まえがき

数理物理学 II は，数理物理学 I で学習した "**微分方程式による物理現象のモデル化とその解析**" を，計算機シミュレーションで確認するという目的を持った講義です．シミュレーションを行うには計算機に手順を指示するため，「プログラミング」は欠かせませんから，前年度までは全能といってもよい **C 言語**を使っていました．しかし，C 言語の習得は意外に難しいらしく，学生は，単にテキストの指示通りにプログラムを打ち込むのみで，コンピュータに自分の考えを伝えるという本来の目的を達成することがほとんどできませんでした．

そこで，今年度からは C 言語をあきらめ（しかし，これは究極のツールですから，挑戦してくれることを願ってやみません），数式をノートに記述する感覚で進めていける高級（人間の言葉に近いという意味です，したがって低級とはコンピュータ寄りという意味になります）数値解析ツールを使うことにしました．

現在，そのようなツールは数多く存在し，中でも数式処理システム *Mathematica* が有名です．本学ではサイトライセンス契約を結んでいるので備え付けのコンピュータでは MS-Windows，Linux どちらの OS の上でも使うことができます．しかし，残念ながら学生個人が所有するノートパソコンにはサイトライセンスの範囲内ではインストールすることができません．もちろん，*Mathematica* は購入に値する優れたツールですから，教科書 10 冊分程度の投資を今すれば運用によっては生涯に渡って大きな知的財産を得ることができますが，全ての学生が財産を築けるというものでもありません．

とにかくも，理工系の場合，ワードプロセッサや表計算に加えてこれに類するツールを一通り使ってみることはたいへん意義のあることと思いますので，個人所有のノートパソコンにもインストールが可能なフリーの数値解析ツール octave を使うことに決めました．

教科書・参考書

- **早野龍五・高橋忠幸：「計算物理」(共立出版)**． 教科書です．物理的な背景を理解するには，この書を熟読する必要があります．この書籍では，プログラミング言語として Fortran を用いています．Fortran は科学技術計算分野では今でも主流ですし，C 言語と同じく万能の汎用言語です．この講義で科学技術計算に興味を持ち，もっと深くこの分野の学習を続けるならば，Fortran のソースを読む必要が生ずるでしょう．
- `http://ayapin.film.s.dendai.ac.jp/~matuda/TeX/lecture.html`： 全講義テキスト (PostScript)．
- **E．クライツィグ：技術者のための高等数学５「数値解析」（培風館）**． 数理物理学 I でも紹介した，優れた教科書．
- **松田七美男：「Linux 活用術」(東京電機大学出版局)**．計算機の OS は MS Win-NT ではなく UNIX 互換の Linux を用いる．多少の慣れが必要なので，この書に限らず，UNIX の基本操作に関する書籍を手許に置いておくと良い．
- **Matlab 関する入門書**． 残念ながら Octave に関する書籍は発刊されていません．雑誌にはいくつか小かい記事があります．実は，Octave は Matlab という優れた数値解析ツールとの互換性が高い（完全ではありません）ことでも有名です．したがって，Matlab の入門書は Octave 自身を理解する上でも大変役立ちます．

目 次

目次へ

今年度は教材を書き直していますから，内容が変更になることが予想されます．したがって，目次は確定したものではありません．

目次へ

第 I 部
数値解析ツール Octave

1 基本的な操作

1.1 起動と終了

まずは使ってみましょう．"octave"（実行ファイル名）とキー入力してください．すると著作権に関するメッセージの後に**プロンプト**が表示されて命令待ちの状態になります．プロンプトに含まれる番号は，**履歴番号 (history number)** と呼ばれています．一般に命令をその都度解釈して実行させる動作状態は**対話 (interactive) モード**と呼ばれます．対話モードでは，命令が長く複雑な場合，以前入力した命令を呼び出したくなる場合があります，そのためには命令を履歴として記憶し，またいつでも呼び出せるように番号づけする必要があります．一般に対話モードで動作するツールには，この履歴機能は必需品といえます．

```
GNU Octave, version 2.1.36 (i686-pc-linux-gnu).
...(中略)

Report bugs to <bug-octave@bevo.che.wisc.edu>.

octave:1>  <-- プロンプト
```

終了する命令は，**quit** あるいは **exit** です．

1.2 初めての操作

命令は**数式をノートに記述する感覚**で使えます．例えば，sin(1) の計算値が欲しい場合は，単に "sin(1)" と入力すればよくて，答えが ans = に並んで表示されます．

```
octave:1> sin(1)
ans = 0.84147  <-- 計算が実行された
octave:2> x
error: 'x' undefined near line 2 column 1
octave:2>  <-- 実行できなかったので履歴番号が 2 のまま
```

さて，変数 x を使うために単に x と入力するとどうなるでしょうか？**定義されていない**というエラーメッセージが表示されます．プログラミング言語では，変数は一般にあらかじめ **定義（宣言）** されなければなりません．C 言語などでは型（整数，実数，配列）なども指定する必要があって大変面倒であることは，何度も体験してますね．ところが，octave では変数は代入時に型が自動的に判別されますから，非常に楽になっています．変数 x（の宣言とともに）に値 $\tan(\pi/5)$ を代入するには x = tan(pi/5) と記します．

```
octave:2> x=tan(pi/5)
x = 0.72654
octave:3>
```

ここで，pi は π の値を持った定数であらかじめ定義されたものです．そのような**組み込み（built-in）** 定数 (constants) を表示させましょう．

```
octave:3> e
e = 2.7183
octave:4> i
i = 0 + 1i
octave:5> true
true = 1
octave:6> false
false = 0
octave:7> realmax
realmax =  1.7977e+308
octave:8> realmin
realmin =  2.2251e-308
```

最後の 2 つは，扱える実数の最大値と最小値です．数学では実数に範囲はありませんし精度 (precision) も無限ですが，計算機シミュレーションでは有限の値しか扱えませんし，もっと重要なことは**精度が限られている**ことです．したがって，計算を行う際には扱える実数の範囲を越えないことや，意味のある数値（有効数字）が得られるように工夫しなければなりません．octave は実際には C 言語で書かれており，数値は内部では**倍精度実数 (double precision)** として扱われます．C 言語では，実数は**浮動小数点数 (floating point)** として表現され，その有効桁は小数点以下 16 です．octave では表示桁精度（組み込み変数 output_precision）を変更することができますが，16 桁以降が有効でない（しかしなんらかの数値が表示されてしまう）場合もありますから注意してください．ちょっと次のようにして試してみましょう．

```
octave:9> output_precision =28
output_precision = 28
octave:10> 1.000000000000001-1.0
ans =  1.1102230246251565404236316680e-15
octave:11> 1.0000000000000001-1.0
ans = 0
octave:12> acos(0)
ans =  1.5707963267948966557998981734e+00
       1.5707963267948966192313216916 <-- 正しい値
```

3 番目は $\arccos(0) = \pi/2$ ですが，参考までに真の値も並べておきましたから比較してください．さて，2 番目の例では小さい数が 0 となってしまいました．この 0 はみかけの 0 ではなく本当に 0 です．計算結果としては，これも人間にとって親切と思うのですが，**数値ではない**という NaN(Not a Number) や**無限大** Inf(Infinity) もきちんと表示してくれます．

最後に，数の演算について説明しましょう．基本的にはノートに記すようにすればよいのですが，**四則演算**のうち

目次へ

乗法（かけ算）は，記号 “*” (アスタリスク，asterisk) を使います．この記法はほとんど全てのプログラム言語で共通です．また，**冪乗**を記号 “** あるいは ^” で表すことができます．表 1 にまとめます．

表 1　数に関する演算

表記	数式	意味
x+y	$x+y$	x と y の和
x-y	$x-y$	x と y の差
x*y	xy	x と y の積
x/y	x/y	x を y で割った商
rem(x,y)		x を y で割った余り
x**y, x^y	x^y	x の y 乗

問題 1　次の数値や関数値を計算してみなさい．
(i) $\sin \pi/6$, $\tan \pi/6$, $\arctan \pi/4$, $\sinh(-1)$
(ii) $\log e$, $\log_{10} 5$, e^3, $e^{-\pi}$
(iii) $\sin(0)/0$, $\tan(0)/\sin(0)$, $\cos(0)/\sin(0)$

history の使い方　history を使うと，命令の入力が楽になります．例えば，上カーソルキーによって，1 つ前の命令が呼び出されるという機能は他のツールにも見られる常識的なものです．命令を打ち込んで積極的に使ってみましょう．まず，単に “history” と入力すると，履歴に残っている（通常 ~/.octave_hist という名前のファイルに保存されます）命令が番号付きで表示されます．“history 5 ” とすると遡って 5 つまでの命令が一覧表示されます．一覧表示をみて，履歴番号 N の命令を再び実行させるには，

```
run_history 履歴番号
```

とします．以下の具体的例を参考にしてください．

```
octave:1> x=1.0; y=2.0;
octave:2> x+y
ans = 3
octave:4> history
    1 x=1.0; y=2.0;
    2 x+y
    3 history
octave:4> x=10;
octave:5> run_history 2 <-- 番号 2 の命令の再実行
ans = 12
```

セミコロンの役割　前の具体例で 1 行に複数の命令を分けるためにセミコロン “;” を使いました．すると，結果が表示されなくなりました．命令の区切りには，コロン “,” を使うこともできますが，その場合には結果が表示されてしまいます．

問題 2　次の等式が正しいことを 3 つ以上の場合に対して確かめてみなさい．

(i) $(x+y)^2 = x^2 + 2xy + y^2$
(ii) $\cos(\theta - \phi) = \cos\theta\cos\phi + \sin\theta\sin\phi$
(iii) $\sinh(x+y) = \sinh(x)\cosh(y) + \cosh(x)\sinh(y)$
(iv) $\arctan\dfrac{1}{x} = \arcsin\sqrt{\dfrac{1}{1+x^2}}$
(v) $|x| \ll 1$ のとき $(1+x)^\alpha \simeq 1 + \alpha x$

2　複素数

複素数が簡単に扱えることも特徴のひとつです．虚数単位 $i = \sqrt{-1}$ は，定数 i，j，I，J として組み込まれています．

```
oct> i,j,I,J
i = 0 + 1i
j = 0 + 1i
I = 0 + 1i
J = 0 + 1i
```

したがって，

```
3+4i, 3+4*i, 3+i*4, 3+4J, 3+4*J, ...
```

などと表記ができます．octave 自身は結果の表示に “数字 i” を用いていますから，それを使いましょう．というのも，もし “数字*i” という表記を使うと，これは数字と変数 i の積と解釈されてしまい，場合によっては重大な間違いとなるからです．その場合とは **組み込み定数 i, j, I, J と同じ名前の変数を使ってしまった場合**です．octave は新しい定義あるいは代入結果を優先してしまいます．例えば

```
oct> i=40
i = 40
oct> 3+4*i
ans = 163
oct> 3+4i
ans = 3 + 4i
```

となってしまいます．したがって**積の記号 * を使わない表記を使いましょう**．

ところが虚数部に変数を使う場合には，yi と表記すると変数名と解釈されてしまうので，

```
x+y*i, x+i*y, x+y*J, x+J*y, ...
```

と記述せざるを得ません．もし組み込み定数 i, j, I, J と同じ名前の変数を使ってしまっていたら，上記のような深刻な間違いが起こります．このような事態を回避するには，**組み込み定数と同じ名前の変数を使わない** のが一番でしょう．

四則演算は実数と同じ表記ができます．複素数に特有の，実数部 (real part)，虚数部 (imaginary part)，偏角 (argument)，共役 (conjunction) を表示させてみましょう．

目次へ

```
oct> z=3+4i
z = 3 + 4i
oct> real(z), imag(z), arg(z), conj(z)
ans = 3
ans = 4
ans = 0.92730
ans = 3 - 4i
oct> tan(arg(z))
ans = 1.3333  <-- 4/3 に等しい
oct> z^2
ans =  -7 + 24i
```

表 2 に複素数に関する主な演算をまとめます.

表 2　複素数に関する演算

表記	数式	意味
`real(z)`	$\mathrm{Re}[z] = \dfrac{z+z^*}{2}$	z の実数部
`imag(z)`	$\mathrm{Im}[z] = \dfrac{z-z^*}{2i}$	z の虚数部
`conj(z)`	z^* または $\bar{z}$	z の複素共役
`arg(z)`	$\mathrm{Arg}(z) = \arctan \dfrac{\mathrm{Im}[z]}{\mathrm{Re}[z]}$	z の偏角
`abs(z)`	$\lvert z\rvert = \sqrt{z^* z}$	z の絶対値

問題 3　次の複素数 (関数) 値を計算してみなさい．なお，$J_n(z)$ は n 次の第 1 種 Bessel 関数で，octave では `besselj(n,z)` という名前で組み込まれています.

(i) $(1+i)^3, \sqrt{i}, (-1+i)^i, i^{-1+i}$

(ii) $\sin(1+i)$, $\tanh(1-i)$, e^{-1+i}, $\log(i)$

(iii) $J_0(1-i)$, $J_1(1+i)$

初期設定ファイル　この節の実行例ではプロンプトが短くなっていることに気づきましたか．実は，ユーザーの要求に応じて変更できる事柄があって，プロンプトも "`PS1="`*strings*`"`" によって *strings* にいつでも変更できます．octave は起動時に個人用の設定ファイルを読み込みます．そのファイルは，通常 `~/.octaverc` です．以下に具体例を示します.

```
PS1="oct> "
output_precision=6
PI=pi
```

問題 4　以下の等式が正しいことを 3 つ以上の複素数を代入して確認しなさい.

(i) $|z| = \sqrt{z^* z}$, $\mathrm{Re}[z] = \dfrac{z+z^*}{2}$

(ii) $e^{iz} = \cos z + i \sin z$, $e^{\log z} = z$

(iii) $\cos z = \dfrac{e^{iz} + e^{-iz}}{2}$

(iv) $\cos^2 z + \sin^2 z = 1$, $\cosh^2 z - \sinh^2 z = 1$

(v) $\sin(z_1 + z_2) = \sin z_1 \cos z_2 + \cos z_1 \sin z_2$

(vi) $\sin(iz) = i \sinh(z)$, $\cos(iz) = \cosh(z)$

3　行列とベクトル

数学では，スカラー（自然数，整数，実数，複素数）からベクトル，行列へと「数」の扱いを拡げて学習してきました．行列を学習し終えた後は，スカラーを 1×1，ベクトル (行ベクトル，列ベクトル) を $1 \times n$ あるいは $m \times 1$ の行列として扱うことに抵抗はなくなっていると思います．すると，基本演算を行列に対して定義すれば全ての「数」を統一的に扱えるという見通しがでます．このような背景のもと octave は行列に対する演算を基本において全体が構成されています．もちろんノートに記すように扱えることに変わりはありません.

3.1　行列の生成

ノートに記す場合は，大きな括弧で両側を括りますが，さすがに複数行にわたる括弧をキー入力するのは手間なので，鍵括弧 "`[` と `]`" で括ります．また，行内の成分は空白あるいはカンマで分け，行の終わりにはセミコロン ";" を用います.

```
oct> A=[1 2 3; 4 5 6]
A =
  1  2  3
  4  5  6

oct> A(1,3), A(1,:), A(2,2:3)
ans = 3
ans =
  1  2  3

ans =
  5  6
```

生成された行列の各成分 A_{ij} は，**範囲指定子** ":" を使って指定することができますから，かなり柔軟に扱えます．ところで，上の具体例で生成した行列 A の大きさは 2×3 です．この行列の `(3,3)` 要素を指定することはできるでしょうか？ 一般のプログラミング言語では，**(配列の定義) 範囲を越えたアクセス**は，メモリ管理上大きな問題を惹き起こす可能性があり，最も行ってはならない間違いの一つとされています．しかし octave ではそれが問題を起こすことなくできます．実際,

目次へ

```
oct> A(3,3)=1+2i
A =
  1 + 0i  2 + 0i  3 + 0i
  4 + 0i  5 + 0i  6 + 0i
  0 + 0i  0 + 0i  1 + 2i
```

と，複素数を代入すると行列も複素数行列になります．したがって，1000×1000 の行列の確保が次のように簡単にできます．

```
oct> C(1000,1000)=0;  <-- ; をつけ忘れると大変...
```

ここでも，C 言語などに比べてとっつきの良いインターフェースを持ち合わせていることが判ります．また，列数や行数が一致すれば，行列を並べて行列を生成することができます．

```
oct> a=[1 2 3]; b=[9 8 7]; [a b], [a; b]
ans =
  1  2  3  9  8  7

ans =
  1  2  3
  9  8  7
```

3.1.1 範囲指定子

範囲指定は等差数列を要素とする数の集合を得る演算で，"始点:差分:終点" のように記述します．差分を省略すると 1 と解釈されます．言葉の説明では判りにくいでしょうが，具体例をみれば納得します．

```
oct> 1:0.2:2
ans =
  1.0000  1.2000  1.4000  1.6000  1.8000  2.0000

oct> 1:5
ans =
  1  2  3  4  5

oct> 1:-0.3:0
ans =
  1.00000  0.70000  0.40000  0.10000
```

実は，octave にはこの範囲のための型 (range) が設けてあって，行列とは区別されていますから，行列の演算ができない場合があります．鍵括弧で括れば，行列に変換されます．

3.1.2 linspace, logspace

範囲は差分を指定するので最後の要素が終点の値とならない場合があります．それに対して，linspace と logspace は指定された要素数（分割数は 1 少ない）の行列を生成します．π などを端にする場合にはこちらが便利でしょう．

```
oct> linspace(0,pi,5)
ans =
  0.00000  0.78540  1.57080  2.35619  3.14159

oct> logspace(0,1,5)
ans =
   1.0000   1.7783   3.1623   5.6234  10.0000
```

3.1.3 特殊な行列

上記以外にも行列を生成する方法は多々あって，あらかじめ特別な行列を得るための関数があります．主要なものを表 3にまとめておきます．

表 3 特殊行列を生成する関数

関数名	生成される行列
zeros(N,M), zeros(N)	全成分が 0 * ．零行列
ones(N,M), ones(N)	全成分が 1 * ．定数行列
eye(N,M), eye(N)	対角成分が 1 他が 0 * ．単位行列
rand(N,M), rand(N), rand("seed",x)	全成分が $(0,1)$ の一様擬似乱数 * "seed" を指定して x で初期化
randn(N,M), randn(N), randn("seed",x)	全成分が正規分布乱数 * 初期化については rand() と同じ
hilb(N)	N 次 Hilbert 行列 $A_{ij} = \frac{1}{i+j-1}$
invhilb(N)	N 次 Hilbert 行列の逆行列

* $N \times M$ あるいは N 次正方.

3.2 行列・ベクトルの演算

数学で習ったままで記述できるという特徴はそのままです．しかし，数値的に解析する（プログラムを記述する）上で便利な特有の演算がありますのでそれに注意しつつ説明しましょう．

3.2.1 逆行列，転置行列

行列 A の逆行列 A^{-1} は残念ながらノートに書くようにはいかず

```
C=inv(A)：逆行列 C = A^-1
```

と表記しなければなりません．転置行列 A^T は関数ではなく記号を用いて簡便に表記できますが，注意が必要です．複素数行列に基本をおいているので，**複素共役をとった上での転置と単なる転置を区別**しなければなりません．

目次へ

C=A' ：複素共役転置 $C = \tilde{A}^T \quad \left[C_{ij} = \tilde{A}_{ji}\right]$
C=A.'：転置 $C = A^T \quad \left[C_{ij} = A_{ji}\right]$

転置は列ベクトルと行ベクトルの変換で多用されます．前節で行ベクトルを生成する命令をあげました．数学ではむしろ列ベクトルに対して左から行列をかけるという方式（これは行ベクトルに右から行列をかけるとしても全く同様に議論できるはず）で学習してきたので，列ベクトルをすぐに生成できないことに少し戸惑いがあります．まあ，行ベクトルを転置して列ベクトルを得るという習慣になれるしかなさそうです．

```
oct> a=[1:0.5:2]; b=linspace(0,pi,3);
oct> [a' b' sin(b')]
ans =
  1.00000  0.00000  0.00000
  1.50000  1.57080  1.00000
  2.00000  3.14159  0.00000
```

3.2.2 四則演算

同じ大きさの行列 A, B 同士の加減は以下のように記述すれば，常識通りに解釈され計算結果が出ます．

```
oct> A=ones(2); B=3*eye(2);
oct> A+B, A-B, A*B
ans =
  4  1     <-- | 1 1 | + | 3 0 |
  1  4         | 1 1 |   | 0 3 |

ans =
  -2   1   <-- | 1 1 | - | 3 0 |
   1  -2       | 1 1 |   | 0 3 |
```

乗算も常識通り左側の列数と右側の行数が等しい場合，すなわち A（$m \times l$ 行列）と B（$l \times n$ 行列）の行列について $C = AB$（$m \times n$ 行列となる）が計算されます．行列の除算は数学では習いません．しかし，スカラー演算では**逆数をかける**として定義されます．これを行列にも適用すれば，行列 B を 行列 A で割るとは，素直には

$$B \div A \ \rightarrow \ BA^{-1}$$

となりそうです．ところでこの式をみると左から A^{-1} をかけてみたくなります．もちろん行列の乗算では一般に交換すると計算結果が異なります．

一般に $BA^{-1} \neq A^{-1}B$

すなわち，どちらから逆行列をかけるかを区別する必要があります．このような理由で数学では聞き慣れない**左除算，右除算**が octave には存在します．

B/A：右除算
A\B：左除算

実は，逆行列は数値計算すると値が不安定になる場合があり，逆行列を求めずに以下の行列方程式の解として除算の結果を算出していることを注意しておきます．

B/A=X は B=XA の解
A\B=X は B=AX の解

順番が逆になりましたが，スカラー a と行列 A の間の四則演算はどうなっているでしょう．スカラーを 1×1 行列ととらえると演算はできないことになりますね．しかし，それはもちろん非常識で，数学で習った通り乗除に関しては**行列のスカラー倍** $aA = (aA_{ij})$ が計算されます．残るは加減ですが，数学では計算できないことになっています．ところが，octave では，A と同じ大きさで成分が全て a である行列との加減を計算してしまうのです．

```
oct> a=1;A=eye(3); a+A, a*ones(3)+A
ans =
  2  1  1
  1  2  1
  1  1  2

ans =
  2  1  1
  1  2  1
  1  1  2
```

数式ではそのような計算をすることはないのですが，行列を計算する場合にはこの表記が簡便と感ずる場合があります．

問題 5 次の行列を生成してみなさい．
(i) 5 次の単位行列
(ii) 成分が全て π である 7×3 行列
(iii) 区間 $[-5\ 5]$ を 10 等分する点を成分とする列ベクトル
(iv) $A_{jk} = jk^2$ である 7×3 行列
(v) $A_{jk} = \sin\left(\frac{j\pi}{2}\right)\cos\left(\frac{k\pi}{4}\right)$ である 8×4 行列

3.2.3 配列演算：成分毎の演算

最後になりましたが，数学では習わない数値解析ツール特有の演算を説明します．それは配列演算と呼ばれ，行列の成分毎に演算を作用させるというものです．行列の和と差はまさに成分毎の結果ですが，それを乗除にさらには関数の作用にまで拡大します．すなわち

目次へ

C=A.+B：成分毎の和 $C_{ij} = A_{ij} + B_{ij}$
C=A.-B：成分毎の差 $C_{ij} = A_{ij} - B_{ij}$
C=A.*B：成分毎の積 $C_{ij} = A_{ij}B_{ij}$
C=A./B：成分毎の商 $C_{ij} = A_{ij}/B_{ij}$
C=A.^p：成分毎の冪 $C_{ij} = A_{ij}^p$
C=f(A)：成分毎の関数値 $C_{ij} = f(A_{ij})$

となります．これは行列やベクトルの方程式をスカラーと同様に表記するという目的をかなえる上で大変重要な演算です．この考え方に慣れることが必要です．

なお，正方行列 A に関しては行列自身の冪 A^p が定義できますから，スカラーと同様にして指数関数も

$$e^C = 1 + C + \frac{C^2}{2!} + \frac{C^3}{3!} + \cdots + \frac{C^k}{k!} + \cdots$$

のように定義されます．そこで，関数名の最後に m を付けて成分毎の指数と区別します．

C=exp(A)： $C_{ij} = \exp(C_{ij})$
C=expm(A)： $C_{ij} = \exp(C)_{ij}$

このような関数は他に行列対数 logm(A)，行列平方根 sqrtm(A) があります．

例題 1 $x_k = k\pi/10$, $y_k = k\pi/20$ $(k = 0, 1, 2, \cdots, 10)$ として，$A_k = \sin(x_k)\cos(y_k)$ を成分とする 10 次元の横ベクトルを生成しなさい．

[解] 普通のプログラミング言語の感覚で for ループを使って

```
for k=0:10
  A(k)=sin(k*pi/10)*cos(k*pi/20);
endfor
```

とすることもできますが，せっかくですから配列演算を用いて

```
x=linspace(0,pi,11);
y=linspace(0,pi/2,11);
A=sin(x).*cos(y);
```

とするのが，octave らしい方法です．

4 グラフィック

計算結果をすぐにプロットして画面上で確認できることは，数値解析ツールの大きな利点です．一般にグラフィックはハードウェア依存の割合が高く，汎用のライブラリを構築することは非常に厄介なものです．そこで Octave は外部の汎用プロッティングツール gnuplot を呼び出し，描画を任せています．Octave は，欧州では Mathmatica よりも流布しているといわれる数値解析ツール Matlab に互換性があります．グラフィック関数も当然 Matlab 互換となっていて，単純なものを作成するには事足りています．

ところが，PostScript 形式のファイルを作成したり，3 次元の色付きのプロットを作成するには，gnuplot の命令を直接使う必要があります．そこでまずは，Matlab 互換の関数を用いて単純なプロットを作成する方法を説明し，続いて Matlab 互換でない gnuplot の命令を使って細かい指定をする方法に言及します．

4.1 2 次元プロット：plot

二つのベクトルデータ v1,v2 を x, y 軸にして 2 次元プロットを描く関数は以下のように呼び出します．

```
plot(v1,v2,options)
```

ここで，*options* には線種や記号の種類や色，データの名前などを指定できます．ガウス型関数 $G(x) = e^{-x^2}$ を描く具体例を示します（図 1）．

```
oct> x=linspace(-3,3,100);
oct> G=exp(-x.^2);
oct> plot(x,G);
```

x，G はデータ数の等しいベクトルであることに注意してください．

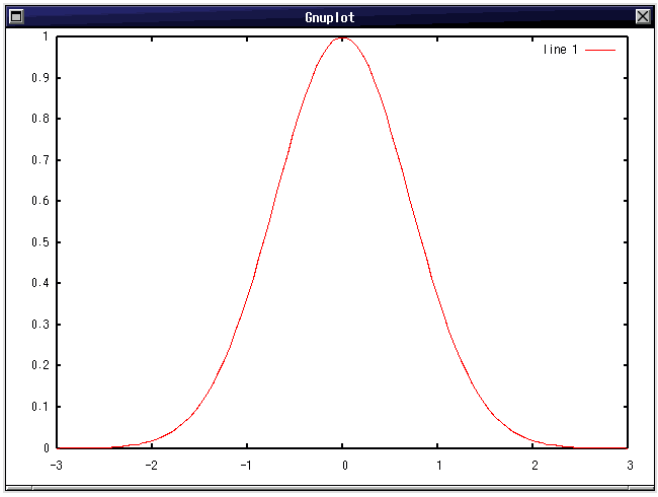

図 1 plot 命令による Gauss 型関数の表示．X11 の画面を画像として取り込んだもの．

4.1.1 対数軸，平面極座標

対数目盛でプロットする場合には，plot に変えて以下の命令を使います．

$\log x - y$： semilogx(...)
$x - \log y$： semilogy(...)
$\log x - \log y$： loglog(...)

また，極座標系の関数形 $r = f(\theta)$ のデータとして曲線をプロットするには，polar(...) を用います．

目次へ

4.2 複数のプロット

複数のプロットを描くには，幾つか方法があります．

4.2.1 plot の引数による

まず，以下のように，plot() 関数に複数のデータ組みを与えることができます．

```
plot(v1,v2,options, u1,u2,options, ...)
```

例えば，前述のガウス型関数にローレンツ型関数 $\frac{1}{1+x^2}$ も加えて描くには以下のようにします（図 2）．

```
oct> x=linspace(-3,3,100);
oct> x2=2*x;
oct> G=exp(-x.^2);
oct> L=1./(1+x2.^2);
oct> plot(x,G,";Gauss;",x2,F,";Lorentz;");
```

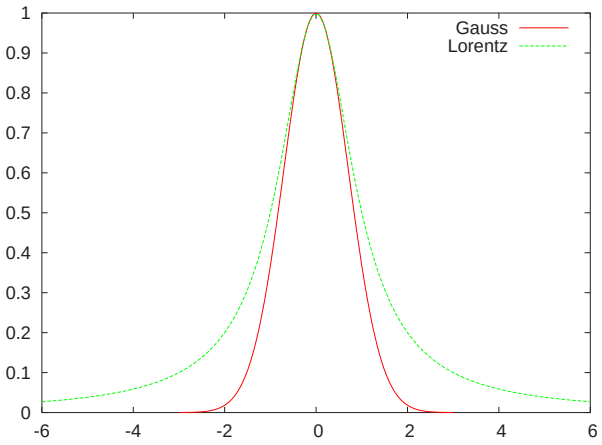

図 2　plot 命令による 2 つの関数の同時描画．印刷用に PostScript 形式で作成したものなので，X11 の画面とは少し異なります．

4.2.2 hold on 命令による

重ね書きを指示する命令が hold on です．通常，初期状態は hold off なので，plot() を実行するたびに，前のプロットは消去されます．いわゆる**上書き**モードです．すなわち，次のように実行すればよいことになります．

```
oct> x=linspace(-3,3,100);
oct> G=exp(-x.^2);
oct> plot(x,G,";Gauss;");
oct> hold on  <-- 重ね書きの指示
oct> x2=2*x;
oct> L=1./(1+x2.^2);
oct> plot(x2,F,";Lorentz;");
```

1 番目と 2 番目の描画にやや間がありますが，図 2と同じプロットが得られます．

4.2.3 subplot による

上記 2 つの方法は，1 つの枠の中に描くものでした．subplot(m,n,k) は 1 つのページを m 行 $\times n$ 列に 分割して k 番目の枠の中に描く命令です．

```
subplot(m,n,1); plot();
subplot(m,n,2); plot();
...
```

以下のような例で確かめてください．

リスト 1　subplot2d.m

```
 1: x=linspace(-3,3,100);
 2: x2=2*x;
 3: G=exp(-x.^2);
 4: L=1./(1+x2.^2);
 5: gset term postscript eps color "Helvetica" 14
 6: gset out "subplot2d.eps"
 7: subplot(2,2,1)
 8: plot(x,G,";Gauss;");
 9: subplot(2,2,2)
10: plot(x2,L,";Lorentz;");
11: subplot(2,2,3)
12: plot(x,G,";Gauss;",x2,L,";Lorentz;");
13: subplot(2,2,4)
14: plot(x,3*G,";Gauss;");
15: hold on
16: plot(x2,2*L+1,";Lorentz;");
```

この例では 2 次元プロットだけですが，枠は大きさが小さくなるだけでそれぞれ独立していますから，3 次元 plot を混在させることもできます．

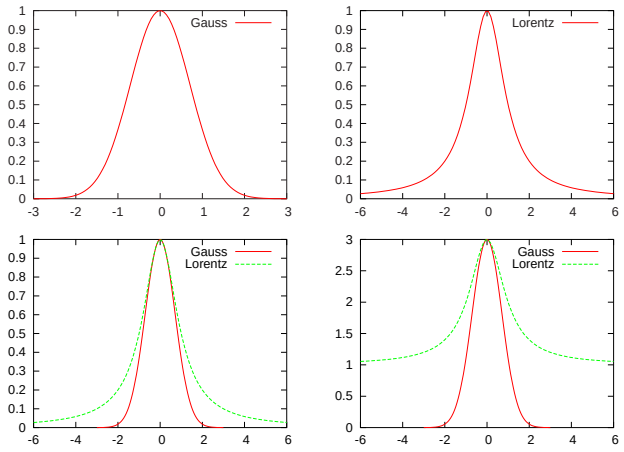

図 3　subplot 命令は，1 つのページを複数の枠を分割し，それぞれにプロットを行うことができます．

目次へ

4.3 3 次元プロット：mesh

3 次元のプロット関数 mesh は $z = f(x, y)$ のように，z が x, y の陽関数で与えられた場合の表面を描画します．$f(x, y, z) = C$ のような陰 (implicit) 関数は残念ながら描画できません（Matlab では可能です）．

plot と似ているのですが，z データを次のような $m \times n$ の格子状に与える必要があります．

```
z(x1, y1) z(x1, y2) z(x1, y3) ...  z(x1, yn)
z(x2, y1) z(x2, y2) z(x2, y3) ...  z(x2, yn)
...
z(xm, y1) z(xm, y2) z(xm, y3) ...  z(xm, yn)
```

また，x, y データもそれに応じた行列が必要になります．この行列はちょっとだけ頭をひねれば自分でも作成できますが，範囲を指定している x, y のベクトルから生成する命令

```
[X,Y]=meshgrid(x,y);
```

を使うのが得策です．具体例を示しますので確認してください．

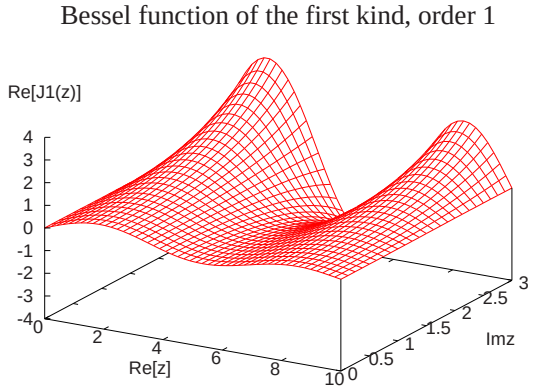

図 4　mesh による Bessel 関数 $J_1(z)$ の 3 次元表示

リスト 2　bessel.m

```
 1: N=50;M=25;
 2: x=linspace(0,10,N);
 3: y=linspace(0,3,M);
 4: [X,Y]=meshgrid(x,y);
 5: Z=X+Y*i;
 6: J=besselj(1,Z);
 7: title("{/Times=30 Besel Function of the \
 8: first kind, order 1}")
 9: xlabel("Re[z]")
10: ylabel("Im{z}")
11: zlabel("Re[J1(z)]")
12: gset ticslevel 0;
13: gset nokey
14: mesh(X,Y,real(J));
15: pause
16: gset term postscript enhanced eps color \
17:      "Helvetica" 20
18: gset output "bessel.eps"
19: replot
```

等高線図：contour　等しい z 値を持つ座標をつないだ曲線を表示する等高線図を描く命令は，**contour** です．描くために必要なデータは mesh とほぼ同じですが，微妙に異なります．contour(x,y,z) では x,y は行列ではなくベクトルを与える必要があるのです．

リスト 3　bessel-c.m

```
 1: N=50;M=25;
 2: x=linspace(0,10,N);
 3: y=linspace(0,3,M);
 4: [X,Y]=meshgrid(x,y);
 5: Z=X+Y*i;
 6: J=besselj(1,Z);
 7: title("{/Times=30 Contour plot of J_1(z)}")
 8: xlabel("Re[z]")
 9: ylabel("Im{z}")
10: zlabel("Re[J1(z)]")
11: gset ticslevel 0;
12: gset nokey
13: gset size square
14: contour(y,x,real(J));
15: pause
16: gset term postscript enhanced eps color \
17:      "Helvetica" 20
18: gset output "bessel-c.eps"
19: replot
```

4.4 タイトル，軸名 etc.

図面には，表題 (title)，軸 (axis)，目盛 (tics) などの要素があります．また，文字列を記入する必要があるかもしれません．細かくは制御できませんが，一通りの機能がありますから，簡単に説明しておきましょう．

目次へ

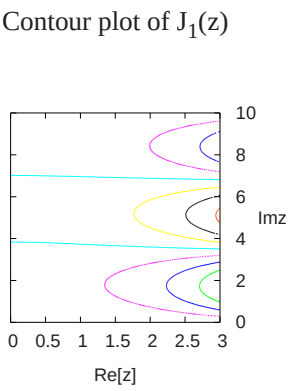

図 5　contour による Bessel 関数 $J_1(z)$ の等高線図

4.4.1　タイトル名：title

図の上に中央揃えで表題を付けるには以下の命令を用います．

```
title("strings")
```

文字列は 2 重括弧で囲まなければいけません．ところで，自動的に日付や描いた曲線のパラメータなどを入れたいと思うこともあるでしょう．Octave には日付を得る命令 date がありますから，日付を含んだ**文字列変数**を **sprintf** で生成して，title 命令に渡してください．具体的には次のようになるでしょう．

```
Title = sprintf("The date is %s", date());
title(Title)
```

4.4.2　軸：axis

座標軸の設定には axis 命令を用います．

```
axis([xs, xf, ys, yf, zs, zf], "options")
```

最初の括弧の中の数値は，x, y, z 軸の描画範囲を指定するもので，y, z に関する指定はなくても構いません．*options* の詳細は，help axis で見てもらうことにして省略しますが，頻繁に使用する square，equal だけを取り上げます．読んで字の如く，枠を正方形にする指定が square です．また，軸の単位長さを一定に揃える指定が equal です．どちらも，描かれる図の形が歪まないようにしたい場合に使います．次の例でその働きを確認してください．

```
oct> t=linspace(0,2*pi,101);
oct> plot(sin(t),cos(t))
oct> axis("square");
oct> replot
oct> axis([-2, 2],"square");
oct> replot
oct> axis([-2, 2],"equal");
oct> replot
```

4.4.3　文字列：text

文字列を図中に描くには text 命令を用います．書式は以下のようになっています．

```
text(x, y[, z], "文字列", "属性", 属性値)
```

描く位置を x, y[, z] で指定しますが，どの座標を使うか属性 Units で指定できます．既定の属性値は"data"で，曲線を描く座標系を使います．曲線の座標とは関係なく，画面上の位置を指定するには "screen" を指定します．これは，この講義では，提出課題の図面の左上に学籍番号などを記入する場合に，以下のようにして使います．

```
text(0.05, 0.95, "01ks000", "Units", "screen");
```

また，曲線のパラメータなどを自動的に書き入れるには，title で説明したように，**sprintf()** を使って text に渡す文字列を予め作成しておけば可能となります．

4.5　gset, gplot, gsplot

プロット関連（イメージ関連は別）のグラフィックに関しては，Matlab 互換の，いわば標準関数（あるいは標準関数が octave で実装されていない）ではできない事柄があります．そのような事柄は，生の gnuplot 命令を用いて実現することが可能となる場合があります．

もちろん gnuplot の命令を知っていることが前提となりますが，それは皆さんの好奇心に任せるとして，この講義で必要となる事柄だけを取り上げます．

4.5.1　gset

gnuplot では，図の属性を多岐に渡って set .... 命令で設定します．この重要な命令を octave から実行するのが **gset** です．

印刷　この講義では図面をプリンタで印刷して提出することが求められます．Matlab では印刷命令 print があるのですが，octave では今のところ実装されていません．そこで，gnuplot に印刷に適した形式のファイルを作成させます．基本的な文は

```
gset term postscript enhanced eps color \
     "Helvetica" 20
gset out "kadai.eps"
replot
```

となります．メディアセンターのプリンタは PostScript（以後 PS と略します）という（ページ記述）言語を理解します．最初の命令は PS を理解する装置（正確には端末：terminal）が相手であることを指定しています．オプ

目次へ

ションがいくつかあって enhanced は PS の拡張命令（ギリシャ文字などが使えます）の使用，eps（encapsulated PS）は大きさを図自身のもの（普通は紙の大きさになってしまう）にする指定，color は色を付ける（既定は白黒：monochrome）指定，最後の "Helvetica" 20 は，ラベルに用いるフォントの種類と大きさの指定です．なお，フォントは Helvetica，Times，Courier，Symbol が使えます．4 種のフォントを表示する例を示します．

リスト 4 textfont.m

```
plot([0, 1], [0, 1], ".;;");
text(0.05, 0.9, "Helveta ABC abc 10^{-2}y_{j}", \
     "FontName", "Helvetica", "FontSize", 36);
text(0.05, 0.7, "Times ABC abc 10^{-2}y_{j}", \
     "FontName", "Times", "FontSize", 36);
text(0.05, 0.5, "Courier ABC abc 10^{-2}y_{j}", \
     "FontName", "Courier", "FontSize", 36);
text(0.05, 0.3, "Symbol ABC abc 10^{-2}y_{j}", \
     "FontName", "Symbol", "FontSize", 36);
text(0.05, 0.1, "Compound {/Symbol s = p}a^{2}", \
     "FontName", "Times", "FontSize", 36);

gset term postscript enhanced eps color
gset out "textfont.eps"
replot
```

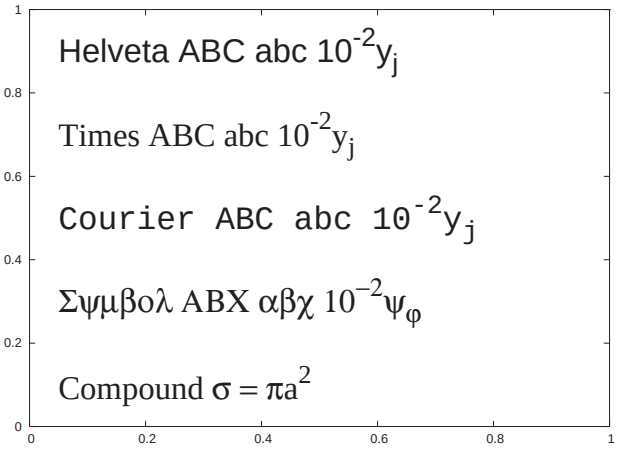

図 6　使用できる 4 種類のフォントの表示

4.5.2　gplot

Gnuplot の 2 次元プロット命令の名前も plot ... です．Ocatve の **gplot** は gnuplot の plot 命令およびオプション指定に準拠した書式を持っています（全てではありませんが）．例えば，2 次元グラフの種類で，階段状グラフや細い棒グラフなどを以下のようにして描かせることができます．

```
ode> t=linspace(0, 2*pi, 101);
ode> data=[t' sin(t')];
ode> gplot data with steps title "step"
ode> gplot data with impulses title "impulse"
```

この例では，octave が一端解釈してから gnuplot を呼び出していますが，本当に gnuplot に直接コマンドを渡すことも可能で，例えば，$sin(x)/x$ を折れ線と記号両方 (lp：linespoints) を使って，記号種 6 番 (pt：point-type) 大きさ 4(ps：point-size)，線種 12 番 (lt:line-type) 線幅 3(lw：line-width) でプロットさせるには

```
oct> gplot "sin(x)/x lp lt 12 lw 3 pt 6 ps 4"
```

等とします．この命令を使えば，gnuplot の最新の機能を使うことも可能です．もっとも，これでは全く octave の出番はないことになります．

4.6　gsplot

Gnuplot の 3 次元プロット命令の名前は splot です．Gnuplot の plot に対する gplot と同様に，Ocatve が splot の書式に準拠して実装している命令が **gsplot** です．gsplot は gplot よりも活躍します，というのも mesh では描けない重要な 3 次元の図があるからです．

3 次元曲線　mesh は曲面を描く関数であって，3 次元の曲線を描くようにはできていません．そこで **gsplot** の登場です．gplot や plot は x, y の組みですが，gsplot では gset parametric としておくと，x, y, z の組みを座標値の並びと解釈して 1 本の 3 次元曲線を描きます．中心軸から遠ざかる螺旋を描いてみましょう．

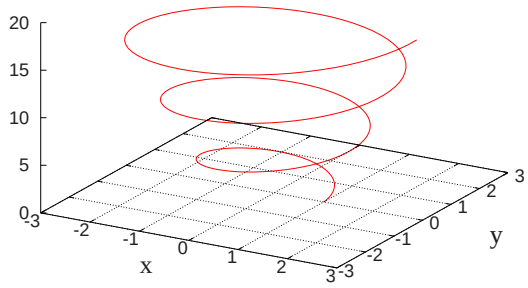

図 7　3 次元曲線の描画例；parametric モードの利用

目次へ

リスト 5 spiral.m

```
 1: t=linspace(0, 6*pi, 201);
 2: x=(1+0.1*t).*cos(t);
 3: y=(1+0.1*t).*sin(t);
 4: z=t;
 5: data = [x' y' z'];
 6: xlabel("{/Times=28 x}");
 7: ylabel("{/Times=29 y}");
 8: grid
 9: gset parametric
10: gset ticslevel 0
11: gset nokey
12: gset ztics 5
13: gsplot data
14: pause
15: %
16: gset term postscript enhanced eps color \
17:       "Helvetica" 20
18: gset out "spiral.eps"
19: replot
20:
```

5 微分

実は Octave には**数値微分の関数がありません**．関数 `diff(X,K)` は，K 次の前進差分を計算するものです．最も単純な数値微分は

$$\frac{df}{dx} \sim \frac{f_{n+1} - f_n}{x_{n+1} - x_n}$$

ですが，これは 1 次の差分列 $\Delta f_n = f_{n+1} - f_n$ から簡単に計算できます．関数 x^3 の導関数 $3x^2$ と単純な数値微分とを比較するスクリプトを以下に示します．誤差をなるべく少なくするように，ちょっとだけ工夫がしてあります．それは，導関数の値を計算する点を等差数列 x_n による区間 $[x_n, x_{n+1}]$ の端ではなく中央 $\frac{x_n + x_{n+1}}{2}$ となるようにしていることです．理由は簡単ですので各自，数値計算の入門書で調べてください．

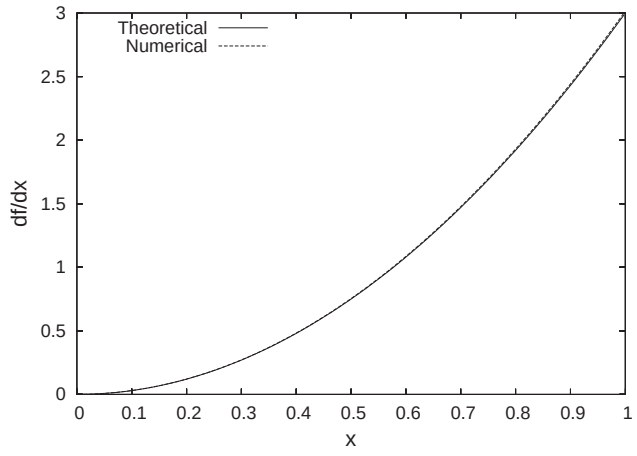

図 8 x^3 **の微分** $3x^2$ **と数値微分との比較**

リスト 6 diff-ex.m

```
 1: N=200;x0=0;x1=1;
 2: X=linspace(x0,x1,N);
 3: dx=(x1-x0)/N;
 4: XC=linspace(x0+dx/2,x1-dx/2,N-1);
 5: F=X.^3;
 6: DFT=3*XC.^2;
 7: DFN=diff(F)/dx;
 8: xlabel("x");
 9: ylabel("df/dx");
10: plot(XC,DFT);
11: hold on
12: plot(XC,DFN);
13: pause
14: hold off
15: #
16: # for creating PS figure.
17: #
18: data = [XC', DFT', DFN'];
19: gset key left
20: gset term postscript eps "Helvetica" 20
21: gset xlabel "x" "Helvetica,24"
22: gset ylabel "df/dx" 1,0 "Helvetica,24"
23: gset out "diff-ex.eps"
24: gplot data t "Theoretical", \
25:       data using 1:3 t "Numerical"
```

問題 6 関数 x^3 について，2 階微分の解析的な計算結果と数値微分による計算結果を比較して図に表示するスクリプトを考えなさい．

6 積分

1 変数の関数 $f(x)$ を数値積分する関数はいくつかありますが，ここでは標準関数 quad (Quadrature) を説明します．

Octave では，公式配布の関数以外に，ユーザーがたくさんの関数を開発，公開しています．数値積分に間しては，さすがに quad 1 つではあまりにも寂しく，かなりの数の関数が寄贈されています．その中には，2，3 次元，あるいは n 次元積分を実行するものも含まれています．

6.1 quad

書式は，以下のようになっています．

```
quad(F, A, B, TOL, SING)
```

`F` に関数名，`A,B` に積分区間の両端，`TOL` に（必要ならば）許容誤差，最後の `SING` は（もしあれば）特異点を並べたベクトルを指定します．正弦積分関数 $\mathrm{Si}(x)$ を計算して表示するスクリプトの例を示します．

目次へ

リスト 7 Si.m

```
 1: function ret=si(x)
 2:  ret=sin(x)/x;
 3: endfunction
 4:
 5: DIV=201;
 6: x=linspace(eps,50,DIV);
 7: for k=1:DIV
 8:   y(k)=quad("si", 0, x(k), [0;1e-6]);
 9: endfor
10:
11: xlabel("x");
12: ylabel("Si(x)");
13: plot(x,y, ";Si(x);");
14: hold on
15: plot(x,pi/2*ones(1,DIV), ";PI/2;");
16: pause
17: #
18: # for creating PS figure
19: #
20: gset term postscript eps "Helvetica" 20
21: gset xlabel "x" "Helvetica,24"
22: gset ylabel "Si(x)" "Helvetica,24"
23: gset out "Si.eps"
24: replot
```

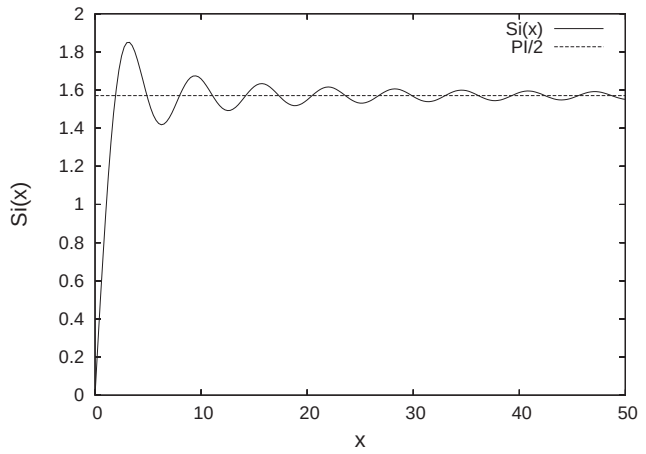

図 9 数値積分 quad による正弦積分 $\mathrm{Si}(x) = \int_0^x \frac{\sin\xi}{\xi}\,d\xi$ **曲線**

問題 7 次の定積分値が正しいことを quad で確かめなさい．なお，octave では無限大を inf で表します．

(i) $\displaystyle\int_0^{\pi/2} \sin x\,dx = \int_0^{\pi/2} \cos x\,dx = 1$

(ii) $\displaystyle\int_0^{\pi} x\sin x\,dx = \pi \qquad \int_0^{\pi} x\cos x\,dx = -2$

(iii) $\displaystyle\int_0^{\infty} \frac{dx}{1+x^2} = \frac{\pi}{2} \qquad \int_0^{\infty} \frac{x^3}{e^x-1}\,dx = \frac{\pi^4}{12}$

7 プログラム言語

少し込み入った計算をするには，どうしてもプログラミングが必要になります．分岐処理，繰り返し，ユーザー関数の定義，変数のスコープ（グローバル変数）など，最低限必要な事柄を説明します．

7.1 条件分岐

式の値に応じて処理を変更するというのはプログラミングの中核です．そのような手続きは **条件分岐** と呼ばれます．**もし ～ ならば イ，そうでなければ ロ** という最も基本的な手続きは

```
if 条件式1, 実行文1
else 実行文0
end[if]
```

という構文で記述します．最後の終了宣言は end でじゅうぶんですが，入り組んできた場合に，他の文の終了と区別するために，endif を用いることもできます．例を示しましょう．

```
oct> x=3;
oct> if x >= 2, x  <-- x が 2 以上なら x を表示
> else x-3         <-- そうでないなら x-3 を表示
> end              <-- if 文の終了
x = 3              <-- 3 >= 2 なので x = 3 を表示
oct> if x < 2, x
> else x-3
> end
ans = 0
```

となります．分岐条件を複数設定することもできます．

```
if 条件式1, 実行文1
elseif 条件式2, 実行文2
...
elseif 条件式k, 実行文k
else 実行文0
end
```

例をしめしましょう．何度も history を使って呼び出せるように 1 行に書いています．

```
oct> x=1;
oct> if x < 1, 1  elseif x > 1, 0  else 1/2 end
ans = 0.50000
oct> x=2;
oct> if x < 1, 1  elseif x > 1, 0  else 1/2 end
ans = 0
oct> x=-1;
oct> if x < 1, 1  elseif x > 1, 0  else 1/2 end
ans = 1
```

7.1.1 論理演算と比較演算

ところで，“x > 1 または x < -1 が（真）ならば” のような複合条件を全て if 文で書くのは大変なので，一般には **論理演算子** を用いて記述します（表 4）．

とても簡単な例を示しましょう．

目次へ

表 4　Octave の論理演算子

記号	記述例	意味
|	A|B	A または B（or）
&	A&B	A かつ B（and）
~	~A	A でない（not）

```
oct> x=0;
oct> if x < -1 | x > 1, 1 else 0 end
ans = 0
oct> x=2;
oct> if x < -1 | x > 1, 1 else 0 end
ans = 1
oct> x=0
oct> if x < 1 & x > -1, 1 else 0 end
ans = 1
```

いままで断りもなく使ってきましたが，大きさを比較する演算子も表 5に整理しておきます

表 5　Octave の比較演算子

記号	記述例	意味
<	A < B	A が B よりも小さい
<=	A <= B	A が B 以下
>	A > B	A が B より大きい
>=	A >= B	A が B 以下
==	A == B	A が B に等しい
~=	A ~= B	A が B に等しくない

ここで注意があります．プログラミングの初心者は

```
if A == B を if A = B と書いてしまうミス
```

を冒しやすいです．後者は代入演算であり結果は常に真です．また

```
記号 +=>, =<, =~ は定義されていません
```

気をつけてください．

7.1.2　配列間の比較と論理演算

Octave の特徴は，行列への演算が基本ということでした．比較演算や論理演算も類をまのがれません．同じ大きさの行列間では成分毎に演算が行われ，**論理値** である 1（真）0（偽）を成分とする行列が結果として得られます．また，スカラとの演算も加減算と同様に成分に対して実行されるのです．また，論理値の算出は 0 のみが偽（論理値 0）であって，それ以外は真（論理値 1）と扱われます．

```
oct> A=[1 2 3 4]; B=[1 0 1 0];
oct> !B,  A > B,  A & B
ans =
  0  1  0  1
ans =
  0  1  1  1
ans =
  1  0  1  0
```

問題 8　次の内容のプログラムを作成しなさい．

(i) 変数 x が正の数ならば 1，0 ならば 0，負の数ならば -1 を表示する．

(ii) 変数 x が正の数ならば記号 “+”，0 ならば 0，負の数ならば記号 “-” を表示する．

ヒント:文字列はダブルクォート “"” で括ります．例えば，`"abc" "String" "+-"`

(iii) 変数 t が 12 より大きく 24 未満ならば 12 をひいた数を表示，0 より小さいか 24 以上ならば `Out of range!` を表示，0 以上 12 以下ならばそのまま表示する．

7.2　繰り返し

パラメータを変えながら同じような処理をある判定条件を満たす限り実行し続ける手続きは**繰り返し** と呼ばれます．

7.2.1　WHILE ループ

C 言語でもお馴染み，汎用の繰り返し手続き while 構文があります．書式は

```
while 条件式
  実行文
end[while]
```

です．例えば 1 から 5 の自乗を求める命令文は

```
oct> x = 1; while x <= 5, x**2, x++; end
ans = 1
ans = 4
ans = 9
ans = 16
ans = 25
```

と記述できます．x++ の ++ は，その前の変数（この場合は x）を 1 増加させる演算子です．

7.2.2　FOR ループ

while ループはパラメータを変える部分と，本来の処理を同一ループ内に記述しなければなりませんでした．パラメータの範囲を（行列で）指定してループを構成するのが

目次へ

FOR 文です.

```
for k = 範囲
  実行文
end[for]
```

例えば,

```
oct> for i=1:0.5:2 i**2 end
ans = 1
ans = 2.2500
ans = 4
```

問題 9 繰り返しループを用いて，次の内容のプログラムを作成しなさい.

(i) 5 次の Hilbert 行列 ($h_{ij} = 1/(i+j-1)$) を生成する.


3.1.3節で説明しましたが，Hilbert 行列を生成する関数 `hilb(N)` があります．結果を比較し，できればソースも調べてみるといいでしょう.

(ii) フィボナッチ数列 ($b_0 = 0,\ b_1 = 1, b_n = b_{n-2} + b_{n-1}$) の最初の 10 項を成分とするベクトルを生成する.


Fibonacci 数列の漸化式を辿らなくとも，第 n 項は次式で与えられます.

$$b_n = \frac{1}{\sqrt{5}}\left\{\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \left(\frac{1-\sqrt{5}}{2}\right)^{n+1}\right\}$$

7.3 ユーザー関数

複雑な処理を単純な処理に分解して関数としてまとめるという方針は，プログラミングの大原則です．無論 Octave でもユーザー関数が定義できます．書式は

```
function [出力リスト] = 関数名 (入力リスト)
  定義文
end[function]
```

関数名は関数が呼び出されるときの名前で，(入力リスト) はなくても構いません．また，[出力リスト] は，戻り値の並びで関数内で定義されている必要があります．また戻り値の変数（もちろん行列です）が 1 つだけならば括弧を省略できます．例をいくつかあげましょう．円の半径 r を引数にとって面積を求める関数を `circlearea` という名前で定義してみましょう.

```
oct> function s = circlearea(r)  s=pi*r.^2; end
oct> circlearea(5)
ans = 78.540
oct> circlearea([1 2; 3 4])
ans =
   3.1416  12.5664
  28.2743  50.2655
```

行列に対しても計算ができるように，配列演算 "`pi*r.^2`" を使っています．"`pi*r^2`" では引数に行列を与えることはできません．次に，面積と円周を戻す関数 `circle` を定義してみましょう.

```
oct> function [s,l]=circle(r)
> s=pi*r.^2; l=2*pi*r;
> end
oct> [S,L]=circle([1,2,3,4])
S =
   3.1416  12.5664  28.2743  50.2655
L =
   6.2832  12.5664  18.8496  25.1327
```

呼び出す場合には，戻り値 [出力リスト] と同じ形式の行ベクトルに代入します.

問題10 次の計算を実行する関数を作成しなさい.

(i) 二つの 3 次元ベクトル $\boldsymbol{a}, \boldsymbol{b}$ の外積 $\boldsymbol{c} = \boldsymbol{a} \times \boldsymbol{b}$ を求める関数 `vecprod(a,b)`.

(ii) 二次方程式 $x^2 + bx + c = 0$ の根を根の公式を使って求める関数 `secpol(b,c)`．判別式 $D = b^2 - 4c$ による場合分けをする必要があります.


Octave には，多項式の根を求める関数 "`roots(V)`" があります．`V` は多項式の係数を順に並べた係数ベクトルです．したがって，上記二次方程式の根は "`roots([1,b,c])`" で求めることができます.

7.4 変数の有効範囲

関数内部で使われる変数名は，その関数内部でのみ有効で，関数の外からは参照できません．このような変数は**局所変数**と呼ばれます．Octave で宣言される変数は基本的には局所変数です．したがって，関数には *inputlist* を通じてパラメータの**値を渡す**ことしかできませんが，その代わり関数の外で宣言された変数の値が関数内部で書き換えられる心配がありません.

```
oct> k = 5;  <-- k を初期化
oct> function tslocal(x) k=x**2 end
oct> tslocal(8)
k = 64  <-- 関数内部の局所変数 k の値
oct> k
k = 5  <-- 最初の k の値は変化してない
```

これは**隠蔽**と呼ばれ，誤りのないプログラムを書く上で大変重要な用件です．しかしながら，どうしてもプログラム全体で同じ変数（どこでも書き換えが可能なので **大域変数**あるいは**グローバル変数**と呼ばれます）を使わざるを得ない場合があります．例えば，本講義の主たる題材である常微分方程式の数値解を得る関数では，扱う関数の *inputlist* が決まっていてパラメータを含ませることができません．この場合にはグローバル変数を使うしか手立てがないのです.

目次へ

```
oct> global k <-- k をグローバル宣言
oct> function tslocal(x) global k; k=x**2 end
        関数内部でもグローバル宣言↑
oct> tslocal(8)
k = 64  <-- 関数内部で k に代入
oct> k
k = 64  <-- グローバルなので書き換えられた
```

問題11 グローバル変数を利用して次の内容のプログラムを作成しなさい.

(i) ボルツマン定数 $k = 1.38 \times 10^{-38}\,\mathrm{J\,K^{-1}}$ およびアヴォガドロ数 $N_A = 6.02 \times 10^{23}$ 個 mol^{-1} をグローバル変数として，気体の速さ分布関数

$$f(v) = \frac{4}{\sqrt{\pi}} \left(\frac{m}{2kT}\right)^{\frac{3}{2}} v^2 \exp\left(-\frac{mv^2}{kT}\right) dv$$

および，熱平均速度

$$\bar{v} = \sqrt{\frac{8kT}{\pi m}}$$

をユーザー関数 `f(T,m)`, `vt(T,m)` として定義しなさい．次に，温度 $T = 300\,\mathrm{K}$ における窒素分子 $m_{N_2} = 28.0 \times 10^{-3}\,\mathrm{kg\,mol^{-1}}$ の速さ分布関数をグラフに描きなさい．また，熱平均速度を求めなさい.

問題12 振動数 ω_0 の固有振動系を，振動数 ω の外力で励振すると，十分時間が経過した後には ω の定常振動のみが残ります．この定常振動の振幅 A は減衰係数 λ にも依存しますが，λ をパラメータとみなして，$A(\omega)$ を描きなさい.

☞ 運動方程式は,

$$\frac{d^2x}{dt^2} = -\omega_0^2 x - \lambda \frac{dx}{dt} + e^{-i\omega t}$$

と表されます．$x(t) = Ae^{-i\omega t}$ の形で書けるとして，与式に代入すると，複素振幅

$$A = \frac{1}{\omega_0^2 - i\omega\lambda - \omega^2}$$

を得ます．これの絶対値が求める関数です．例えば $\omega_0 = 100$, $\lambda = 10, 20, 50, 100$ として描いた結果を図 10に示します.

7.5 ファイルの読み書き

データを保存する命令は **save** です．書式は

```
save [options] file v1 v2 ...
```

となっていて，保存ファイル名 (*file*) に - を指定すると標準出力が用いられます．オプションには出力形式が指定できますが，当面は "-ascii" として**平テキスト（plain text）**で保存しましょう．その他にどのような**バイナリ（binary）**形式が実装されているかは，" help save" で調べてください．例を示しましょう

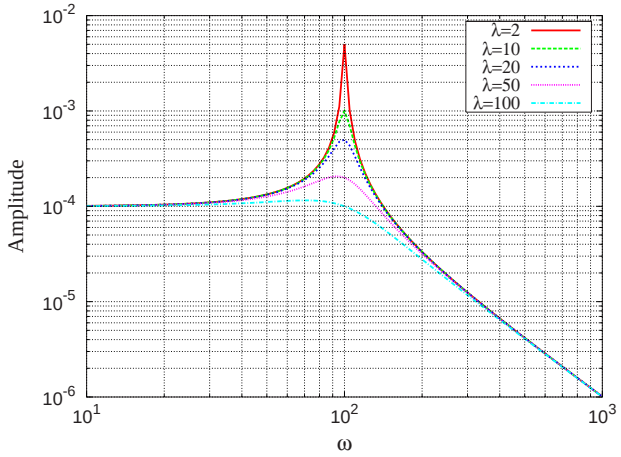

図 10 （自由）固有振動数 $\omega_0 = 100$ をもつ減衰振動系に，振動数 ω の外力を作用させた場合の共振曲線

```
oct> x=linspace(0,1,11); y=x';
oct> save testfile.dat x y
```

以上で，"testfile.dat" という名前のファイルが作成されて，行ベクトル x とその転置の列ベクトル y が保存されます．なお，-ascii が既定（default）の保存形式に設定されていれば，ここでオプション指定する必要はありません．保存された内容を表示してみましょう．Octave には残念ながらファイルの中身を表示する命令がありません．したがって，基本的には一端 octave を終了して，他のツールを実行することになります．Unix の世界では，テキストファイルを表示させるツールとしては less が有名です．では，一端終了してと... 待ってください．終了しなくても，外部ツールを利用する命令があります．"system(*strings*)" です．具体的には以下のようにできます.

```
oct> system("less testfile.dat")
# Created by Octave 2.1.36, Wed Jul 17 17:00:04 2002 JST
  <matuda@book2> <-- 折り返しています．実際には 1 行.
# name: x
# type: matrix
# rows: 1
# columns: 11
 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
# name: y
# type: matrix
# rows: 11
# columns: 1
 0
 0.1
 0.2
 0.3
 0.4
 0.5
 0.6
 0.7
 0.8
 0.9
 1
```

#記号で始まる行はヘッダーで，ファイルの作成時刻，変

目次へ

数名，変数型，行数，列数 などが記録されています．

逆に読み込みは “load” 命令を用います．

```
load [options] file
```

load はファイルに記された変数をそのまま使おうとしますので，既に同じ名前の変数があった場合には，警告を出して中止します．もし，変数の内容を書き換えても良いならばオプション “-force” を指定します．あるいは，同名の変数を”clear v1 v2...” 命令で消去するという荒技を使うこともできます．以下の具体例は，変数を全て消去して（clear のみの場合はユーザー変数を全て消去します）から読み込むという，とんでもない荒技の例です．もちろんこんな無謀はいけません．よーく考えてから読み込みを行ってください．

```
oct> clear
oct> load testfile.dat
oct> who
*** local user variables:
x  y
```

8 常微分方程式

この講義の中核となる常微分方程式を解く関数を説明しましょう．標準配布されているものは，lsode という名前で，Alan C. Hindmarsh 氏の解法を実装したものです．数値解析の教科書で必ず載っている Runge-Kutta 法によるものは，標準配布されていませんが，Marc Compere 氏が開発した rk4fixed，rk8fixed があります．それぞれ 4 次と 8 次の Runge-Kutta 法を実装したものです．どの関数も微分方程式

$$\frac{d\boldsymbol{x}}{dt} = f(\boldsymbol{x}, t)$$

を解く (すなわち，$\boldsymbol{x}(t)$ が計算される) ように設計されています．微分方程式の内容は関数 f(x,t) で与えます．すなわち f(x,t) を定義する際，戻り値に，変数ベクトル $\boldsymbol{x} = (x_1, x_2, x_3, \cdots)$ の各成分の t による微分を必ず記述します．

$$\begin{aligned}
\frac{dx_1}{dt} &= g_1(\boldsymbol{x}, t) \\
\frac{dx_2}{dt} &= g_2(\boldsymbol{x}, t) \\
&\cdots \\
\frac{dx_k}{dt} &= g_k(\boldsymbol{x}, t) \\
&\cdots
\end{aligned}$$

結局のところ，連立の 1 階微分方程式を解くという単純な機能となっているのです．最も簡単な例を示しましょう．

$$\frac{dx}{dt} = \frac{1}{1+t^2},\ x(0) = 1 \ \Leftrightarrow\ x(t) = \arctan t + 1 \qquad (1)$$

という微分方程式？は，以下のように定義します．

```
function xdot = f1(x,t)
  xdot(1)=1/(1+t^2);
end
```

もう少し微分方程式らしい

$$\frac{dx}{dt} + x = 0,\ x(0) = 1 \ \Leftrightarrow\ x(t) = e^{-t} \qquad (2)$$

は次のように定義します．

```
function xdot = f2(x,t)
  xdot(1)=-x(1);
end
```

以上のように機能は単純ですが，工夫次第で使い道が広がります．教科書にあるように，

$$\frac{dx_1}{dt} = x_2 \quad \text{と定義すれば} \quad \frac{dx_2}{dt} = \frac{d^2x_1}{dt^2}$$

となって x_1 の 2 階微分を扱えることになります．例えば，

$$\begin{aligned}
&\frac{d^2x}{dt^2} + 4x = \sin(t),\ x(0) = 1,\ \dot{x}(0) = 0 \\
&\Leftrightarrow\ x(t) = \cos 2t - \frac{1}{6}\sin 2t + \frac{1}{3}\sin t
\end{aligned} \qquad (3)$$

のような 2 階の常微分方程式は

```
function xdot = f3(x,t)
  xdot(1)=x(2);
  xdot(2)=-4*x(1)+sin(t);
end
```

と定義すれば良いことになります．この考え方は非常に重要ですから良く習熟してください．

では lsode，rk4fixed，ode78 の具体的例を示しながら使い方を説明しましょう．引数が微妙に異なるので注意が必要です．

8.1 lsode

標準配布関数 lsode の書式は以下の通りです．

```
lsode(FCN, X0, T, T_CRIT)
```

FCN には微分方程式を定義している関数の名前，X0 は変数ベクトル $\boldsymbol{x}$ の初期値，T には独立変数 t の範囲（分割点の値を並べたベクトル）を与えます．T_CRIT はオプションで，計算から外す t の値を並べたベクトルを与えます．たとえば，微分不連続点や特異点などを除く場合に有効です．(1) 式の数値解を計算し，解析解に重ねて図に表示するためのスクリプトを以下に示します．

目次へ

リスト 8 lsode-f1.m

```
 1 : function dx = f1(x,t)
 2 :  dx(1) = 1/(1+t^2) ;
 3 : end
 4 :
 5 : t=linspace(0,10,101);
 6 : x=lsode("f1", 1, t);
 7 : gset key bottom
 8 : title("Solution for x'=1/(1+t^2), x(0)=1")
 9 : plot(t, x, "@16;lsode;");
10 : hold on
11 : plot(t, atan(t)+1, "-;true;");
12 : pause
13 : #
14 : # for creating PS figure
15 : #
16 : gset term postscript eps color "Helvetica" 20
17 : gset out "lsode-f1.eps"
18 : replot
```

初期値に [1] とベクトルとして与えてますが，もちろん変数が 1 つの場合には，スカラ 1 も可です．数値解は図で丸で表されるように，"@16"（@:記号による離散プロット，1:赤，6:記号の種類で丸）をオプション指定しています．図 11 に結果を示します．

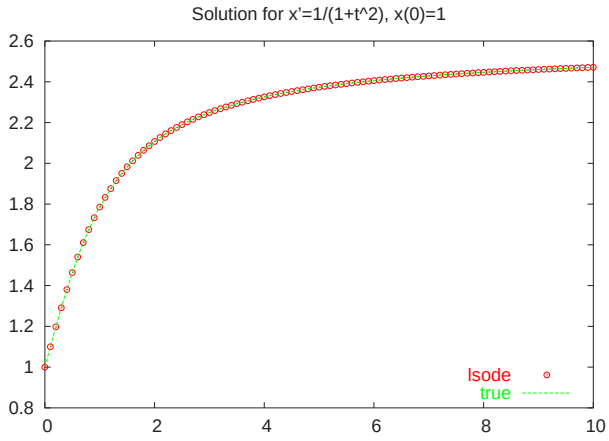

図 11 微分方程式 $x' = 1/(1+t^2),\ x(0) = 1$ の lsode による数値解と解析解 $\arctan(t) + 1$ の比較

問題13 (2), (3) 式の数値解を lsode で求め，解析解と重ねて図示してみなさい．図 12，13 に解答例を示します．

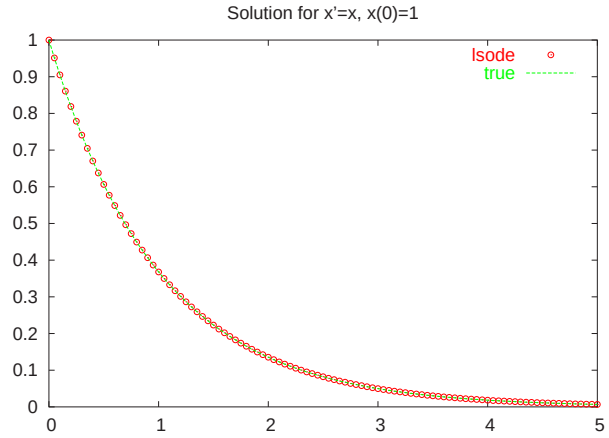

図 12 微分方程式 $x' = x,\ x(0)$ の lsode による数値解と解析解 e^{-t} の比較

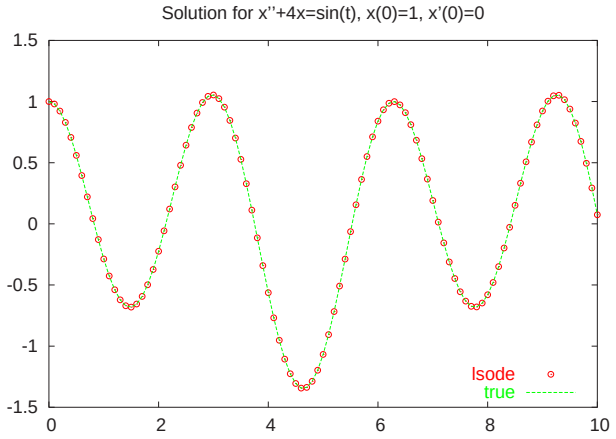

図 13 微分方程式 $x'' + 4x = \sin(t),\ x(0) = 1,\ x'(0) = 0$ の lsode による数値解と解析解 $\cos 2t - \sin 2t/6 + \sin t/3$ の比較

8.2 rk4fixed

4 次の Runge-Kutta 法は，数値解析の教科書で必ず取り上げられる優れた解法です．ということで関数の名前の前半 rk4 の意味はすぐに判ることでしょう．後半の fixed は，t の刻み幅が固定（等間隔）であることを表しています．lsode では，計算を行う t の値をユーザーが指定します．linspace などを使って等間隔にしましたが，その理由は特にありませんでした．単にそれ以外を指定する理由もなかったからです．しかし，良く考えてみると，解の変化の激しい部分では刻みを細かくし，変化の少ない部分では刻みを粗くする方が効率が良いはずです．そのような工夫

目次へ

が凝らされているのが ode45,ode78 です．しかしながら，一般には刻み幅固定でも十分な精度が得られる場合も多く，教科書の解説の通りに組まれた rk4fixed，rk8fixed はそのソースが勉強になることにおいても存在意義があります．

リスト 9 rk4-f3.m

```
 1: function dx = f3(x,t)
 2:  dx(1) = x(2);
 3:      dx(2)=-4*x(1)+sin(t);
 4: end
 5:
 6: [t,x]=rk4fixed("f3", [0 10], [1; 0], 100, 1);
 7: xt=cos(2*t)-sin(2*t)/6+sin(t)/3;
 8: gset key bottom
 9: title("Solution for x''+4x=sin(t), x(0)=1, x'(0)=0")
10: plot(t, x(:,1), "@16;lsode;");
11: hold on
12: plot(t, xt, "-;true;");
13: pause
14: % for creating PS figure
15: gset term postscript eps color "Helvetica" 20
16: gset out "rk4-f3.eps"
17: replot
```

さて書式ですが，lsode よりも少し複雑です．

```
[tout, xout] = rk4fixed(FCN, tspan, x0,
                Nsteps, format, trace, count)
```

FCN は関数の名前，tspan は独立変数 t の計算区間を [tstart, tfinal] の形式で指定します．x0 は解ベクトル $\boldsymbol{x}$ の初期値ですが **列ベクトル** で与えなければなりません．ここまでは，lsode と同様の要素です．以降 Nstep は t 区間の刻み具合を分割数で指定するものです．format は，関数の引数の順序が (x,t) ならば 1，(t,x) ならば 0 を指定します．既定は 0 です．すなわち，何も指定しない場合には lsode と**引数の順序が逆となりますから要注意**です．trace を真（0 以外の数値）にすると，計算経過が表示されます．もちろん既定では表示しません．count は重要ではないので省略します．

では，(3) 式の数値解を rk4fixed で求め，解析解と比較するスクリプトを**リスト 9** に示しましょう．得られる図は lsode の場合（図 13）とほとんど差がありません．

問題14 **(ロジスティック方程式)** 生物集団の個体数 x を時刻 t に関するの連続量とみなし，その時間変化が個体数 x に比例するだけではなく，個体数と飽和値との差 $s-kx$（k：定数）にも比例すると考えると，微分方程式は．

$$\frac{dx}{dt} = (s-kx)x$$

となります．数値解析を行って解析解

$$x(t) = \frac{se^{st}}{C' + ke^{st}} \qquad (C' : \text{任意定数})$$

と比較しなさい．

目次へ