{V}{20}\right) \qquad (7.12)$$

$$\beta_h = \frac{1}{\exp\left[\frac{30-V}{10}\right] + 1} \qquad (7.13)$$

$\bar{g}_{Na}$: ナトリウムの膜コンダクタンス，120[mS/cm^2]

E_{Na}: ナトリウムの反転電位，115[mV]

式 (7.9)，(7.10) を NCS 言語によって記述すると，

```
am = 0.1 * (25. - V) / (exp((25. - V) / 10.) - 1);
bm = 4. * exp(-V / 18.);
```

ただし，α_m は V = 25[mV] 時の値を別に与える必要がありますので，次のように書き換えます.

```
if(V != 25.){
am = 0.1 * (25. - V) / (exp((25. - V) / 10.) - 1);
}else{
am = 0.1 * 10.;
}
```

ナトリウムチャンネルの開閉確率 m は微分方程式で記述されていますので，次のようになります.

```
dmNa = am * (1. - mNa) - bm * mNa;
mNa = integral(mNa0,dmNa);
```

同様にして h の記述は式 (7.11)，(7.12)，(7.13) より次のようになります.

```
ah = 0.07 * exp(-V / 20.);
bh = 1. / (exp((30. - V) / 10.) + 1.);
dhNa = ah * (1. - hNa) - bh * hNa;
hNa = integral(hNa0,dhNa);
```

最終的にナトリウム電流 I_{Na} は，式 (7.7) より次のようになります.

```
INa = GNa * pow(mNa,3.0) * hNa * (V - VNa);
```

カリウム電流は次式で表されます.

$$I_k = \bar{g}_K \cdot n^4 (V - E_K) \qquad (7.14)$$

$$\frac{dn}{dt} = \alpha_n (1 - n) - \beta \cdot n \qquad (7.15)$$

$$\alpha_n = \frac{0.01(10 - V)}{\exp\left[\frac{10-V}{10}\right] - 1} \qquad (7.16)$$

$$\beta_n = 0.125 \exp\left(-\frac{V}{80}\right) \qquad (7.17)$$

$\bar{g}_K$: カリウムの膜コンダクタンス，36[mS/cm^2]

E_K : カリウムの反転電位，-121[mV]

カリウムチャンネルの開閉確率はナトリウムの場合と同様にして，式 (7.15)，(7.16)，(7.17) より，次のように記述されます.

```
if(V != 10.){
     an = 0.01 * (10. - V) / (exp((10. - V) / 10.) - 1.);
}else{
     an = 0.01 * 10;
}
bn = 0.125 * exp(-V / 80.);
dnK = an * (1. - nK) - bn * nK;
nK = integral(nK0,dnK);
```

カリウム電流は式 (7.14) より，次のように記述されます.

```
IK = GK * pow(nK,4.0) * (V - VK);
```

漏れ電流は次式で表されます.

$$I_L = \bar{g}_L (V - E_L) \qquad (7.18)$$

$\bar{g}_L$: 漏れ電流の膜コンダクタンス，0.3[mS/cm^2]

E_L : 漏れ電流の反転電位，10.6[mV]

上式より，漏れ電流の記述は次のようになります.

```
Il = Gl * (V - Vl);
```

膜を横切る電流 I は，次式で表されます.

$$C_m \frac{dV}{dt} = I - I_{Na} - I_K - I_L \qquad (7.19)$$

注入電流を I_{ex} とすると，全電流は次式で表されます.

$$I = I_{ex} - I_{Na} - I_K - I_L \qquad (7.20)$$

これより，式 (7.19) は，次式のように変形できます.

$$\frac{dV}{dt} = \frac{I}{C_m} \qquad (7.21)$$

以上より，NCS 言語での記述は次のようになります.

```
Iall = Iex - INa - IK - Il;
dV = Iall / Cm;
V = integral(V0,dV);
```

次に，こうした特性を持つセルタイプモジュールを記述します．モジュール名は，"hh" とします.

```
module:   HH;
```

外部入力に注入電流 I_{ex}，膜電位 V を出力するものとします.

```
exinput:   Iex;
output:    V;
```

ここで，ナトリウム電流 I_{Na}，カリウム電流 I_K，漏れ電流 I_L を観測することにします.

```
observable:   INa, IK, Il;
```

各イオン電流の反転電位 E_{Na}, E_K, E_L を，定数として宣言します.

```
constant:   VNa = 115.0, VK = -12.0, Vl = 10.6;
```

膜容量 C_m，積分の初期値，各イオンの膜コンダクタンス $\bar{g}_{Na}, \bar{g}_K, \bar{g}_L$ をパラメーターとして宣言します.

```
parameter:   Cm = 1.0, mNa0 = 0.05293, GNa = 120., GK = 36.,
             Gl = 0.3, hNa0 = 0.5961, nK0 = 0.3177, V0 = 0.;
```

function 文には，前述した記述を用います．こうして，リスト 3 の NCS 言語による記述が完成します.

パルス状の電流を注入した時のモデルシミュレーションを実行し，結果をグラフに表示するスクリプトの例をリスト 4 に示します．実行方法の詳細は 項 7.3.「NCS の使用法」 で説明します．結果を 図 7.7.「H-H モデルのシミュレーション結果」 に示します.

リスト 3 : NCS 言語による Hodgkin-Huxley モデルの記述

```
1   /*   HH module   */
2   module:      hhmodel;
3   exinput:     Iex;
4   output:      V;
```

```
5   observable:  INa, IK, Il, Ig;
6   constant:    VNa = 115.0, VK = -12.0, Vl = 10.6;
7   parameter:   Cm = 1.0, mNa0 = 0.05293, GNa = 120.0, GK = 36.0,
                 Gl = 0.3, hNa0 = 0.5961, nK0 = 0.3177, V0 = 0.0;
8   function:
9                if(V != 25.0){
10                    am = 0.1*(25.0-V)/(exp((25.0-V)/10.0)-1.0);}
11               else{
12                    am = 0.1*10.0;}
13               bm = 4.0*exp(-V/18.0);
14               dmNa = am*(1.0-mNa)-bm*mNa;
15               mNa = integral(mNa0,dmNa);
16               ah = 0.07*exp(-V/20.0);
17               bh = 1.0/(exp((30.0-V)/10.0)+1.0);
18               dhNa = ah*(1.0-hNa)-bh*hNa;
19               hNa = integral(hNa0,dhNa);
20               INa = GNa*pow(mNa,3.0)*hNa*(V-VNa);
21               if(V != 10.0){
22                    an = 0.01*(10.0-V)/(exp((10.0-V)/10.0)-1.0);}
23               else{
24                    an = 0.01*10.0;}
25               bn = 0.125*exp(-V/80.0);
26               dnK = an*(1.0-nK)-bn*nK;
27               nK = integral(nK0,dnK);
28               IK = GK*pow(nK,4.0)*(V-VK);
29               Il = Gl*(V-Vl);
30               Iall = Iex-INa-IK-Il;
31               dV = Iall/Cm;
32               V = integral(V0,dV);
33               end;
```

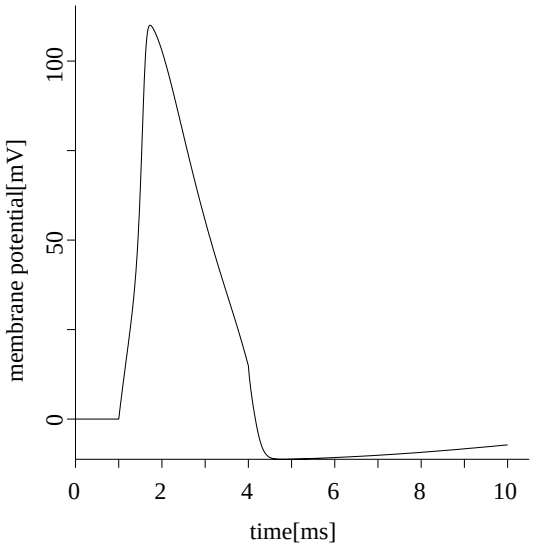


図 7.7. H-H モデルのシミュレーション結果

リスト 4 : H-H モデルのシミュレーション実行用バッチスクリプトの例

```
1   nasgn("hhmodel");                    # モデルファイルの登録
2   npp();                               # プリプロセッサの起動
3   nlink();                             # 実行ファイルの作成
```

```
4   ntime(10,0.001,0.01,1);                    # 時間条件の設定
5   nstim("hhmodel",0,"P",1,0,100,3,999);      # 外部入力条件の設定

6   series V,Iin,INa0;
7   nout(Iin,"hhmodel",0,2);                   # 外部出力条件の設定
8   nout(V,"hhmodel",0,1);
9   nout(INa0,"hhmodel",0,3,"INa");
10  ninteg("R");                               # 積分方式の変更
11  ncal();                                    # シミュレーションの実行

12  wopen(1,"A4",0,0);                         # グラフ表示
13  time=(0~1000)/100;
14  graph(V,time,0,0,0,1,3.5);
15  title(1,"time[ms]","membrane potential[mv]");
16  axis(1,1,"XY","XY",5,0,3,0,0,0);
```

7.3. NCS の使用法

本システムは SATELLITE 上で稼動しており，会話型処理，またはバッチファイルを用いた一括処理ができます．コマンドについての詳細は，SATELLITE 言語リファレンスマニュアルを参照して下さい．

NCS でシミュレーションを行う場合の手順は，次のようになっています．

1. モデルファイルの作成

   NCS 言語を用いてモデルを記述したファイルを作成します．NCS 言語を用いた記述方法の詳細は 項 7.2.「NCS 言語」 を参照してください．

2. モデルファイルの登録

   シミュレーションを行うモデルの，モデル記述ファイルを登録します．NCS は以後，このモデルファイルに対して処理を行います．

3. 実行ファイル，シミュレーション条件ファイルの作成

   登録されたモデルファイルに対してプリプロセッサを起動，そしてリンクを行うことにより，実行ファイルとシミュレーション条件ファイル群を作成します．

4. シミュレーション条件の設定

   シミュレーションの際に必要なシミュレーション時間，外部入力，出力変数などの設定を行います．

5. シミュレーションの実行

   シミュレーションを実行します．

6. シミュレーション結果の表示・解析

   シミュレーションが終わって得られた結果のグラフを表示します．

以下にはこの手順の詳細を，リスト 1 のモデル記述を使ったシミュレーションを例に用いて述べていきます．

7.3.1. モデルファイルの作成

NCS 言語の仕様に従って，モデルファイルを記述します．モデルファイルのファイル名は拡張子に ".mdl" を付けてください．

注) ver2.95

次項で説明する nassign 関数でモデルファイル名が登録してあるとき，ne 関数を実行すると，そのモデルファイルを対象としてエディタが実行されます．この時立ち上がるエディタのデフォルトは vi エディタですが，UNIX の環境変数 editor を設定することによって，好きなエディタに変更できます．

7.3.2. モデルファイルの登録

後述する npp 関数によるプリプロセッサの起動や，nlink 関数による実行ファイルの作成は，SATELLITE のワークエリア内に登録されたモデルファイルに対して実行されます．従って，NCS を使ってシミュレーションを行うためには，使用するモデルを登録する必要があります．このモデルファイルの登録は nassign 関数か，次項に述べる npp 関数によってなされます．npp 関数は引数にモデルファイル名をとり，そのファイル名のモデルファイルを登録します．このモデルファイル名には拡張子 ".mdl" を付ける必要はありません．

```
[]satellite[]~/home/demo:[1]% nassign("circuit")
```

上記の例では，"circuit.mdl" をモデルファイルとして登録しています．

7.3.3. 実行ファイル，シミュレーション条件ファイルの作成

npp 関数実行により，プリプロセッサが起動し，NCS 言語で記述されたモデルファイルが C 言語ソースファイルとシミュレーション条件ファイル群に変換されます．

```
[]satellite[]~/home/demo:[2]% npp()
```

とすることにより，プリプロセッサが起動します．

npp 関数はモデルファイル名を引数にとることができます．nassign 関数でモデルファイルを登録しなくても，npp 関数でモデルファイル名を指定することで登録が可能です．この場合も nassign 関数同様，モデルファイル名に拡張子 ".mdl" を付ける必要はありません．

```
[]satellite[]~/home/demo:[3]% npp("circuit")
```

とすることにより，"circuit.mdl" をモデルファイルとして登録して，プリプロセッサを起動することになります．

npp 関数実行後，nlink 関数により，C 言語のソースファイルをコンパイル後，NCS のライブラリとリンクされ，実行ファイルが作成されます．

```
[]satellite[]~/home/demo:[4]% nlink()
```

7.3.4. シミュレーション条件の設定

シミュレーションを行うにあたって，各種シミュレーション条件を設定します．このシミュレーション条件は npp 関数が実行された時点でモデルファイルより設定されているものと，設定されていないものが存在します．シミュレーションを行うにあったて，設定の必要なシミュレーション条件がありますので注意してください．

以下に述べるシミュレーション条件の設定関数は，npp 関数実行によって生成されたシミュレーション条件ファイル群に対して行われます．従って，シミュレーション条件設定関数実行以前に，npp 関数が実行されている必要があります．npp 関数が実行されていない状態で，シミュレーション条件設定関数を実行した場合，

```
sl : Error [<NCS:nout> No.1] : Improper Model File Name
near at line <n>
```

というエラーが表示されます．また，各種シミュレーション条件を設定後，再度 npp 関数を実行すると，シミュレーション条件は初期化されます．

●時間条件

時間条件は ntime 関数によって設定します．ntime 関数の書式は次のようになっています．

書式) ntime(last, cal, str, itv)

引数)

1. last : 計算時間
2. cal : 計算刻み
3. str : 計算結果の抽出時間幅
4. itv : バッファに書き込む時間幅

計算時間はシミュレーションが終わるまでの時間です．計算刻みは，数値計算の際の時間刻み幅で，これの大小によって誤差や，収束度，シミュレーションの実行時間が左右されます．適当と思われる値に設定してください．ストア刻みは，計算結果を抽出する時間幅で，この値を小さくすればするほど，細かい時間幅の実行結果が得られます．ただし，データ容量の面から言えば，実行結果の容量が大きくなります．使用しているコンピューター環境を考えて適切な値に設定してください．書き込み刻みは，実行結果を後で説明する nout 関数で指定されたバッファに書き込む時間幅です．書き込み刻みを小さくしすぎると，シミュレーション実行時間が非常に長くなりますので注意してください．

```
[]satellite[]~/home/demo:[5]% ntime(0.1,0.005,0.005,0.05)
```

としたときには，計算時間は 0.1[s]，計算刻みは 0.005[s]，計算結果の抽出時間幅を 0.005[s]，バッファに書き込む時間幅は 0.05[s] と設定したことになります．

設定した時間条件は nsclist 関数の引数に "T" を用いて表示することができます．

```
[]satellite[]~/home/demo:[6]% nsclist("T")

TIMER

  Last  Time =    0.1
  Calc. Step =  0.005
  Store Step =  0.005
  BufferStep =   0.05
```

nsclist 関数を用いることによって，時間条件が正しく設定されているか確認してください．

●外部入力条件

外部入力条件は nstim 関数によって設定します．nstim 関数はモジュール名，コンポーネント番号，入力波形を指定する書式になっています．入力波形には，パルス関数，ランプ関数とファイル入力，バッファ入力が用意されています．nstim 関数の基本的な書式は次のようになっています．

書式) nstim(mdul, com, type, p1 [, p2, p3, p4, p5])

引数)

1. mdul : モジュール名
2. com : コンポーネント番号
3. type : 入力波形の指定 ("P", "R", "B")
4. $p_1 \sim p_5$: パラメーター列

入力波形	p_1	p_2	p_3	p_4	p_5
パルス関数	入力開始時刻	入力初期値	高さ	時間幅	周期
ランプ関数	入力開始時刻	入力初期値	傾き		
バッファ入力	バッファ名				

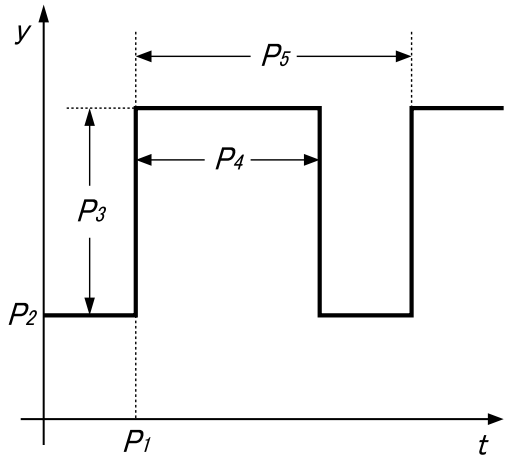


図 7.8. パルス関数

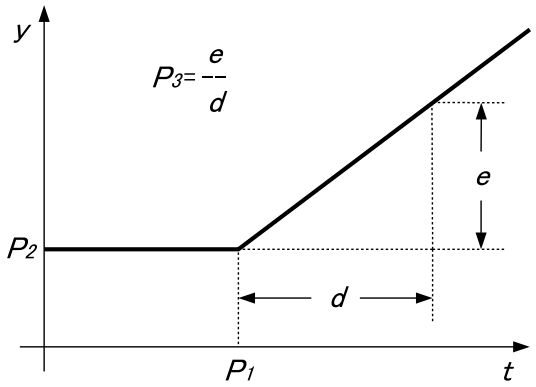


図 7.9. ランプ関数

この関数で指定するモジュールでは，exinput 文による外部入力の記述がなされている必要があります.

```
[]satellite[]~/home/demo:[7]% nstim("circuit",0,"P",0,1,0,10,999)
```

とすることにより，モジュール名 "circuit" のモジュールに対してパルス関数を入力する設定を行います．シングルモジュールからなるモデルを扱う場合は，コンポーネント番号を必ず 0 に設定します.

設定した外部入力条件は nsclist 関数の引数に "S" を用いて表示することができます.

```
[]satellite[]~/home/demo:[8]% nsclist("S")

EXTERNAL  INPUTS
   Data No.      Component    Function Data No.
   1          circuit(0)

No.1  Function     start_tm init_out height width   period
      <  PULSE>    [   0]   [   1]   [   0] [   10] [   999]
```

nsclist 関数を用いることによって，外部入力条件が正しく設定されているか確認してください.

●出力条件

出力条件は nout 関数によって設定します．nout 関数は出力値を格納するバッファ名，モジュール名，コンポーネント番号，出力情報の属性を引数に持ちます．ただし，出力情報の属性を内部変数値に設定した場合のみ，内部変数名を引数に指定する必要があります．nout 関数の基本的な書式は次のようになっています.

書式) nout(buff, mdl, com, type [, val])

引数)

1. buff : 出力値を格納するバッファ名
2. mdl : モジュール名
3. com : コンポーネント番号
4. type : 出力情報の属性 (1 : 出力値，2 : 入力値，3 : observable に指定した変数値)
5. val : type = 3 を指定したとき，observable に指定した変数名を指定する．

ここで，出力値とはモジュール名 mdl のモジュールの記述において output 文で指定されている変数の値が入ります．また入力値は，exinput 文で指定されている変数の値が入ります．

なお，出力値を格納するバッファは nout 関数実行以前に series 型のバッファとして宣言されている必要があります．例えば，nout 関数実行以前に，

```
[]satellite[]~/home/demo:[9]% series Vin, i, vr, vl
```

とすれば，nout 関数で，バッファ Vin，i，vr，vl を出力値の格納用バッファとして扱うことができます．バッファが宣言されていない場合には，nout 関数実行時にエラーが表示されます．注意してください．

```
[]satellite[]~/home/demo:[10]% nout(Vin,"circuit",0,2)
[]satellite[]~/home/demo:[11]% nout(vl,"circuit",0,1)
```

によって，モジュール名 "circuit" のモジュールの入力値がバッファ Vin に，出力値が vl にそれぞれ出力されます．

また，その他の内部変数を出力するためには次のように設定します．

```
[]satellite[]~/home/demo:[12]% nout(vr,"circuit",0,3,"Vr")
[]satellite[]~/home/demo:[13]% nout(i,"circuit",0,3,"i")
```

これによって，モジュール名 "circuit" のモジュールの内部変数 "Vr" の値がバッファ vr に，"i" の値がバッファ i にそれぞれ出力されます．

設定した外部出力条件は nsclist 関数の引数に "O" を用いて表示することができます．

```
[]satellite[]~/home/demo:[14]% nsclist("O")

OUTPUT
   Variable    OUTPUT    VARIABLE
   Vin [   0]  EX.INPUT  OF circuit(0)
   vl  [   0]  OUTPUT    OF circuit(0)
   vr  [   0]  Vr        OF circuit(0)
   i   [   0]  i         OF circuit(0)
Max of Number of Output = 255
```

●パラメーター値の変更

パラメーター値は npara 関数によって変更することができます．npara 関数の書式は次のようになっています．

書式) npara(mdl, var, num)

引数)

1. mdl : モジュール名
2. var : パラメーター名

3. num : パラメーターに設定する値

npara 関数の実行にあたって，モデルファイルの記述で，モジュール名 mdl の変数 var が parameter 文で宣言されている必要があります．

```
[]satellite[]~/home/demo:[15]% npara("circuit","R",20)
```

npara 関数で変更した値は，npp 関数等によってシミュレーション条件ファイル群が初期化されるまで有効になります．

現在のパラメーター値は，nlist 関数で表示することができます．

```
[]satellite[]~/home/demo:[16]% nlist("circuit")
Parameter    List
Module name :   circuit
   R        =   [    20]
   i0       =   [     0]
```

nlist 関数はモジュール名を引数とし，そのモジュールのパラメーター値を表示します．

積分方式の選択

通常，数値積分法には自動刻み積分法が用いられています．NCS はこの他に，ルンゲクッタ - ギル法とオイラー法を備えており，ninteg 関数によって，これらの積分法を用いることができます．ninteg の書式は次のようになっています．

書式) ninteg(type)

引数)

1. type : 積分方式 ("F" : 自動刻み積分法，"R" : ルンゲクッタ‐ギル法，"E" : オイラー法)

7.3.5. シミュレーションの実行

各種シミュレーション条件を設定後，ncal 関数によりシミュレーションが実行されます．

```
[]satellite[]~/home/demo:[17]% ncal()
```

と ncal 関数を実行すると，

```
## NCS Ver.6.8.3 SIMULATION PROGRAM on osf4.0 ##
>> THE CALCULATION HAS FINISHED ................... !! << 0.0% done.
```

と表示されます．"0.0%" は現在の実行の割合をパーセンテージで示しており，この数字が "100%" に達すると，

```
## NCS Ver.6.8.3 SIMULATION PROGRAM on osf4.0 ##
>> THE CALCULATION HAS FINISHED ................... !! << 100.0% done.
```

と表示が変わり，シェルは関数待ち状態になって，シミュレーションが終了したことを示します．

7.3.6. バッチファイルの使用

ここまで述べてきた，シミュレーションを実行するまでを 1 つのバッチファイルにまとめることができます．そして，inline 関数を使用することにより，バッチファイルを用いたシミュレーションの実行が行えます．

リスト 5 : シミュレーションを行うためのバッチスクリプトの例

```
1   nassign("circuit");                    # モデルファイルの起動
2   npp();                                 # プリプロセッサの起動
3   nlink();                               # 実行ファイルの作成
```

```
4   ntime(0.1,0.005,0.005,0.05);              # 時間条件の設定
5   nstim("circuit",0,"P",0,1,0,10,999);      # 外部入力条件の設定
6   series Vin,i,vr,vl;
7   nout(Vin,"circuit",0,2);                  # 外部出力条件の設定
8   nout(i,"circuit",0,1);
9   nout(vr,"circuit",0,3,"vr");
10  nout(vl,"circuit",0,3,"vl");
11  ninteg("R");                              # 積分方式の変更
12  ncal();                                   # シミュレーションの実行
```

```
[]satellite[]~/home/demo:[18]% inline("バッチファイル名")
```

リスト 5 にモデルファイル circuit.mdl を使ったシミュレーションを行うための，バッチファイル例を示します．このファイルを "circuit.sl" としますと，シミュレーションの実行は次のようになります．

```
[]satellite[]~/home/demo:[19]% inline("circuit.sl")
```

7.3.7. シミュレーション結果の表示・解析

計算結果は，nout 関数で指定したバッファに格納され，SATELLITE の他のシステムモジュールによって，結果のグラフ表示，解析などが行えます．例えば，circuit.sl を用いたシミュレーション結果は，GPM を使用することにより，次のようにしてグラフに表示することができます．

```
[]satellite[]~/home/demo:[20]% wopen(1,"A4",0,0)
[]satellite[]~/home/demo:[21]% time = (0~20) / 200
[]satellite[]~/home/demo:[22]% graph(i,time,0,0,0,0,0)
[]satellite[]~/home/demo:[23]% title(1,"time[s]","current[a]")
[]satellite[]~/home/demo:[24]% axis(1,1,"XY","XY",5,0,0,0,0,0)
```

グラフ表示の詳細は SATELLITE システムの GPM モジュールマニュアルを参照してください．

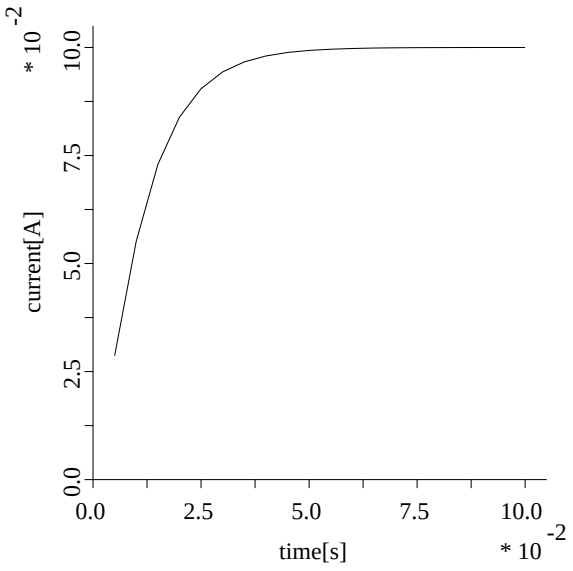


図 7.10. RL 直列回路モデルのシミュレーション結果

7.4. NCS による神経回路網のモデリングシミュレーション

7.4.1. モジュール化の概念

ここでは NCS におけるタイプ，モジュール，コンポーネントの概念を述べます．これらの関係を 図 7.11.「NCS における神経回路素子とモデル構造」，図 7.12.「実際の神経回路との対応」 に示します．

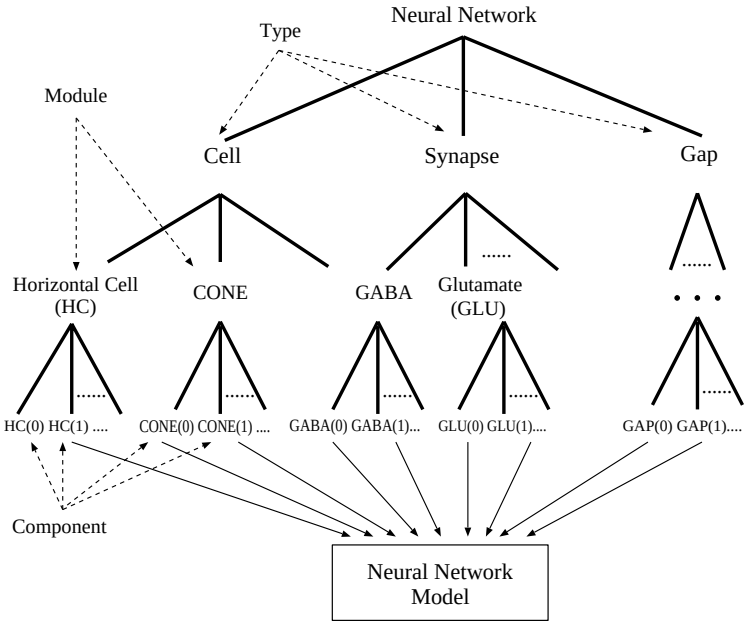


図 7.11. NCS における神経回路素子とモデル構造

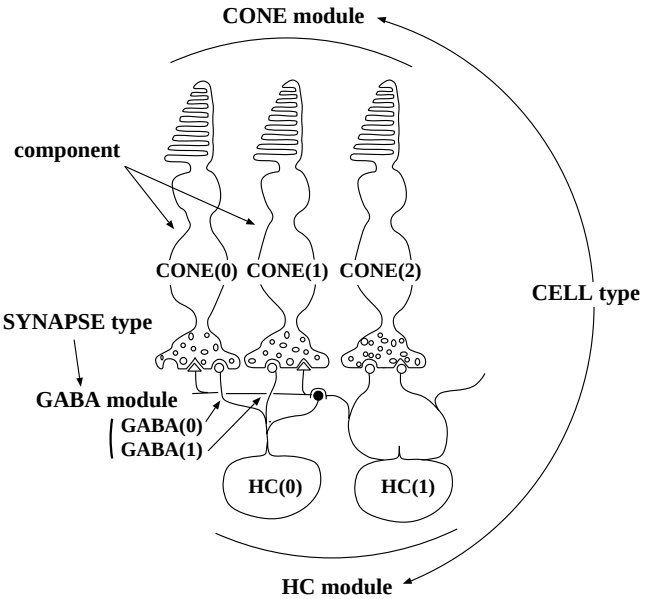


図 7.12. 実際の神経回路との対応

神経回路は，多数の神経細胞が化学シナプス，電気シナプス (ギャップジャンクション) を介して結合することにより構成されています．すなわち，神経回路は，神経細胞，化学シナプス，電気シナプスの 3 種類の組み合わせにより成り立っています．神経回路の各構成要素はそれぞれ特定の機能をもっており，神経細胞は信号の処理，化学シナプスは化学伝達物質による情報伝達，電気シナプスは電位差による情報の伝達という機能を

果たしています．NCS ではモデル記述を行う上で，これら 3 つの要素を基本の型 (タイプ) と考え，神経回路を神経細胞型 (cell type : セルタイプ)，化学シナプス型 (synapse type : シナプスタイプ)，電気シナプス型 (gap type : ギャップタイプ) としています．つまり，

神経回路の構成要素 = { cell type, synapse type, gap type }

と表記できます．

各タイプはさらに細分化でき，それらをモジュールと呼びます．モジュールは，各タイプの中で同一の特性を持つ素子全体を表現するものです．例えば網膜の場合，セルタイプモジュールには光を受容する錐体視細胞 (CONE)，視覚物質を修飾，制御する水平細胞 (HC) など特性の異なる神経細胞があります．そこで，NCS では各タイプを異なる特性を有する素子ごとに分割し，分割された一つ一つをモジュールと定義します．セルタイプは

cell type = { HC module, CONE module, … }

と表され，他のタイプも同様です．

モジュールは同じ特性を持つ素子一つ一つの集合であり，HC モジュールは次のように表現できます．

HC module = { HC[0], HC[1], … }

HC[0]，HC[1]，…はコンポーネントと呼ばれ，これは神経回路網を構成する素子の実態に対応します．コンポーネントはそれの属するモジュール名に数字を添え，番号付けすることにより表され，一つの素子が特定されます．

7.4.2. NCS 言語による結合モデル

NCS では大規模なモデルを記述しやすく，かつ柔軟性のあるモデル記述を実現するため，前節のモジュール化の概念のもと，結合記述を用いることができます．例えば，表 7.4.「Hodgkin-Huxley モデル」 に示した Hodgkin-Huxley モデル (以下 H-H モデル) が 図 7.13.「神経回路モデルの例」 のように抵抗で直列に継がっている神経回路モデルの記述は，リスト 6 のようになります．

- 2 〜 12 行目

  ネットワーク記述部．図 7.13.「神経回路モデルの例」 の結合状態を定義しています．

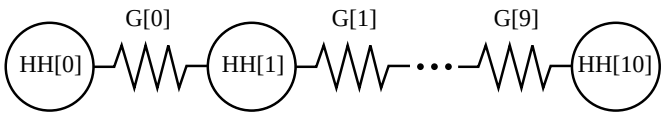


図 7.13. 神経回路モデルの例

- 14 〜 47 行目

  H-H モデルの記述，ギャップ性電流 I_g を入力として，膜電位 v が出力されます．function 文は，表 7.4.「Hodgkin-Huxley モデル」 の連立微分方程式の記述です．

表 7.4. Hodgkin-Huxley モデル

膜電位 $C_m \frac{dV}{dt} = I - I_{Na} - I_K - I_L$	V : 膜電位[mV] C_m : 膜容量[$\mu F/cm^2$] I : 全膜電流[$\mu A/cm^2$]
ナトリウム電流 $I_{Na} = \overline{g}_{Na} \cdot m^3 \cdot h(V - E_{Na})$ $\frac{dm}{dt} = \alpha_m(1-m) - \beta_m \cdot m$ $\alpha_m = \frac{0.1(25-V)}{\exp\left[\frac{25-V}{10}\right]-1}$ $\beta_m = 4\exp\left(-\frac{V}{18}\right)$ $\frac{dh}{dt} = \alpha_h(1-h) - \beta_h \cdot h$ $\alpha_h = 0.07\exp\left(-\frac{V}{20}\right)$ $\beta_h = \frac{1}{\exp\left[\frac{30-V}{10}\right]+1}$	I_{Na} : ナトリウム電流 [$\mu A/cm^2$] $\overline{g}_{Na}$: ナトリウムの膜コンダクタンス 120[mS/cm^2] E_{Na} : ナトリウムの反転電位 115[mV]
カリウム電流 $I_k = \overline{g}_K \cdot n^4(V - E_K)$ $\frac{dn}{dt} = \alpha_n(1-n) - \beta_n \cdot n$ $\alpha_n = \frac{0.01(10-V)}{\exp\left[\frac{10-V}{10}\right]-1}$ $\beta_n = 0.125\exp\left(-\frac{V}{80}\right)$	I_K : カリウム電流[$\mu A/cm^2$] $\overline{g}_K$: カリウムの膜コンダクタンス 36[mS/cm^2] E_K : カリウムの反転電位 -12[mV]
リーク電流 $I_L = \overline{g}_L(V - E_L)$	$\overline{g}_L$: リーク電流の膜コンダクタンス 0.3[mS/cm^2] E_L : リーク電流の反転電位 10.6[mV]

- 49 〜 56 行目

  電気シナプスの記述．膜電位に応じて電流を出力する記述になっています．

このように NCS 言語による記述は，微分方程式などで表されるモジュール記述 (リスト 6 の 14 〜 62 行目), モジュール記述の結合状態を表したネットワーク記述 (リスト 6 の 2 〜 12 行目) の 2 つに大別できます．

ここでは，こうした結合記述を用いた結合モデルの記述および使用方法を説明します．

リスト 6 : NCS 言語による結合記述を用いた Hodgkin-Huxley モデルの記述

```
1   /*    Hodgkin-Huxley's cell model    */
2   type:          NETWORK;
3   module:        SQUID;
4   cell:          HH[11];
5   gap:           G[10];
6   connection:
7                  HH[0] < ( G[0] < HH[1] );
8                  for( n = 1; n <= 9; n++ ){
9                       HH[n] < (G[n-1] < HH[n-1] + G[n] < HH[n+1]);
10                 }
11                 HH[10] < ( G[9] < HH[9] );
```

```
12                  end;
13  /*    HH module    */
14  type:           CELL;
15  module:         HH;
16  exinput:        Iex;
17  input:          Ig;
18  output:         V;
19  observable:     INa, IK, Il, Ig;
20  constant:       VNa = 115.0, VK = -12.0, Vl = 10.6;
21  parameter:      Cm = 1.0, mNa0 = 0.05293, GNa =120.0, GK = 36.0,
22                  Gl = 0.3, hNa0 = 0.5961, nK0 = 0.3177, V0 = 0.;
23  function:
24                  if(V != 25.){               /* sodium current */
25                       am = 0.1 * (25.-v)/(exp((25. - V)/10.) - 1.);}
26                  else{
27                       am = 0.1 * 10.;}
28                  bm = 4. * exp(-V / 18.);
29                  dmNa = am * (1. - mNa) - bm * mNa;
30                  mNa = integral(mNa0,dmNa);
31                  ah = 0.07 * exp(-V / 20.);
32                  bh = 1. / (exp((30. - V) / 10.) + 1.);
33                  dhNa = ah * (1. - hNa) - bh * hNa;
34                  hNa = integral(hNa0,dhNa);
35                  INa = GNa * pow(mNa,3.0) * hNa * (V - VNa);
36                  if(V != 10.){               /* potassium current */
37                        an = 0.01*(10.-V)/(exp((10. - V)/10.) - 1.);}
38                  else{
39                        an = 0.01 * 10.; }
40                  bn = 0.125 * exp(-V / 80.);
41                  dnK = an * (1. - nK) - bn * nK;
42                  nK = integral(nK0,dnK);
43                  IK = GK * pow(nK,4.0) * (V - VK);
44                  Il = Gl * (V - Vl);         /* leakage current */
45                  Iall = Iex - INa - IK - Il + Ig;
46                  dV = Iall / Cm;
47                  V = integral(V0,dV);
48                  end;
49  /*    G module    */
50  type:           GAP;
51  module:         G;
52  input:          VOP(0.1,0);
53  output:         Ig;
54  parameter:      Gl = 5.0;
55  function:
56                  Ig = Gl * (VOP - POSOUT);
57  end;
```

7.4.3. 結合モデルの予約語

結合モデル記述の際は，項 7.2.1.「予約語」 の予約語に加えて，さらに次の 3 つの予約語がシステムにより定義されています.

1. PRECN

   化学・電気シナプスへの入力値を出力しているセルモジュールを，そのシナプスのシナプス前セルモジュールと呼びます．PRECN はシナプス前セルモジュールのコンポーネント番号を保持しています．化学シナプス，電気シナプスモジュールにおいて適用できます.

2. POSTCN

   化学・電気シナプスからの出力値を入力しているセルモジュールを，そのシナプスのシナプス後セルモジュールと呼びます．POSTCN はシナプス後セルモジュールのコンポーネント番号を保持しています．化学シナプス，電気シナプスモジュールにおいて適用できます．

3. POSOUT

   シナプス後セルモジュールからの出力値を保持しています．化学シナプス，電気シナプスモジュールにおいて適用できます．

7.4.4. 結合モデルにおけるモジュール記述

結合モデルで使用する各モジュールは次項に述べる文により記述します．これらの文の組み合わせはタイプによって決められており，ユーザはそれにしたがって記述を行います．以下には文の詳細とタイプ別記述法について順に説明していきます．

●文

結合記述では，項 7.2.3.「モジュール記述」 に加えて，さらにいくつかの文を使います．文の詳細を順に説明していきます．

1. type

   結合記述を用いる場合はモジュールのタイプを宣言しなければなりません．タイプには以下の 3 種類があります．

   - CELL : 細胞型 (セルタイプ)
   - SYNAPSE : 化学シナプス型 (シナプスタイプ)
   - GAP : 電気シナプス型 (ギャップタイプ)

   また，結合記述の場合には次のように宣言を行います．

   ```
   type : NETWORK
   ```

   記述例)

   ```
   /*    結合記述の宣言     */
   type : NETWORK;
   /*    セルタイプの宣言     */
   type : CELL;
   ```

2. cell

   モデルで使用するセルモジュールの宣言を行います．モジュール名とコンポーネント数を記述します．複数のセルモジュールを宣言するときは "," で継げます．

   記述例)

   ```
   /*    "HH"というモジュール名のセルモジュールを
         コンポーネント数 11 個分使用することを宣言      */
   cell :    HH[11];
   ```

3. synapse

   結合記述において，モデルで使用する化学シナプスモジュールの宣言を行います．モジュール名とコンポーネント数で記述します．複数のセルモジュールを宣言するときは "," で継げます．

   記述例)

```
/*    "SYN" というモジュール名の化学シナプスモジュールを
      コンポーネント数 5 個分使用することを宣言      */
synapse :    SYN[5];
```

4. gap

結合記述において，モデルで使用する電気シナプスモジュールの宣言を行います．モジュール名とコンポーネント数で記述します．複数のセルモジュールを宣言するときは "," で継げます．

記述例)

```
/*    "G" というモジュール名の電気スナプスモジュールを
      コンポーネント数 10 個分使用することを宣言      */
gap :    G[10];
```

5. input

モジュールの入力変数名を定義します．セルモジュールの入力変数の数と記述順序は後述する connection 文の結合関係式における記述と一致している必要があります．化学シナプスモジュール，電気シナプスモジュールの入力変数の数は 1 つです．複数のセルモジュールを宣言するときは "," で継げます．また，化学シナプス，電気シナプスの input 文には遅延情報を付加することができます．遅延情報は次の形で記述します．

```
input : 変数名 (遅延時間，初期値) ;
```

時刻 0 から遅延時間までの間は初期値が出力されます．

記述例)

```
/*    "Ig" を入力変数として定義     */
input :    Ig;
/*    "VOP" を入力変数として定義
      ただし，遅延時間 0.1 で初期値 0     */
input :    VOP(0.1,0);
```

6. output

モジュールの出力変数名を定義します．出力変数の数は 1 つです．

記述例)

```
/*    "V" を出力変数として定義     */
output :    V;
```

7. connection

コンポーネント同士の結合関係を記述します．これを結合関係式と呼びます．結合関係式は "<" をはさんで，右辺の出力が左辺に入力される構成になります．また，1 つの入力は "()" で囲むことで記述します．したがって，

```
mdl[i] < (input1)(input2);
```

という書式の場合，input1 がモジュール mdl のコンポーネント i の 1 番目の入力に，input2 がモジュール mdl のコンポーネント i の 2 番目の入力変数に代入されることになります．input 文で宣言されている変数の数と，結合関係式で表される入力の数は等しくなければいけません．つまり，この場合モジュール名 mdl の記述で，

```
input : V1, V2;
```

のように 2 つの入力変数が宣言されている必要があります．この時，V1 には input1 からの出力が，V2 には input2 からの出力が代入されます．

ところで，入力をとらない，つまり input 文が存在しないモジュールのコンポーネントへの入力は存在しないことになります．そこで，そのようなセルモジュールのコンポーネントの結合関係式は，次のような書式になります．

```
mdl[i] < ();
```

また，セルモジュールの入力は，化学または電気シナプスモジュールのコンポーネントを介している必要があります．したがって，セルモジュール mdl のコンポーネント i の 1 番目の入力に，セルモジュール mdl のコンポーネント j からの出力が化学 (または電気) シナプスモジュールのコンポーネント k を通って入る場合の書式は次のようになります．

```
mdl[i] < (syn[k] < mdl[j])(input2);
```

さらに，各モジュールのコンポーネントへの入力には，モジュールのコンポーネントからの出力を加算したものを指定することも可能です．この場合，"+" で継いで列挙します．例えば，セルモジュール mdl のコンポーネント n の出力がシナプスモジュール syn のコンポーネント m を介したものと，セルモジュール mdl のコンポーネント l の出力がシナプスモジュール syn のコンポーネント o を介したものが結合して，セルモジュール mdl のコンポーネント i の 2 番目の入力となっている場合の記述は次のようになります．

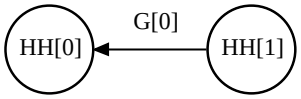
```
mdl[i] < (input1)(syn[m] < mdl[n] + syn[o] < mdl[l]);
```

結合関係式において右辺に現れるセルモジュールのコンポーネントは，別の結合関係式の最左辺に表れている必要があります．また，化学および電気シナプスモジュールのコンポーネントは単体では用いられず，入力側と出力側に必ずセルモジュールのコンポーネントを持つことに注意してください．

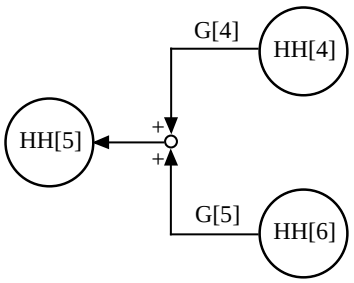
記述例)

```
connection:
/*     セルモジュール "HH" のコンポーネント 1 からの出力が,
       シナプスモジュール "G" のコンポーネント 0 を通って,
       セルモジュール "HH" のコンポーネント 0 に入れる.      */

HH[0] < (G[0] < HH[1]);
```

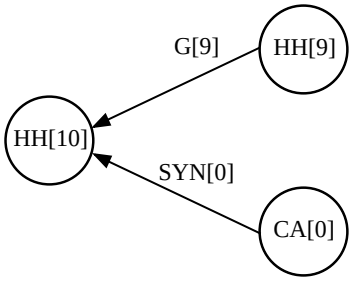



```
/*    セルモジュール "HH" のコンポーネント 4 からの出力が,
      シナプスモジュール "G" のコンポーネント 4 を通ったものと,
      セルモジュール "HH" のコンポーネント 6 からの出力が
      シナプスモジュール "G" のコンポーネント 5 を通ったものと結
      合され，セルモジュール "HH" のコンポーネント 5 に入力される.    */
HH[5] < (G[4] < HH[4] + G[5] < HH[6]);
```




```
/*    セルモジュール "HH" のコンポーネント 9 からの出力が,
      シナプスモジュール "G" のコンポーネント 9 を通って,
```

```
     セルモジュール "HH" のコンポーネント 10 の 1 番目の入力になり,
     セルモジュール "CA" のコンポーネント 0 からの出力が,
     シナプスモジュール "SYN" のコンポーネント 0 を通って,
     セルモジュール "HH" のコンポーネント 10 の 2 番目の入力になる.    */
HH[10] < (G[9] < HH[9])(SYN[0] < CA[0]);
```



●セルタイプモジュールの記述

このタイプはシングルモジュールで用いていたものと同じタイプです．セルタイプモジュールで用いられている文は次のようになっています.

1. type 文
2. module 文
3. exinput 文
4. input 文
5. output 文
6. observable 文
7. constant 文
8. parameter 文
9. function 文

3 から 8 は順不同です．また各文の詳細は 項 7.2.3.「モジュール記述」 および 項 7.4.4.「結合モデルにおけるモジュール記述」 を参照してください．なお，モジュール記述の終わりには必ず，"end;" をつけることを忘れないでください.

記述例)

```
/*    celll type module    */
type:        CELL;
module:      CL;
exinput:     Iex;
input:       Ig;
output:      V;
observable:  IK, Il, Ig, nK;
constant:    VK = -12.0, Vl = 10.6;
parameter:   Cm = 1.0, GK = 36.0, Gl = 0.3;
finction:
if(V! = 10.){
     an = 0.01 * (10. - V) / (exp((10. - V) / 10.) - 1);
}else{
     an = 0.01 * 10.;
```

```
}
bn = 0.125 * exp(-V / 80.);
dnK = an * (1. - nK) - bn * nK;
nK = integral(0.3177,dnK);
IK = GK * pow(nK,4.) * (V - VK);  /*  カリウム電流  */
Il = Gl * (V - Vl);               /*  リーク電流  */
Iall = Iex - IK - Il + Ig;
dV = Iall / Cm;
V = integral(0.,dV);              /*  膜電位  */
end;
```

●化学シナプスタイプモジュールの記述

化学シナプスタイプモジュールは入出力が 1 つのモジュールであり，セルタイプのモジュール同士を継ぐ役割を果たします．化学シナプスタイプと電気シナプスタイプの記述上の違いはありません．化学シナプスタイプモジュールで用いられている文と，その並びは次のようになっています．

1. type 文
2. module 文
3. input 文
4. output 文
5. observable 文
6. constant 文
7. parameter 文
8. function 文

3 から 7 は順不同です．また各文の詳細は 項 7.2.3.「モジュール記述」 を参照してください．なお，モジュール記述の終わりには必ず，"end;" をつけることを忘れないでください．

記述例)

```
/*    synapse type module    */
type:       SYNAPSE;
module:     GABA;
input:      P0(0.1,0);
output:     Tr;
parameter:  FB = 0.01;
function:
Tr = P0/ (FB + P0);
end;
```

●電気シナプスタイプモジュールの記述

化学シナプスタイプモジュールは入出力が 1 つのモジュールであり，セルタイプのモジュール同士を継ぐ役割を果たします．化学シナプスタイプと電気シナプスタイプの記述上の違いはありません．電気シナプスタイプモジュールで用いられている文と，その並びは次のようになっています．

1. type 文
2. module 文
3. input 文

4. output 文
5. observable 文
6. constant 文
7. parameter 文
8. function 文

3 から 7 は順不同です．また各文の詳細は 項 7.2.3.「モジュール記述」 を参照してください．なお，モジュール記述の終わりには必ず，"end;" をつけることを忘れないでください．

記述例)

```
/*    gap type module    */
type:        GAP;
module:      G;
input:       VOP(0.1,0);
output:      Ig;
parameter:   GL = 5.0;
function:
Ig = GL * (VOP - POSOUT);
end;
```

●結合記述

モデルで使用するモジュールを宣言し，各コンポーネントの結合関係を数式的に記述します．結合記述は，他のモジュール記述より先に，つまりモデルファイルの最初に記述するようにして下さい．結合記述で用いられている文と，その並びは次のようになっています．

1. type 文
2. module 文
3. cell 文
4. synapse 文
5. gap 文
6. connection 文

この文の並びは，絶対的なものですので，文を省略するのは構いませんが，順番を入れ替えないように注意してください．また各文の詳細は 項 7.2.3.「モジュール記述」 を参照してください．尚，結合記述の終わりには必ず "end;" をつけるのを忘れないでください．

結合記述には for 文が用意されており，大幅に記述を簡略化することができます．for 文の書式は次のようになっています．

書式)

```
for(式１ ; 式２ ; 式３){
      文
}
```

式 1，2，3 には，代入式，関係式，演算式などの式を 1 つ記述します．式 1 は初期式であり，for ループが始める前に参照されます．そして式 2 が満たされている間，文が実行されます．式 3 は for ループの 1 回のループ毎の最後に行われる式です．この for 文において文を囲む "{"，"}" は不要です．式 2 には，表 7.3.「条件式の関係演算子」 に示した関係演算子が使用できます．

記述例)

```
/*    Hodgkin-Huxley's cell model    */
type:       NETWORK;
module:     SQUID;
cell:       HH[11];
gap:        G[10];
connection:
         HH[0] < (G[0] < HH[0]);
         for( n = 1; n <= 9; n++ ) {
               HH[n] < (G[n-1] < HH[n-1] + G[n] < HH[n+1]);
         }
         HH[10] < (G[9] < HH[9]);
end;
```

7.4.5. 神経回路網のモデリング

ここでは，H-H モデルを 図 7.13.「神経回路モデルの例」 のように接続した神経回路網の記述を例に，そのモデリングについて説明しています.

細胞の結合を考える場合，隣接する細胞からの電流も考える必要があります.

まず，セルタイプモジュールを記述していきます．モジュールのタイプはセルタイプ，モジュール名は "HH" です.

```
type:         CELL;
module:       HH;
```

外部入力に注入電流 I_{ex} を，入力に隣接する細胞からの電流 I_g，そして膜電位 V を出力するものとします.

```
exinput:      Iex;
input:        Ig
output:       V;
```

ナトリウム電流の反転順位 I_{Na}，カリウム電流 I_K，漏れ電流 I_L，隣接する細胞からの電流 I_g を観測することにします.

```
observable:   INa, IK, Il, Ig;
```

各イオン電流の反転電位 E_{Na}, E_K, E_L を，定数として宣言します.

```
constant:     VNa = 115.0, VK = -12.0, Vl = 10.6;
```

膜容量 C_m，積分の初期値，各イオンの膜コンダクタンス $\bar{g}_{Na}, \bar{g}_K, \bar{g}_L$ をパラメーターとして宣言します.

```
parameter:    Cm = 1.0, mNa0 = 0.05293, GNa = 120., GK = 36.,
              Gl = 0.3, hNa0 = 0.5961, nK0 = 0.3177, V0 = 0.;
```

function 文には，前述した記述を用います.

細胞同士を結合する電気シナプスタイプモジュールの記述は次のようになります．電気シナプスタイプモジュールのモジュール名は "G" です.

```
type:         GAP;
module:       G;
```

電圧を入力とし，電流を出力とします．ただし，電圧は初期値が 0 で 0.1 秒の遅れを持って入力されます.

```
input:        VOP(0.1,0);
output:       Ig;
```

電気シナプスのコンダクタンスをパラメーターとして宣言します.

```
parameter:    GL = 5.0;
```

出力電流 I_g は 図 7.13.「神経回路モデルの例」 より次式で表されます.

$$I_g = g \times (V_1 - V_2)$$

ここで, V_1, V_2 は電気シナプス両端の電圧です. これより, function 文は次のように記述されます.

```
function:
              Ig = GL * (VOP - POSOUT);
```

最後に結合記述を考えます. このモデルの名前を "SQUID" とします.

```
type:         NETWORK;
module:       SQUID;
```

図 7.13.「神経回路モデルの例」 より, "HH" を 11 個と "G" を 10 個が必要ですので宣言します.

```
cell:         HH[11];
gap:          G[10];
```

これらの結合関係を for 文を用いて記述します.

```
connection:
         HH[0] < (G[0] < HH[0]);
         for( n = 1; n <= 9; n++ ) {
               HH[n] < (G[n-1] < HH[n-1] + G[n] < HH[n+1]);
         }
         HH[10] < (G[9] < HH[9]);
```

結合記述部をまとめると, 次のようになります.

記述例)

```
/*    Hodgkin-Huxley's cell model    */
type:         NETWORK;
module:       SQUID;
cell:         HH[11];
gap:          G[10];
connection:
         HH[0] < (G[0] < HH[1]);
         for( n = 1; n <= 9; n++ ) {
              HH[n] < (G[n-1] < HH[n-1] + G[n] < HH[n+1]);
         }
         HH[10] < (G[9] < HH[9]);
end;
```

結合記述は, モデル記述ファイルの先頭に必ず書かれていなければなりません.

3 つの 3 種類のモジュールと結合記述の記述例を全てまとめると, リスト 6 の NCS 言語による神経回路の結合モデル記述が完成します.

7.4.6. 結合モデルの使用法

作成したモデルファイルを使用し, シミュレーションを行う方法について述べます. ただし, 基本的な手順は 項 7.3.「NCS の使用法」 に記したものと同様ですので, ここでは簡単に示すことにします. 関数の書式の詳細などは 項 7.3.4.「シミュレーション条件の設定」 を参照してください.

さて, はじめにモデルファイルを登録し, プリプロセッサを起動します.

```
[]satellite[]~/home/demo:[25]% nassign("hh-netowrk")
[]satellite[]~/home/demo:[26]% npp()
```

または，nassign 関数を用いずに

```
[]satellite[]~/home/demo:[27]% npp("hh-network")
```

とすることにより，"hh-network.mdl" をモデルファイルとして登録して，プリプロセッサを起動することになります.

npp 関数実行後，nlink 関数により，C 言語ソースファイルをコンパイル後，NCS のライブラリとリンクされ，実行ファイルが作成されます.

```
[]satellite[]~/home/demo:[28]% nlink("-02")
```

npp 関数を実行し，シミュレーション条件ファイル群を生成したら，次に，各種シミュレーション条件を設定します.

時間条件は ntime 関数によって設定します.

```
[]satellite[]~/home/demo:[29]% ntime(10,0.001,0.01,1)
```

としたときには，計算時間を 10[ms]，計算刻みを 0.001[ms]，ストア刻みを 0.01[ms]，書き込み刻みを 1[ms] に設定したことになります.

外部入力条件は nstim 関数によって設定します．コンポーネント番号の指定をすれば，設定の範囲内で任意のコンポーネントに入力を行うことができます.

```
[]satellite[]~/home/demo:[30]% nstim("HH",10,"P",1,0,100,3,999)
```

とすることにより，モジュール名 "HH" のコンポーネント 10 に対してパルス関数を入力する設定を行います.

出力条件は nout 関数によって設定します．コンポーネント番号の指定により，任意のコンポーネントの出力を得ることができます.

```
[]satellite[]~/home/demo:[31]% nout(Iin,"HH",10,2 )
```

によって，モジュール名 "HH" のモジュールのコンポーネント 10 の入力値がバッファ Iin に出力されます．出力値を格納するバッファは nout 関数実行以前に series 型のバッファとして宣言されている必要があります.

また，複数のコンポーネントからの出力をまとめて配列に格納したい時などは，あらかじめ series 型の配列を宣言しておき，nout 関数の第一引数と第二引数との間に新たに配列番号を指定する引数を設けることで実現できます．例えば,

```
[]satellite[]~/home/demo:[32]% series V[3]
[]satellite[]~/home/demo:[33]% nout(V,0,"HH",0,1)
[]satellite[]~/home/demo:[34]% nout(V,2,"HH",5,1)
[]satellite[]~/home/demo:[35]% nout(V,3,"HH",10,1)
```

とすることによって，モジュール名 "HH" のモジュールのコンポーネント 0，5，10 の出力値が配列 V[0]，V[1]，V[2] に出力されます.

●遅延条件の変更

遅延条件は，ndelay 関数によって変更することができます．ndelay 関数の書式は次のようになっています.

書式) ndelay(mdl, var, dt [, init])

引数)

1. mdl : モジュール名

2. var : 内部変数名
3. dt : ディレイ時間
4. init : 出力の初期値

モジュール名 "G" の内部変数 "VOP" の遅延条件を変更する場合には次のようになります.

```
[]satellite[]~/home/demo:[36]% ndelay("G","VOP",0.25)
```

設定した外部入力条件は nsclist 関数の引数に "D" を用いて表示することができます.

```
[]satellite[]~/home/demo:[37]% nsclist("D")

DELAY

DELAY INFORMATION
   Data No.,  Input Name,  Delay Time,  Initial Output
     1        G(  VOP)       0.25            0
```

nsclist 関数を用いることによって，遅延条件が正しく設定されているか確認してください.

電気シナプスや化学シナプスに遅延条件を付加した場合，積分方式にはデフォルトである自動刻み積分法を用いることはできません．したがって，ninteg 関数により，ルンゲ - クッタ - ギル法またはオイラー法を指定する必要があります.

```
[]satellite[]~/home/demo:[38]% ninteg("R")
```

●シミュレーションの実行

シミュレーションを実行する前に，パラメータ値が正しく設定されているか nlist 関数で確認しましょう．引数に "ALL" を指定すると，全てのモジュールのパラメーター値を表示します.

```
[]satellite[]~/home/demo:[39]% nlist("ALL")

Parameter List

Module name :   HH
   Cm       =   [1]
   mNa0     =   [0.05293]
   GNa      =   [120]
   GK       =   [36]
   Gl       =   [0.3]
   hNa0     =   [0.5961]
   nK0      =   [0.3177]
   V0       =   [0]

Module name :   G
   Gl       =   [5]
```

リスト 7 : 神経回路網モデルのシミュレーション実行用のバッチファイル

```
1   nassign("hh-network");              # モデルファイルの登録
2   npp();                              # プリプロセッサの起動
3   nlink();                            # 実行ファイルの作成
4   ntime(10.0,0.001,0.01,1);           # 時間条件の設定
5   nstim("HH",10,"P",1,0,100,3,999);   # 外部入力条件の設定
6   series Iin, V[3];
7   nout(Iin,"HH",0,2);                 # 外部出力条件の設定
8   nout(V,0,"HH",0,1);
```

```
9   nout(V,1,"HH",5,1);
10  nout(V,2,"HH",10,1);
11  ndelay("G","VOP",0.25);              # 遅延条件の変更
12  ninteg("R");                         # 積分方式の変更
13  ncal();                              # シミュレーションの実行
```

さて，単一セルタイプモデルと同様に，各種シミュレーション条件を設定後，ncal 関数によりシミュレーションが実行されます.

```
[]satellite[]~/home/demo:[40]% ncal()
```

実行後，シェルはコマンド待ち状態になって，シミュレーションが終了したことを示します.

リスト 7 にファイル hh-network.mdl を使ったシミュレーションを行うためのバッチファイル例を示します. このファイルを "hh-network.sl" とするとシミュレーション実行は次のようになります.

```
[]satellite[]~/home/demo:[41]% inline("hh-network.sl")
```

シミュレーション結果は，システムモジュール GPM を使用することにより，次のようにしてグラフに表示することができます.

```
[]satellite[]~/home/demo:[42]% wopen(1,"A4",0,0)
[]satellite[]~/home/demo:[43]% time=(0~1000)/100
[]satellite[]~/home/demo:[44]% graph(V[0],time,0,0,0,0,0)
[]satellite[]~/home/demo:[45]% scale("N","D","N","D")
[]satellite[]~/home/demo:[46]% graph(V[1],time,0,0,0,0,0)
[]satellite[]~/home/demo:[47]% scale("N","D","N","D")
[]satellite[]~/home/demo:[48]% graph(V[2],time,0,0,0,0,0)
[]satellite[]~/home/demo:[49]% title(1,"time [ms]",
                                     "membrane potential [mV]")
[]satellite[]~/home/demo:[50]% axis(1,1,"XY","XY",5,0,0,0,0,0)
```

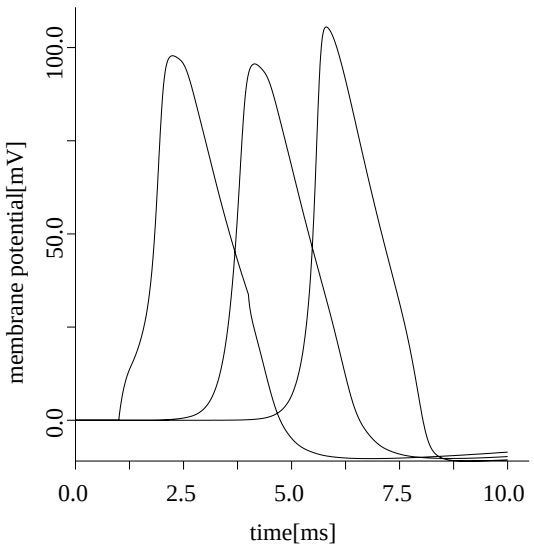


図 7.14. 神経回路網モデルのシミュレーション結果

グラフ表示の詳細は，SATELLITE モジュール GPM のマニュアルを参照してください.

第 8 章 非線形パラメータ推定システム

8.1. NPE の概要

NPE (Nonlinear Parameter Estimation module) は，最適化法を利用して最適化問題の解を求めるためのシステムです．NPE は，実験データからモデルパラメータを推定したり，要求された条件を満たすフィルタ，制御系，電気回路を設計するなど，モデリングにおいて大きな力を発揮します．また，NPE では，最適化法を利用して問題を解く際に必要となる付随的な作業を極力軽減し，最適化法のプログラミングから推定実行までを全て自動化しています．NPE の特徴としては，

- 統計学，数値解析の知識は特に必要なく，最適化法の特徴を知るだけで容易に利用できる．
- 非制約，制約条件付き最適化問題を扱うことができる．
- Simplex 法，BFGS 法，DFP 法，SSVN 法，共役勾配法といった代表的な最適化法を装備しており，用途に応じて選択することができる．
- プログラミング言語 (現在は，C 言語，C++ 言語をサポート) で記述されたモデル以外にも，SATELLITE に統合されている神経回路シミュレータ NCS を利用して記述されたモデルも扱うことができる．

などが挙げられます．

C 言語によるモデルを用いたパラメータ推定

NPE を利用したパラメータ推定は，(1) モデルの記述 (項 8.2.「モデル記述」)，(2) フィッティングに用いるデータファイルの準備 (項 8.3.「データファイルの準備」)，(3) 推定条件の設定 (項 8.4.「推定条件の設定」)，(4) 推定実行 (項 8.5.「NPE の実行」)，という手順で行われます．以下では，こうした手順に従ったパラメータの推定法を，

$$y_1(t) = A_1 \sin(2\pi t + B) + C \qquad (8.1)$$

$$y_2(t) = A_2 \cos(2\pi t + B) + C \qquad (8.2)$$

という 4 つのパラメータ (A_1, A_2, B, C) をもつ 2 出力 (y_1, y_2) モデルを例に説明していきます．

8.2. モデル記述

NPE を用いてパラメータ推定を行うためには，対象となるモデルを NPE の仕様に従って記述しておく必要があります．また，制約条件付きの推定を行う場合には，制約条件を設定するプログラムも決められた入出力関係を満たすように記述しなければなりません．本節では，NPE で扱うことのできるモデルと制約条件の記述形式について説明します．

8.2.1. C 言語によるモデル記述

NPE では，C 言語もしくは C++ 言語を用いてモデルを記述することができます．モデル関数では，時刻に対するデータインデックスおよびパラメータを受け取り，これらに基づいてモデル出力を計算するように記述します．このとき，モデル関数の引数は以下のように対応づけられます．

第 1 引数 : 時刻に対応するデータインデックス (int 型の変数)
第 2 引数 : 推定したいパラメータ (double 型の配列変数)
第 3 引数 : モデル出力 (double 型の配列変数)

第 2 引数のパラメータは，項 8.4.「推定条件の設定」 で説明する初期値設定関数 cinit() で宣言した順番で引き渡されますので，モデル記述の際はその対応に注意してください．また，第 3 引数のモデル出力もフィッティングに用いるデータファイルに格納されている順番に対応しているため，複数の出力を持つモデルを扱う場合には，配列変数に代入するモデル出力の順番に注意が必要です．

この形式に従って式 (8.1), (8.2) の例を記述したモデルファイルをリスト 8 に示します．例のように，モデル関数内で時刻が必要な場合は，リスト 8 の 14 行目にあるように get_sampling() 関数で SATELLITE で設定されているサンプリング周波数を利用できますので，point/samp を計算することによりデータインデックス point に対応する時刻をモデル関数内で利用することができます．

ここでは例として単純な関数を取り上げましたが，記憶・内部状態を持つ連続系モデルを扱いたい場合は，モデル関数内部で static 変数などを利用するか，次節で説明する NCS 言語を用いて記述してください．

```
1   /*  NPE USR モデルファイル (sin2.cpp)  */
2   /*  SIN の振幅, 位相, バイアス を求める  */
3   /*  COS の振幅, 位相, バイアス を求める  */
4   /*  2-出力の場合  */

5   #include <stdio.h>
6   #include <math.h>
7   #include <libsatellite.h>
8   #if !defined(PI)
9   #define PI 3.14159
10  #endif

11  int model(int point, double *param, double *output)
12  {
13      double y, theta, samp;
14      samp = get_sampling();
15      theta = 2.0 * PI * ((double)point / samp) + param[2];
16      y = param[0] * sin(theta) + param[3];
17      output[0] = y;
18      y = param[1] * cos(theta) + param[3];
19      output[1] = y;
20      return(0);
21  }
```

リスト 8 : C 言語によるモデルの記述例 (sin2.cpp)

8.2.2. NCS 言語によるモデル記述

NPE では，NCS 言語で記述されたモデルに対してもパラメータを推定できます．但し，この場合も C 言語でモデル記述した場合と同様に，モデルファイルをコンパイルしてオブジェクトファイルにしておく必要があります．NCS モデル (例 : sample.mdl) は，以下のコマンドを実行することによりコンパイルできます．

```
[]SATELLITE[]~/home/demo:[1]% npp ("sample")
[]SATELLITE[]~/home/demo:[2]% nlink ("-O2","O")
```

8.2.3. 制約条件の記述

NPE では，推定したいパラメータの取り得る範囲があらかじめわかっている場合，そうした情報をペナルティ関数として C 言語で記述しておくことで，制約条件付きの推定ができます．式 (8.1), (8.2) の例において，各パラメータの範囲を

$$A_1, A_2 < 5 \qquad (8.3)$$

$$\frac{\pi}{12} < B < \frac{\pi}{2} \qquad (8.4)$$

$$C > 0 \qquad (8.5)$$

と定めた場合，制約条件の記述はリスト 9 のようになります．ペナルティ関数では，C 言語で記述したモデル関数の第 2 引数と同様に，初期値設定関数 cinit() で宣言した順番でパラメータ値が格納された配列変数を引

数として受け取ります (リスト 9 の 6 行目の param)．また，制約条件に基づいて計算されたペナルティ (double 型) が関数の戻り値となります．この戻り値は，評価関数に加算されるため，結果的にパラメータ値が制約条件に反しないように推定が進行します．作成された制約条件プログラムは，モデルと同様，コンパイルしてオブジェクトファイルにしておく必要があります．

```
1   /*  2-出力モデル　制約条件プログラム (sin2_pena.cpp)  */

2   #include <stdio.h>
3   #include <math.h>
4   #ifdef __cplusplus

5   #define  Pena_Val   1000.0

6   double penalty( double param )
7   {
8       double pena = 0.0;
9       if( param[0]<=5. )        pena += Pena_Val;
10      if( param[1]<=5. )        pena += Pena_Val;
11      if( param[2]=<M_PI/12. )  pena += Pena_Val;
12      if( param[2]<=M_PI/2. )   pena += Pena_Val;
13      if( param[3]<=0 )         pena += Pena_Val;
14      return(pena);
15  }
```

リスト 9 : 制約条件の記述例 (sin2_pena.cpp)

8.3. データファイルの準備

NPE では，実験などで得られたデータとモデル出力との誤差で表される評価関数が最小になるようにパラメータを推定します．そのためには，推定を実行する前に，フィッティングに用いるデータ (以下，推定用データと呼ぶ) を格納したファイルを用意しておく必要があります．また，評価関数に任意の重み付けをする場合は，重みデータを格納したファイルも用意する必要があります．両者とも SATELLITE のファイル形式で，モデル出力と同じレコード数のデータとして用意します．ファイルには，レコード 0 からデータを格納します．例えば，モデルの出力が 2 の場合には，

```
[]SATELLITE[]~/home/demo:[3]% data1 = $"data.dat":[0]
[]SATELLITE[]~/home/demo:[4]% data2 = $"data.dat":[1]
```

という操作により，出力 1，2 のデータが，それぞれ，data1，data2 に取り出せるような形式のデータファイルとして用意しておく必要があります．

ここでは，リスト 8 に示した 2 出力のモデルにおいて，各パラメータの真値を $A_1 = 2, A_2 = 3, B = \frac{\pi}{8}, C = 2$ と定め，適当な雑音を重畳させたものを実験データの代わりとして SATELLITE 上で作成し，ファイルとして保存する手続きを説明します．データポイントは 1000 点，サンプリング周波数は 1kHz とします．すなわち，1 秒間のデータファイルが作成されることになります．重みデータは，簡単なものにするため，単純に全て 1 としておきます．リスト 10 に，これらのデータファイルを作成するためのバッチファイルを示します．このバッチにより作成されたデータは，図 8.1.「推定用データ例」 のようになります．黒色が式 (8.1) ならびにリスト 8 の 16 行目に記述したサイン関数，灰色が式 (8.2) ならびにリスト 8 の 18 行目に記述したコサイン関数につての推定用データです．バッチファイル内における各 SATELLITE コマンドの使用方法に関しては，リファレンスマニュアルならびにユーザーズマニュアルの該当項目を参照してください．

```
1   const dpt = 1000;
2   const A1  = 2;
3   const A2  = 3;
4   const B_  = PI/8;
5   const C_  = 2;

6   series y[dpt],wgt[dpt];
```

```
7   # sin,cos
8   t = (0~(dpt - 1)) / dpt;
9   y:[0] = A1 * sin(2 * PI * t + B_) + C_;
10  y:[1] = A2 * cos(2 * PI * t + B_) + C_;

11  # d
12  ran0 = nrand(dpt,1,0,0.01);
13  ran1 = nrand(dpt,3,0,0.01);

14  # f
15  y:[0] = y:[0] + ran0;
16  y:[1] = y:[1] + ran1;

17  #
18  wgt:[0] = (0~(dpt - 1)) * 0 + 1;
19  wgt:[1] = wgt:[0];

20  #
21  $"sin2.dat" = y;
22  $"sin2.wgt" = wgt;
```

リスト 10 : データファイルの作成用バッチファイル (sin2_mkdata.sl)

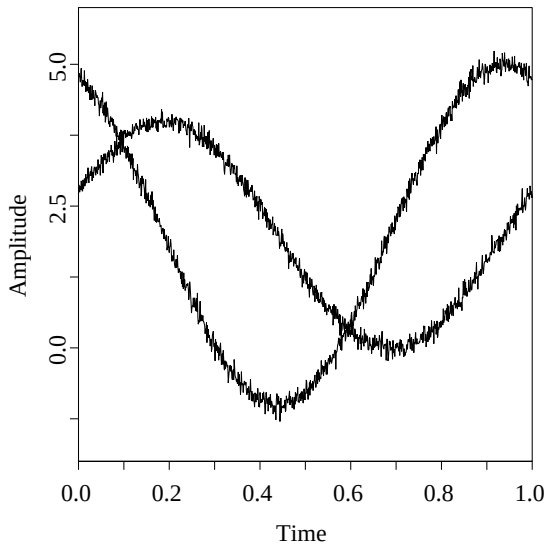


図 8.1. 推定用データ例

以上の例では，推定用データを SATELLITE で作成しましたが，実際には，各々の実験環境において測定されたデータのフォーマットを，SATELLITE データに変換する処理が必要になります．各データ点が，スペース，あるいは，タブ，カンマ，改行で区切られ，アスキー形式で保存されているデータは，SATELLITE のユーティリティモジュールに登録されている text2buffer() を用いることにより，簡単に SATELLITE データに変換できます．例えば，リスト 8 に示したモデルのパラメータ推定に利用したい実験データが，

```
2.845, 4.746, 2.875, 4.845, 2.751, 4.808, 2.747, 4.779 ...
```

のように，ファイル "data.txt" に合計 2000 点格納されており，奇数番目のデータをモデル出力 1，偶数番目のデータをモデル出力 2 の推定用データとして利用したい場合には，

```
[]SATELLITE[]~/home/demo:[5]% data = text2buffer("data.txt")
[]SATELLITE[]~/home/demo:[6]% data
```

```
[0]:%   2.845   4.746   2.875   4.845   2.751
[5]:%   4.808   2.747   4.779   ...
[]SATELLITE[]~/home/demo:[7]% data = trans(reform(data,(1000,2)))
[]SATELLITE[]~/home/demo:[8]% data:[0]
[0]:%   2.845   2.875   2.751   2.747   ...
[]SATELLITE[]~/home/demo:[9]% data:[1]
[0]:%   4.746   4.845   4.808   4.779   ...
[]SATELLITE[]~/home/demo:[10]% $"data.dat" = data
```

とすれば，推定に必要なデータファイルを作成できます (章 9. *データ変換システム*)．ただし，実験データがバイナリ形式でファイルに保存されている場合には，そのフォーマットを解読した上で SATELLITE データに変換するプログラムを独自に作成しなければなりません．

8.4. 推定条件の設定

NPE では，モデルのパラメータ推定を行う前に，表 8.1.「推定条件の設定コマンド」 に示すような各種の推定条件を設定する必要があります．このうち，(1) 〜 (9) は必ず設定しなければならないものです．

表 8.1. 推定条件の設定コマンド

推定条件	設定コマンド
(1) 最適化アルゴリズム	cmethod()，clsearch()
(2) 推定モデル	cmodel()
(3) 推定の制約条件	cpenalty()
(4) 推定パラメータ	cinit()
(5) スケーリング法	cscale()
(6) 推定の停止条件	clogic()，cterm()
(7) 推定モデルの出力数	cnumber()
(8) 推定用データ	cpoint()，cdata()，cweight()
(9) 推定結果の格納	creault()，chistory()
(10) 表示する評価関数値の計算方法	cdisp()
(11) 評価関数の計算方法	cnorm()

表 8.1.「推定条件の設定コマンド」 の右側は，NPE で各推定条件を設定するための関数一覧です．各関数の引数の説明は，リファレンスマニュアルを参照してください．以下では，最低限設定しなければならない推定条件について解説します．

1. 最適化アルゴリズム…cmethod()，clsearch()

   非線形最適化法に用いる最適化アルゴリズムを指定します．NPE には，最適化アルゴリズムとして，Simplex 法，BFGS (Broyden-Fletcher-Goldfarb-Shnno) 法，DFP (Davideon-Fletcher-Powell) 法，SSVM (Self-Scaling Variables Metric) 法，共役勾配法 (Fletcher-Reeves の更新式)，共役勾配法 (Polak-Ribiere-Polyak の更新式) の 6 種類が用意されています．Simplex 法以外の最適化アルゴリズムでは，直線上で評価関数値を最小にする一次元探索法を指定する必要があります．一次元探索法には，黄金分割法と三次補間法が用意されています．

2. 推定モデル…cmodel()

   推定モデルを指定します．モデルが C 言語で記述されている場合には，第 2 引数にモデルのファイル名を指定します．NCS 言語で記述されたモデルの場合には，オブジェクトファイル名を指定します．

3. 推定の制約条件…cpenalty()

   推定の制約条件を記述したペナルティ関数のオブジェクトファイルを指定します．

4. 推定パラメータ…cinit()

   推定するパラメータの初期値などを指定します.

5. スケーリング法…cscale()

   スケーリング法を指定します．推定すべきパラメータのオーダーが不揃いの場合，収束するまでのアルゴリズムや一次元探索の反復回数が増加し，推定時間が増大していまいます．こうした問題に対して，スケーリングでは，推定前に書くパラメータに適当な係数を乗じてパラメータのオーダーを揃える操作を行い，推定速度の向上を図ります．推定終了後には，逆の操作によってパラメータを元のオーダーに戻します.

6. 推定の停止条件…clogic()，cterm()

   推定の停止条件を指定します．停止条件には，最大反復回数，評価関数値，Simplex 法の評価関数値の標準偏差が設定できます．また，clogic() は，必ず cterm() の前に実行しておく必要があります.

7. 推定モデルの出力数…cnumber()

   推定モデルの出力数を指定します．この関数で指定した数は，推定用データや重みデータを格納しているファイルのレコード数と一致している必要があります.

8. 推定用データ…cdata()，cweight()，cpoint()

   推定に用いるデータファイルを指定します．cdata()，cweight() では，それぞれ推定用データ，重みデータを格納したファイル名を指定します．また，cpoint() ではデータファイル中のデータ点数を指定します.

9. 推定結果の格納…cresult()，chistory()

   推定結果を格納するファイルを指定します．cresult() ではモデル出力の履歴を，chistory() でパラメータ・評価関数値の履歴を格納するファイルをそれぞれ指定します.

このような推定条件をリスト 8 のモデルに対して設定する SATELLITE のバッチファイルの例 (sin2_com.sl) をリスト 11 に示します．ここでは，最適化法として経験的に優れているといわれている BFGS 法を指定し，一次元探索法には黄金分割法を選択しています．また，スケーリングは無しに設定しています.

こうしたバッチファイルは SATELLITE のシェルコマンドに登録されている inline() により,

```
[]SATELLITE[]tom//home/demo:[11]% inline("sin2_com.sl")
```

とすることで実行できます．また，一度設定した推定条件は，関数 cstore() を用いることで

```
[]SATELLITE[]tom//home/demo:[12]% cstore("sin2.prm")
```

のようにファイル (sin2.prm) に格納することもできます．このパラメータファイルを関数 cload() により読み込んで推定条件を同様に設定することができます.

```
1   ## NPE 推定条件スクリプト(sin2_com.sl) (USR モデル)

2   # 最適化アルゴリズム
3   cmethod("bfgs");                # "BFGS 法"を設定
4   clsearch("golden",0.00001);     # 直線探索法として"黄金分割法"を設定

5   # 推定モデル
6   cmodel("USR","sin2.cpp");       # C 言語のモデル"sin2.cpp"を設定

7   # 推定の制約条件
8   cpenalty("sin2_pena.cpp");      # ペナルティ関数"sin2_pena.cpp"を設定

9   # 推定パラメータ
10  cinit(4,1,1.8,"VAR","A1",0.000001); # "A1"の初期値を"1.8"に設定
```

```
11  cinit(4,2,2.8,"VAR","A2",0.000001); # "A2"の初期値を"2.8"に設定
12  cinit(4,3,-0.1,"VAR","B",0.000001); # "B"の初期値を"-0.1"に設定
13  cinit(4,4,3.0,"VAR","C",0.000001);  # "C"の初期値を"3.0"に設定

14  # スケーリング法
15  cscale(0);                  # "スケーリングなし"に設定

16  # 推定の停止条件
17  clogic("0|1");              # 条件に"最大反復回数または評価関数値"を設定
18  cterm(0,100);               # 最大反復回数 ≧ 100
19  cterm(1,20);                # 評価関数値 ≦ 20

20  # 推定モデルの出力数
21  cnumber(2);                 # 出力数"2"に設定

22  # 推定データ
23  cpoint(1000);               # データ点数を"1000"に設定
24  cdata("sin2.dat");          # 実験データ"sin2.dat"を設定
25  cweight("sin2.wgt");        # 評価関数の重みデータ"sin2.wgt"を設定

26  # 推定結果の格納
27  cresult("sin2_res.dat");        # モデル出力の履歴を格納するファイルを

28  # "sin2_res.dat"に設定
29  chistory("sin2_hist.dat",10);  # パラメータ・評価関数値の履歴を格納する
30                                 # ファイルを"sin2_hist.dat"に設定

31  ## ※以下のコマンドは必要に応じて設定する

32  # 表示する評価関数値の計算方法
33  cdisp(0);                   # 表示する評価関数値の計算を"重みなし"に設定

34  # 評価関数の計算方法
35  cnorm(2);                   # 評価関数の計算方法を"2-ノルム"に設定
```

リスト 11 : 推定条件の設定バッチファイル例 (sin2_com.sl)

8.5. NPE の実行

ここまでの説明で，パラメータ推定を実行するために必要な，モデル (sin2.cpp)，制約条件 (sin2_pena.cpp)，推定用 (sin2.dat) ならびに重みデータ (sin2.wgt) を格納したファイルの作成 (sin2_mkdata.sl) や，推定条件の設定 (sin2_com.sl) が完了しました．こうした準備が間違いなくできていれば，以下のように推定実行関数 npe() を実行するだけで，自動的にパラメータ推定が行われます (sin2_est.sl) .

```
[]SATELLITE[]~/home/demo:[13]% series result,hist
[]SATELLITE[]~/home/demo:[14]% npe(result,hist)
Making Executable file is done.

[   NPE]   BFGS

      <<< 0   Step:   Error = 3495.14  Penalty = 1000  >>>
          A1          A2          B           C
          1.8         2.8         -0.1        3

      ................................................
```

```
<<< 1   Step:   Error = 799.14   Penalty = 0  >>>
    A1          A2           B            C
1.79589     2.78321      0.52453      2.55808
```

関数 npe() の第 1 引数 (result) には，最終的に推定されたパラメータでのモデル出力，第 2 引数 (hist) には，推定されたパラメータと誤差の履歴が格納されます．これらの内容は，推定条件設定関数 cresult()，chistory() で指定したファイルに格納されているデータと同じです．パラメータの推定状況をみると，B には制約条件を超える処理値 (-0.1) を設定したので，初期状態で "Penalty=1000" とペナルティが課せられていることがわかります．しかし，1 回目の推定後には，制約条件内に全てのパラメータが更新され，ペナルティが取り消されています．

このパラメータ推定は，

```
<<< 11  Step:   Error = 799.14   Penalty = 0  >>>
    A1          A2           B            C
1.97965     2.97731      0.39562       1.9979

Error < 20
    3 points are stored in "sin2_hist.dat"
Estimation is done.
```

と，推定条件で設定したように，誤差が 20 以下になると 11 回の更新で終了します．パラメータの推定後に，

```
[]SATELLITE[]~/home/demo:[15]% hist
[0]:[0]%   1.8     2.8     -0.1     3       3495
[1]:[0]%   1.969   2.967   0.3864   1.997   20.44
```

と入力すると，パラメータの履歴が確認できます．最後の行に表示されているのが最終的に推定されたパラメータです．左から，A_1, A_2, B, C，誤差の順に表示されています．推定用データを生成するときに用いた値は，それぞれ，$A_1 = 2, A_2 = 3, B = \frac{\pi}{8}, C = 2$でしたので，真の値に近いパラメータが推定されていることがわかります．図 8.2.「推定結果例」 に，推定されたパラメータを用いたモデル出力 (白色) と，推定用データ (黒色) の重ね書きを示します．パラメータ推定後のモデル出力が，推定用データとほとんど重なっており，パラメータが良好に推定されていることがわかります．

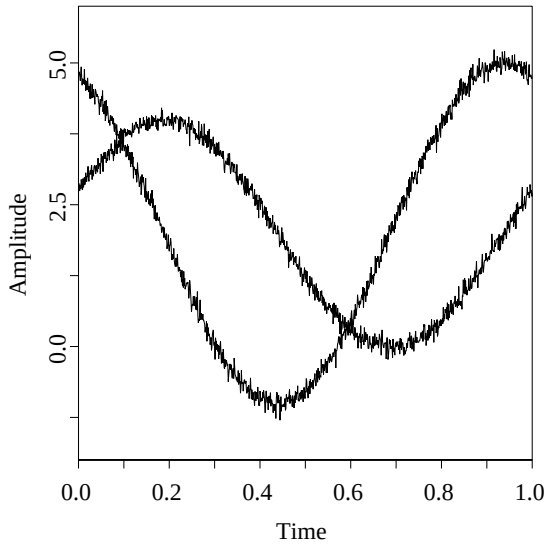


図 8.2. 推定結果例

8.6. NCS 言語を用いたパラメータ推定

NPE では，NCS 言語で記述されてモデルに対してもパラメータを推定できます．以下，Hodgkin-Huxley 型モデルによるスパイク応答波形から，モデルのナトリウム電流コンダクタンス ($\overline{g_{Na}}$) と膜容量 (C_m) を推定する

ために NPE を適用した例をしめします．なお，NCS 言語に関しては 章 7. *神経回路シミュレータ* を参考にしてください．

- モデル記述

  NCS モデルには 7 章のリスト 3 : NCS 言語による Hodgkin-Huxley モデルを用います．このモデルを保存したファイルを hh.mdl とします．

- データ作成

  実践的には実験データを NPE で扱えるデータフォーマットに変換して使用しますが，ここでは，NCS で計算したシミュレーション結果を NPE の教師データとして利用することにします (リスト)．NCS で HH モデルをシミュレーションし，結果をファイルオブジェクトに保存します．重みファイルも作成します．また，2-3 行目で NCS モデルをコンパイルしていますが，3 行目の第 2 引数に注意してください．NCS モデルを NPE で使用する場合には，C 言語モデルとは異なり，モデルのオブジェクトファイルを用意する必要があります．nlink 関数の第 2 引数に "O" オプションを渡すことにより，オブジェクトファイルの生成を行います．

```
1   nasgn("hh");
2   npp();
3   nlink("-O2","O");
4   ntime(10,0.01,0.01,1);
5   nstim("HH",0,"P",1,0,100,3,999);
6   series V;
7   nout(V,"HH",0,1);
8   ninteg("F");
9   ncal();
10  datapt = length(V);
11  series vd[datapt];
12  vd:[0] = V;
13  wgt:[0] = (1~(length(V))) * 0 + 1;
14  $"hh.dat" = vd;
15  $"hh.wgt" = wgt;
```

リスト 12 : 教師データ作成スクリプト (NCS によるシミュレーション : hh_mkdata.sl)

- ペナルティ関数

  C 言語のモデルのときと同じように，推定するパラメータに対する制約条件を設定することができます．ここでは C++ 言語で制約条件を設定することにします．推定するパラメータである GNa は 0 から 1000 の間，Cm は 0 から 10 の間であるという制約条件の場合には次のリスト 13 のようなファイルを作成します．

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <math.h>
4   #define lambda  0.1

5   #ifdef __cplusplus
6   extern "C" {
7   #endif

8   double penalty(double *param)
9   {
10     double ret=0.0;
11     if(param[0] < 0. ) ret += param[0]*param[0];
12     if(param[0] < 1000.) ret += (param[0]-1000.)*(param[0]-1000.);
13     if(param[1] < 0. ) ret += param[1]*param[1];
14     if(param[1] < 10.) ret += (param[0]-10.)*(param[0]-10.);

15     return(ret * lambda);
```

```
16  }
17  #ifdef __cplusplus
18  }
19  #endif
```

- NPE 条件設定

  C 言語のモデルのときと同じように，パラメータを推定するときの NPE の設定条件をリスト 14 に示します.

```
1   cmethod("bfgs");
2   clsearch("golden",0.00001);
3   cmodel("NCS","hh.o");
4   cpenalty("hh_pena.cpp");
5   cinit(2,1,140.0,"VAR","GNa@HH",0.000001);
6   cinit(2,2,0.1,"VAR","Cm@HH",0.000001);
7   cscale(3);
8   clogic("0|1");
9   cterm(0,20);
10  cterm(1,0.0001);
11  cnumber(1);
12  cdata("hh.dat");
13  cweight("hh.wgt");
14  cresult("hh_res.dat");
15  chistory("hh_hist.dat",10);
16  cdisp(0);
17  cnorm(2);
```

  リスト : 14 Hodgkin-Huxley モデルパラメータ推定のための設定条件 (hh_com.sl)

  C 言語を用いた場合との相違点をまとめると以下のようになります.

  - モデル指定

    3 行目
    cmodel ("NCS","hh.o")
    第 1 引数は，C 言語のモデルの場合には "USER" でしたが，NCS 言語の場合には "NCS" になります．また，第 2 引数には nlink 関数で作成したオブジェクトファイルを指定します.

  - パラメータの初期値設定

    5，6 行目
    cinit(2, 1, 140.0, "VAR", "GNa@HH", 0.000001);
    cinit(2, 2, 0.1, "VAR", "Cm@HH", 0.000001);
    推定するパラメータを指定する cinit の 5 番目の引数の書式は NCS のパラメータ名 @NCS のモジュール名となります.

- 推定の実行

  ここまで推定に必要なモデル，データ，NPE 条件ファイルがそろいましたので，あとは NPE を実行するだけです．推定のためのスクリプトをリスト 15 に示します．NCS モデルを使ったパラメータ推定の場合，NPE が連続系のシミュレーションを NCS と協調して行うわけですが，そのため，npe() 関数を実行する前に，シミュレーション条件を推定したいデータを取得した条件に合わせて設定する必要があります (リスト 15 : 1 から 8 行目).

```
1   nasgn("hh");
2   npp();
3   nlink("-O2");
4   ntime(10,0.01,0.01,1);
5   nstim("HH",0,"P",1,0,100,3,999);
6   series V;
```

```
7   nout(V,"HH",0,1);
8   ninteg("F");
9   inline("hh_com.sl");
10  cpoint(length($"hh.dat":[0]);
11  series res,hi;
12  npe(res,hi);
```

リスト 15 : 推定実行スクリプト (hh_est.sl)

第 9 章 データ変換システム

9.1. DCM の概要

DCM (Data Conversion Module) は，様々なデータ解析ソフトウェアやシミュレータ間でデータを共有するためのシステムです．このシステムでは，次に示すソフトウェアのデータをサポートしています．

- AXON pClamp の ABF 形式
- AVS のフィールドデータ形式
- テキスト形式ファイル
- MATLAB の Matrix 形式
- Mathematica のリスト形式
- Genesis の disk_out オブジェクト出力ファイル
- Neuron の出力ファイル
- TEAC 製ディジタルレコーダ DR-M2a / M3a の出力ファイル

9.2. Excel との連携

9.2.1. SATELLITE ファイルを Excel で読む

例として，項 7.4.「NCS による神経回路網のモデリングシミュレーション」 で使用したサンプル (hhmodelE.sl) を使用します．

シミュレーションを実行し，Excel で読むファイルを作る手続きは，次のようになります．

```
[]SATELLITE[]~/home/demo:[1]% inline("hhmodelE.sl")
[]SATELLITE[]~/home/demo:[2]% V1=V[0]
[]SATELLITE[]~/home/demo:[3]% V2=V[1]
[]SATELLITE[]~/home/demo:[4]% V3=V[2]
[]SATELLITE[]~/home/demo:[5]% buffer2text("hhE.txt"," ,%8.4f ,%8.4f ,
                                %8.4f\n",V1,V2,V3)
```

これで，Excel で読むことができるファイルが作成され，Excel からファイル hhE.txt をテキスト形式 (カンマ区切り形式) で読んだ結果は，次のようになります．

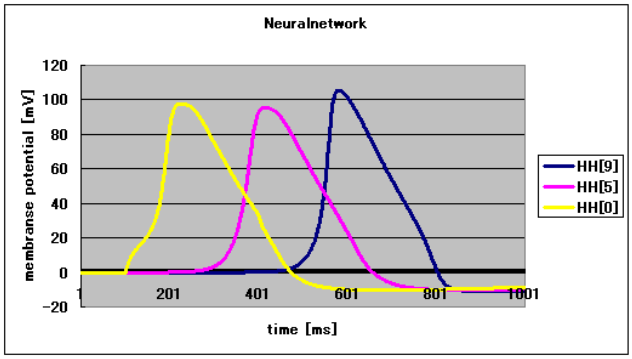


図 9.1. SATELLITE ファイルを Excel で読んだ結果

9.2.2. Excel ファイルを SATELLITE で読む

式 (1) で作成されたデータを 図 9.2.「式 (9.1) を Excel で実行した結果」 に示します.

$$t = 10^{\frac{i}{1000}+0.44} \qquad (9.1)$$

$$x = \sin(t)$$

但し，B 列が t，C 列が x を表しています.

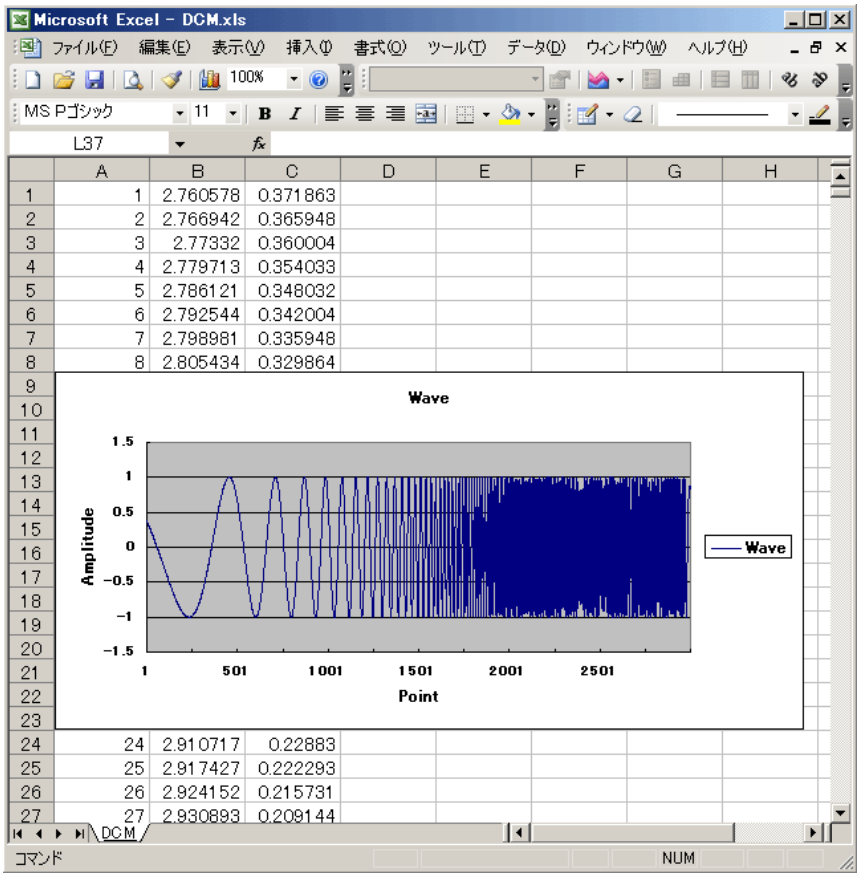


図 9.2. 式 (9.1) を Excel で実行した結果

Excel でタブ付きテキスト形式で保存すれば，SATELLITE 変数に代入することができます．SATELLITE で，このテキスト形式ファイルを読む例を次に示します.

```
[]SATELLITE[]~/home/demo:[6]% x=text2buffer("sweep.txt")
[]SATELLITE[]~/home/demo:[7]% index(x)
[0]:%   3000   3
[]SATELLITE[]~/home/demo:[8]%

[]SATELLITE[]~/home/demo:[9]% wopen(1,"A4",0,1)
[]SATELLITE[]~/home/demo:[10]% size(120,80)
[]SATELLITE[]~/home/demo:[11]% color("black","black")
[]SATELLITE[]~/home/demo:[12]% lwidth(1,1)
[]SATELLITE[]~/home/demo:[13]% origin(40,40)
[]SATELLITE[]~/home/demo:[14]% scale("N","F","N","F",0,3000,-1.2,1.2)
[]SATELLITE[]~/home/demo:[15]% graph(x[2],"T",0,0,0,0,0)
[]SATELLITE[]~/home/demo:[16]% title(1,"point","Amplitude")
[]SATELLITE[]~/home/demo:[17]% axis(1,1,"XY","XY",5,0,0,0,0,0)
[]SATELLITE[]~/home/demo:[18]% frame()
```

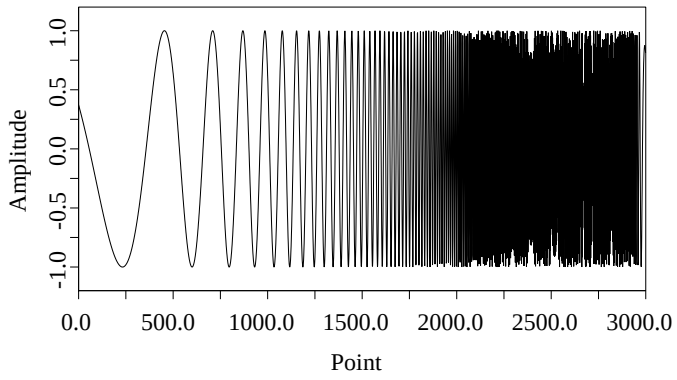


図 9.3. 実行結果

9.3. MATLAB との連携

9.3.1. SATELLITE ファイルを MATLAB で読む

例として，項 7.4.「NCS による神経回路網のモデリングシミュレーション」 で使用したサンプル (hhmodelE.sl) を使用します.

シミュレーションを実行し，MATLAB で読むファイルを作る手続きは，次のようになります.

```
[]SATELLITE[]~/home/demo:[19]% inline("hhmodelE.sl")
[]SATELLITE[]~/home/demo:[20]% buffer2matlab("V.mat","V",V)
```

これで，MATLAB で読むことができるファイルが作成されます．MATLAB では，次のように関数を入力することで，Workspace に buffer2matlab 関数で指定した変数名で取り込むことができます.

```
>>load V.mat
```

取り込んだ結果をプロットするには，次のように入力します.

```
>>VT=V'
>>plot(VT)
```

結果を 図 9.4.「実行結果」 に示します.

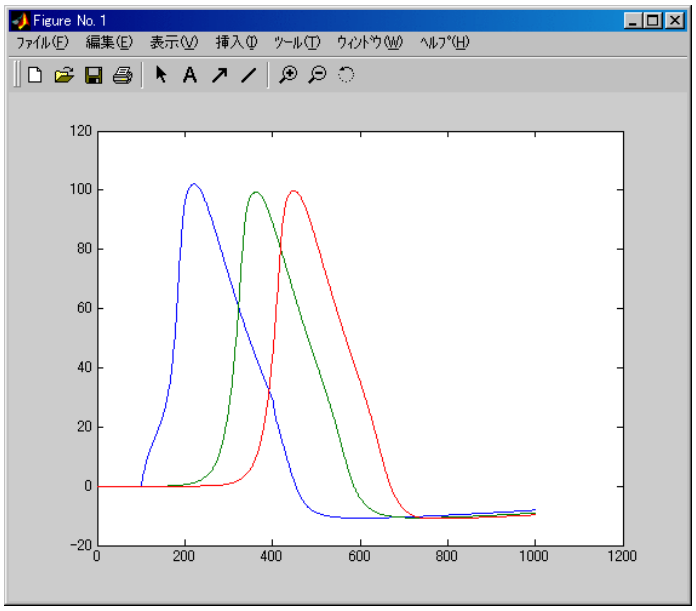


図 9.4. 実行結果

9.3.2. MATLAB ファイルを SATELLITE ファイルに変換する

```
>>num=[4 5]
>>den=[1 3 2]
>>[k,p]=residue(num,den)
>>t=0:0.01:10
>>y=k(1)*exp(p(1)*t)+k(2)*exp(p(2)*t)
>>plot(t,y)
>>xlabel('time [s]'); ylabel('y(t)'); grid
```

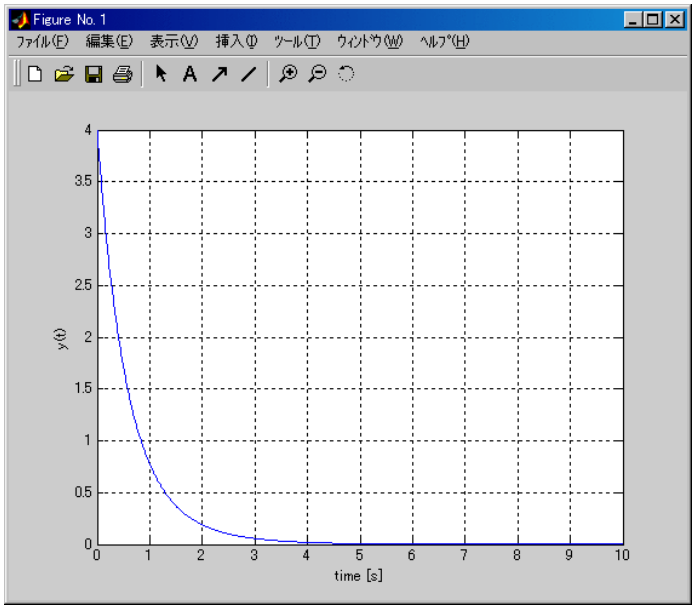


図 9.5. MATLAB 画面

```
>>save time.dat t
>>save outp.dat y
```

```
[]SATELLITE[]~/home/demo:[21]% matlab2satellite("time.dat")
[]SATELLITE[]~/home/demo:[22]% time = $"t"
[]SATELLITE[]~/home/demo:[23]% matlab2satellite("outp.dat")
[]SATELLITE[]~/home/demo:[24]% outp = $"y"
[]SATELLITE[]~/home/demo:[25]% wopen(1,"A4",0,1)
[]SATELLITE[]~/home/demo:[26]% color("black","black")
[]SATELLITE[]~/home/demo:[27]% sx = 80
[]SATELLITE[]~/home/demo:[28]% sy = 80
[]SATELLITE[]~/home/demo:[29]% origin(40,40)
[]SATELLITE[]~/home/demo:[30]% size(sx,sy)
[]SATELLITE[]~/home/demo:[31]% title(1,"time [s]","y(t)")
[]SATELLITE[]~/home/demo:[32]% graph(yout,tim,0,0,0,0,0)
[]SATELLITE[]~/home/demo:[33]% axis(1,1,"XY","XY",5,0,0,0,0,0)
[]SATELLITE[]~/home/demo:[34]% ltype(2,1)
[]SATELLITE[]~/home/demo:[35]% line(0,sy / 2,sx,sy / 2)
[]SATELLITE[]~/home/demo:[36]% ltype(1,1)
[]SATELLITE[]~/home/demo:[37]% frame()
```

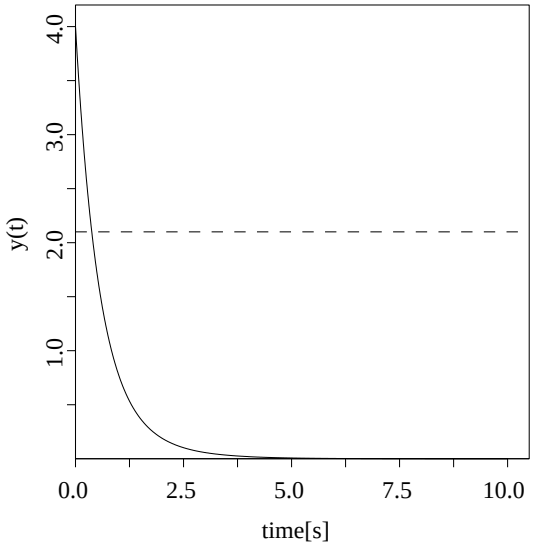


図 9.6. 実行結果