数値計算法

池田建司 (C403)

【講義計画】

1	計算機における数の表現	
2	非線形方程式の解法 I	二分法
	非線形方程式の解法 II	Newton 法
3	非線形方程式の解法 III	割線法
4	数値積分 I	台形則
5	数値積分 II	Romberg 積分と Richardson 補外
6	常微分方程式の解法 I	Euler 法
7	常微分方程式の解法 II	修正 Euler 法
	常微分方程式の解法 III	Runge-Kutta 法
8	連立一次方程式の解法 I	LU 分解
9	連立一次方程式の解法 II	3 重対角, 対称行列の LU 分解
10	連立一次方程式の解法 III	ピボットの部分選択
11	最小 2 乗法	QR 分解, Householder 変換
12	最小 2 乗法	QR 分解, Householder 変換
13	行列の固有値問題	対称行列の固有値, Hessenberg 形, 原点移動, 減次

資料など

【講義目的】
数値誤差と計算の手間 (計算時間，作業領域)
代表的な数値計算のアルゴリズム
【参考書】
篠原能材「数値解析の基礎」日新出版
伊理正夫・藤野和建「数値計算の常識」共立出版
森正武「数値計算プログラミング」岩波書店

http://www-b2.is.tokushima-u.ac.jp/~ikeda/num/

—1. 計算機における数の表現—

数は有限桁

- 計算機の実数型は数学の実数ではない!! 無限の精度で計算できるなら、数学の公式通りにやればよい。
- 誤差, 精度を常に意識せよ。限られた資源で、大きな効果を生む。
  ··· エンジニアとしての腕の見せ所

計算機の中での数の表現と誤差

- 数の表現と丸め誤差
  浮動小数点表現、丸め誤差、マシンイプシロン、丸め誤差の蓄積
  (塵も積もれば山となる)
- 情報落ちと桁落ち
  桁揃え、情報落ち、桁落ち、2 次方程式の根 (数学公式との違い)

浮動小数点

$\beta(=2)$ を基数 (radix) とする β 進 t 桁の浮動小数点数 x は

$$x = \pm(d_1.d_2 \dots d_t)_\beta \times \beta^e \tag{1}$$

$$0 \le d_i \le \beta - 1, \quad d_1 \ne 0, \quad L \le e \le U \tag{2}$$

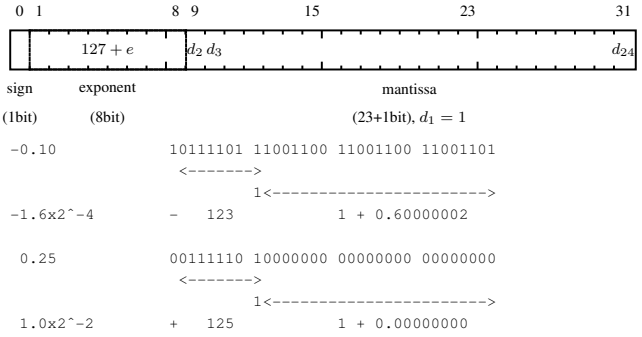
という形で表される。ここで、

$$\pm\,(d_1.d_2 \dots d_t)_\beta = \pm(d_1 + d_2\beta^{-1} + \cdots + d_t\beta^{-t+1}) \tag{3}$$

は仮数部 (mantissa), e あるいは β^e は指数部 (exponent) とよばれる。

IEEE 方式

SPARCstation20 での数の表現 (IEEE 方式)




```
-0.10           10111101 11001100 11001100 11001101
                 <------->
                         1<------------------------>
-1.6x2^-4       -   123           1 + 0.60000002

 0.25           00111110 10000000 00000000 00000000
                 <------->
                         1<------------------------>
 1.0x2^-2       +   125           1 + 0.00000000
```


オーバーフロー、アンダーフロー

浮動小数点表現で表すことのできる
絶対値最大の数 $(1 - \beta^{-t})\beta^{U+1}$
絶対値最小の数 β^L

オーバーフロー (overflow): 最大の数を越えること
アンダーフロー (underflow): 最小の数より小さくなること

丸め誤差 (roundoff error), 表現誤差

実数 x が有限桁の数 x^* で表されるとき $x^* - x$ を丸め誤差という. 実数 x が $\beta^{e-1} \le |x| \le \beta^e$ のとき, x^* の丸め誤差は

$$|x^* - x| \le \gamma\beta^{e-t}, \quad \gamma = 1(\text{切捨て}),\ \gamma = \frac{1}{2}(\text{四捨五入}) \tag{4}$$

相対誤差 $|x^* - x|/|x|$ は $d_1 = 1, d_2 = \cdots = d_t = 0$ のとき最大で $\gamma\beta^{1-t}$(マシンエプシロン, machine epsilon)

マシンエプシロン: 1.0 に加えたときに 1.0 と異なる結果になるような最小の正の浮動小数点数 ε_m のこと.

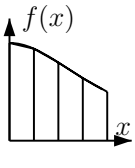
語長 32 ビットの典型的なコンピュータの ε_m はほぼ 3×10^{-8}.

丸め誤差の累積

$$I = \int_0^1 \frac{1}{1+x^2}dx = \frac{\pi}{4} = 0.7853982 \tag{5}$$

をきざみ幅 h を次第に小さくしながら, 台形則で積分

$$I_h = h\left[\frac{1}{2}f(0) + \sum_{j=1}^{n-1} f(jh) + \frac{1}{2}f(1)\right] \tag{6}$$

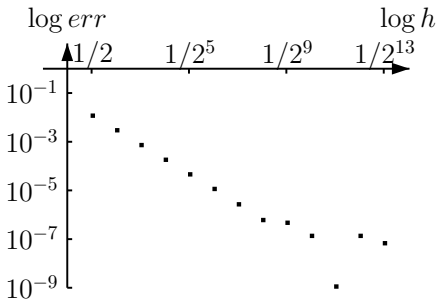


$$h = \frac{1}{n}, \quad f(x) = \frac{1}{1+x^2} \tag{7}$$

丸め誤差の累積

```
 1 /* Trapezoidal rule (tentative) */
 2 #include <stdio.h>
 3 #include <math.h>
 4 main()
 5 {
 6     float   h, s, ev = M_PI/4.0;
 7     int i,n,k;
 8
 9     for (k=1,n=2; k<14; k++,n*=2) {
10         h = 1.0/n;
11         s = 0.5*(1.0+0.5);
12         for (i=1; i<n; i++)
13             s += (1.0/(1.0+(h*i)*(h*i)));
14         s *= h;
15         printf("h=%f s=%f err=%e\n",h,s,s-ev);
16     }
17 }
```


丸め誤差の累積



誤差の発生

関数 $f(x)$ の計算 (例えば $f(x) = \exp(x)$)
x を丸めたものを x^* とする。f の有限回の四則演算での近似 f_a 例えば

$$f_a(x) = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 \quad \text{(テイラー展開)}$$

$f_a(x^*)$ の丸め $f_a^*(x^*)$ は

$$\begin{aligned} f_a^*(x^*) - f(x) = & \{f_a^*(x^*) - f_a(x^*)\} \\ & + \{f_a(x^*) - f(x^*)\} \\ & + \{f(x^*) - f(x)\} \end{aligned} \tag{8}$$

右辺第 1 項: 生成誤差, 発生誤差
右辺第 2 項: 打切り誤差, 公式誤差, 離散化誤差

右辺第 3 項: 代入誤差, 伝播誤差

情報落ちと桁落ち

桁揃え: 浮動小数点の加減算を行うために指数を揃えること.

情報落ち: 指数の小さいほうの数の下位の何桁かが, 桁揃えのために無視されること.

10 進 3 桁で考える.
$a = 0.123 \times 10^4$, $b = 0.556 \times 10^1$, $c = 0.552 \times 10^1$

a	0.123	$\times 10^4$
b	0.000556	$\times 10^4$(**桁揃えのため 3 桁右へシフト**)
$a+b$	0.123556	$\times 10^4$

切捨ての場合は 0.123×10^4, **四捨五入の場合は** 0.124×10^4

情報落ちと桁落ち

桁落ち: 大きさがほぼ等しい数の間の減算で, 上位の何桁かが失われること.

b	0.556	$\times 10^1$
c	0.552	$\times 10^1$
$b-c$	0.004	$\times 10^1 = 0.400 \times 10^{-1}$

有効数字は 1 桁に減ってしまう.

2 次方程式の根

2 次方程式 $0.001x^2 + 600x - 60.00001 = 0$ の根を解の公式 $x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ で求める。
本当の答えは
$x_1 = -600000.062500$, $x_2 = 0.1$(exactly).
解の公式に基づいて求めた結果は
$x_1 = -600000.062500$,$x_2 = 0.104167$.

2 次方程式の解法（×）

```
 1 #include <stdio.h>
 2 #include <math.h>
 3 main()
 4 {
 5     float   a,b,c,d,x1,x2;
 6     while (scanf("%f%f%f",&a, &b, &c)>=3) {
 7         printf("(%f %f %f) ",a,b,c);
 8         d = b*b-4.0*a*c;
 9         if (d < 0.0) {
10             x1 = -b/2.0/a;
11             x2 = sqrt(-d)/2.0/fabs(a);
12             printf("x = %f +- %f i\n",x1,x2);
13         } else {
14             x1 = (-b-sqrt(d))/2.0/a;
15             x2 = (-b+sqrt(d))/2.0/a;
16             printf("x1=%f, x2=%f\n",x1,x2);
17         }
18     }
19 }
```


2 次方程式の根

$0.001x^2 + 600x - 60.00001 = 0$ **の一つの解は,**

$$\begin{aligned} x_2 &= \frac{-300 + \sqrt{300^2 + 0.001 \times 60.00001}}{0.001} \\ &= \frac{-300 + \sqrt{90000.06000001}}{0.001} \\ &= \frac{-300 + 300.0001}{0.001} = 0.1(\text{exactly}) \end{aligned}$$

2次方程式の根

これを有効数値 8 桁で計算し，相対誤差を求めると，

$$
\begin{aligned}
D &= 90000.060 \\
\sqrt{D} &= 300.00009 \\
\hat{x}_2 &= \frac{-300.00000 + 300.00009}{0.0010000000} \\
&= \frac{0.0000090000000}{0.0010000000} = 0.090000000 \\
\frac{|\hat{x}_2 - x_2|}{|x_2|} &= 0.1 \qquad 10\% \text{ の相対誤差}
\end{aligned}
$$

2次方程式の根

やはり，有効数値 8 桁で次のようにして求めると，

$$
\begin{aligned}
\hat{x}_2' &= \frac{-b + \sqrt{b^2 - ac}}{a} \cdot \frac{-b - \sqrt{b^2 - ac}}{-b - \sqrt{b^2 - ac}} \\
&= \frac{c}{-b - \sqrt{b^2 - ac}} \\
&= \frac{-60.000010}{-300.00000 - 300.00009} \\
&= \frac{60.000010}{600.00009} = 0.100000001 \qquad 8 \text{ 桁正しい}
\end{aligned}
$$

$b \geq 0$ ならば 分子を $-b - \sqrt{D}$ で計算し、$b < 0$ ならば 分子を $-b + \sqrt{D}$ で計算する。$x_2 = \frac{c/a}{x_1}$ とすればよい.

2次方程式の解法（○）

```
 1 #include <stdio.h>
 2 #include <math.h>
 3 main()
 4 {
 5     float   a,b,c,d,x1,x2;
 6     while (scanf("%f%f%f",&a, &b, &c)>=3) {
 7         printf("(%f %f %f) ",a,b,c);
 8         d = b*b-4.0*a*c;
 9         if (d < 0.0) {
10             x1 = -b/2.0/a;
11             x2 = sqrt(-d)/2.0/fabs(a);
12             printf("x = %f +- %f i\n",x1,x2);
13         } else {
14             if (b>=0)
15                 x1 = (-b-sqrt(d))/2.0/a;
16             else
17                 x1 = (-b+sqrt(d))/2.0/a;
18             x2 = c/a/x1;
19             printf("x1=%f, x2=%f\n",x1,x2);
20         }
21     }
22 }
```


演習問題(レポート)

余裕がある人は * のついている問題もやってみよ。

1. 丸め誤差
   - pp.8–10 の丸め誤差の蓄積を確認せよ。
   - 自分の使っている計算機のマシンイプシロンはいくらか？
   - (6) 式の数値積分で刻み幅を小さくしていくと誤差は刻み幅に比例して小さくなるはずであるが、刻み幅がある値以下になると不規則になる。その値とマシンイプシロンの関係を考察せよ。

演習問題(レポート)

2. 桁落ち
   - pp.15 の 2 次方程式の解法（×）で桁落ちを確認せよ。
   - 資料とは別の数値例を作れ。

*3. 丸め誤差、桁落ちが問題となる数値計算の例を作り、確認せよ。

グラフの描き方のヒント

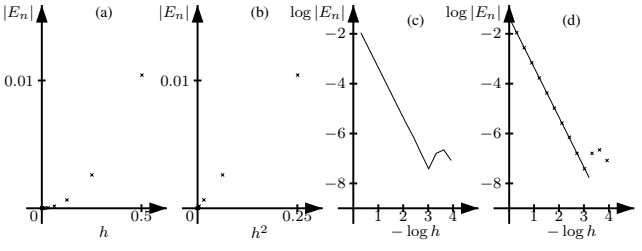
次頁の表は、(5) 式の積分を (6) 式に基づいて計算した結果である。表のままでは、刻み幅を小さくしていくと誤差がどのように変わっていくのかわかりにくいので、グラフにすべきである。
実験の結果をグラフにまとめるときは、一目でグラフの特徴がわかるように描かねばならない。次のページの 4 つのグラフは、上の結果を $h = 1/n$ と $|E_n|$ に関してプロットしたものである。どのグラフが、この実験の結果としてふさわしいか? また、ふさわしくないグラフは何がいけなかったのか? 考えてみよう。

cf. 能動的に変化させている変数は $h = 1/n$ なので、h を横軸にとる。さらに、n は 2 のべき乗なので h はコンピュータ内で丸め誤差はない。

台形則による $I = \int_0^1 \frac{1}{1+x^2} dx$ の計算

n	I_n	$E_n = I - I_n$
2	0.77499998	-1.03981872e-02
4	0.78279412	-2.60404547e-03
8	0.78474712	-6.51039679e-04
16	0.78523540	-1.62758429e-04
32	0.78535748	-4.06881167e-05
64	0.78538811	-1.00513293e-05
128	0.78539562	-2.54114403e-06
256	0.78539747	-6.93400042e-07
512	0.78539801	-1.56958239e-07
1024	0.78539813	-3.77489497e-08
2048	0.78539801	-1.56958239e-07
4096	0.78539795	-2.16562884e-07
8192	0.78539824	8.14603398e-08

4つのグラフ



グラフによる考察

前ページの 4 つのグラフの中で、実験結果の表示として適切なものは、唯一 (d) だけである。

台形則の離散化誤差の絶対値 $|E_h|$ は刻み幅 h の 2 乗に比例することが理論的に言えるので、$-\log h$ に対して $\log|E_n|$ をプロットすると傾き -2 の直線になるハズである。実際、グラフ (d) から h が 10^{-3} くらいまでは理論通りの結果になっていることが一目でわかる。しかしながら、離散化誤差が $\pi/4$ に対して相対的にマシンイプシロン程度になるあたりから誤差は増大に転じている。これらの考察から、この現象は丸め誤差の蓄積が原因と考えられる。

グラフの書き方

このように、適切なグラフは、現象の正しい理解に結びつく。しかしながら、(a) や (b) のグラフから、このような考察をすることは不可能である。グラフ (c) は、いくつかの問題点を含んでいる。まず、h は $0.5, 0.25, 0.125, \ldots$ と離散的な値しかとっていないにも関わらず、連続的に変化しているような印象を与える。それでは、グラフ (d) の $\times$ 印を (c) に重ねたらよいかというとそうでもない。この場合、隣同士の点を直線で結んだ折れ線に意味がないからである。理論的な予想は、各点が傾き -2 の直線上に乗るというものであり、当然、各点は誤差を含むと考えられる。よって、直線を引く場合は傾き -2 の直線に乗っていると思われる点を選んで、最小 2 乗法などを使って一本の直線を引くべきである。このとき、別の要因 (丸め誤差の蓄積など) のため、理論に乗っていないと思われる点 (この例の場合 $h < 0.001$ の点) は計算に含めてはいけない。

対数グラフの読み方

ダイナミックレンジの広い現象を展望するときは、対数グラフが活躍する。刻み幅を指数的に小さくしていったときの誤差の振る舞いなどは、対数をとらなければ観察することはほとんど不可能である。

さらに、台形則の実験結果のように、対数グラフで表すと直線関係になる現象は多い。そのときグラフからどのようなことがいえるのか? 高校数学の復習になるが、工学部の常識として改めて復習しておきたい。

対数グラフの読み方

変数 x と変数 y の対数をとると直線関係になったとすると、

$$\log y = \alpha \log x + \beta$$

である。ここで、α, β はそれぞれ直線の傾きと Y 切片である。$\beta = \log c$ $(c = 10^{\beta})$ とおくと

$$\log y = \log cx^{\alpha}$$

となる。よって、

$$y \propto x^{\alpha}$$

という関係、すなわち、y は x の α 乗に比例するという関係が導かれる。

先ほどの台形則の結果が、傾き -2 の直線になるのはそのためである。

—2. 非線形方程式の解法 I, II—

二分法, ニュートン法

非線形方程式 (1 次元の場合のみを考える)

$$f(x) = 0$$

線形の場合を除けば、一般に反復を伴う。
成功は解の初期値の良否にかかっている。
“計算の目的は洞察であって数値ではない ”
関数の形を考えて、根を囲い込む
($f(l)f(r) < 0$ なる区間 (l, r)) を見つける)
多重根や非常に接近した根は厄介
$f(x)$ が連続で $f(l)f(r) < 0$ $(l < r)$ ならば、
区間 (l, r) 内に少なくとも 1 個の根が存在する。(中間値の定理)

二分法 (bisection method)

根を含む区間の幅を半分にしていく, 失敗のない方法
n 回目の繰り返しおける根を含む区間の幅を ε_n とすると

$$\varepsilon_{n+1} = \frac{1}{2}\varepsilon_n$$

よって、許容誤差 ε まで到達するのに必要な反復数は

$$N = \log_2 \frac{\varepsilon_0}{\varepsilon}$$

繰り返しごとに誤差幅が前回の誤差幅の定数 (< 1) 倍になる方法は 1 次収束 (linear convergence) するという。(指数関数的、等比数列的)
cf. 超 1 次収束 (superlinear convergence)

$$\varepsilon_{n+1} = C(\varepsilon_n)^m, \quad m > 1$$

収束判定基準

- 絶対誤差 $|f(x)| < \varepsilon$
  $x \sim 1$ と $x \sim 10^{26}$ では達成可能な ε は異なる
- 相対誤差 $|f(x)/x| < \varepsilon$
  $x \sim 0$ では不可能
- 例えば許容幅は絶対誤差で指定し、$\varepsilon = \dfrac{\varepsilon_m(l_0 + r_0)}{2}$ とする。ε_m はマシンイプシロン, l_0, r_0 は最初の囲い込みの端点。
- 根が 0 付近の場合は、その関数について妥当な許容範囲の意味をよく考える必要がある。

二分法のプログラム

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  #define JMAX    64
 4  #define EPS 1.110223e-16
 5  double func(double x) {return tanh(x)+0.2*x+0.3;}
 6  double bisec(double (*func)(double),
 7               double xl, double xr, double xa)
 8  {
 9      int     j;
10      double  xmid, f, fmid, dx, x;
11
12      f = (*func)(xl);        fmid = (*func)(xr);
13      if (f*fmid >= 0.0) {
14          fprintf(stderr,"f(l)*f(r) must be negative.\n");
15          return 0.0;
16      }
17      x = f < 0.0 ? (dx = xr-xl, xl) : (dx = xl-xr, xr);
18      for (j=0; j<JMAX; j++) {
19          fmid = (*func)(xmid=x+(dx*=0.5));
20          if (fmid <=0.0) x = xmid;
21          if (fabs(dx)<xa||fmid==0.0) return x;
22      }
23      fprintf(stderr,"Too many bisections\n");
24      return 0.0;
25  }
26  int main()
27  {
28      double  a=-1.0,b=0.0;
29      printf("%le\n",bisec(func,a,b,fabs(EPS*(a+b)/2)));
30      return 0;
31  }
32
```

ニュートン法 (Newton method)

- 方程式 $f(x) = 0$ の数値解法.
- 非常に収束の早い方法 (2 次収束).
- 根の適当な近似値 x_0 からはじめて,

$$(9)\ x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \quad \text{for} \quad k = 0, 1, 2, \ldots$$

を繰り返し, 所望の精度が得られたら終わり.

ニュートン法の収束性

方程式の根を α ($f(\alpha) = 0$), 根 α の近似値を x_k, その誤差を $\delta_k = x_k - \alpha$ とする.
$f(x_k) = f(\alpha + \delta_k)$, $f'(x_k) = f'(\alpha + \delta_k)$ の Taylor 展開:

$$(10)\qquad f(\alpha+\delta_k)=f'(\alpha)\delta_k + \frac{1}{2!}f''(\alpha)\delta_k^2 + \mathcal{O}(\delta_k^3)$$

$$(11)\qquad f'(\alpha+\delta_k)=f'(\alpha) + f''(\alpha)\delta_k + \mathcal{O}(\delta_k^2)$$

ニュートン法の収束性

前ページの式より,

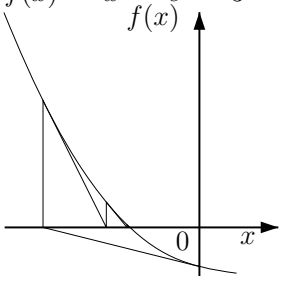
$$\delta_{k+1}=\delta_k - \frac{f(\alpha+\delta_k)}{f'(\alpha+\delta_k)} = \frac{\frac{1}{2}f''(\alpha)\delta_k^2 + \mathcal{O}(\delta_k^3)}{f'(\alpha) + f''(\alpha)\delta_k + \mathcal{O}(\delta_k^2)}$$

$$(12)\quad =\frac{f''(\alpha)}{2f'(\alpha)}\delta_k^2 + \mathcal{O}(\delta_k^3)$$

δ_{k_1} は δ_k の二乗に比例して小さくなる。(正しい桁数が 1 ステップ毎に倍に増える。) これを 2 次収束という.

ニュートン法の計算結果

$f(x) = x^2 - e^x - 3 = 0$ のニュートン法による解



ニュートン法の計算結果

$f(x) = x^2 - e^x - 3 = 0$ のニュートン法による解

k	x_k	$f(x_k)$	$f'(x_k)$
0	0.00000000	-4.00000000e+00	-1.00000000
1	-4.00000000	1.29816844e+01	-8.01831564
2	-2.38099609	2.57668395e+00	-4.85445062
3	-1.85020813	2.66065676e-01	-3.85762070
4	-1.78123668	4.37439798e-03	-3.73090309
5	-1.78006421	1.25888664e-06	-3.72875573
6	-1.78006387	1.04360964e-13	-3.72875511

ニュートン法のプログラム

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  #define EPS 1.0E-10
 4  #define X0  0.0
 5  double  f(double x) { return x*x-exp(x)-3.0; }
 6  double  g(double x) { return 2.0*x-exp(x); }
 7                        /* g(x)=f'(x) */
 8  main()
 9  {
10      double  f0,g0,x=X0;
11      int k=0;
12
13      do {
14          f0 = f(x);
15          g0 = g(x);
16          printf("%2d%12.8lf%16.8le%12.8lf\n",
17                  k++,x,f0,g0);
18          x -= f0/g0;
19      } while(fabs(f0)>=EPS);
20  }
```


休憩 — PostScript —

PostScript (PS) は多くのプリンタで使われている標準的なページ記述言語である。言語としてはスタック型のインタプリタ言語であり、(効率的とは言えないけれども) 簡単な数値計算も可能である。P.8 の絵は次頁のプログラムで表示している。根拠のある絵などを描きたいときに便利である。

休憩 — PostScript —

```
 1  %!PS-Adobe-2.0
 2  %%Title: newton.ps
 3  %%Creator: ikeda
 4  %%EndComments
 5
 6  /SCALE {10} def
 7  SCALE SCALE scale
 8  1.0 SCALE div setlinewidth
 9  40 20 translate
10  /f {dup dup mul exch 2.7182818 exch exp sub 3.0 sub} def
11  /g {dup 2.0 mul exch 2.7182818 exch exp sub} def
12  /fp {dup 4.0 mul exch f} def
13  /xp {x 4.0 mul 0} def
14
15  -25 0 moveto 5 0 lineto stroke
16  0 -10 moveto 0 40 lineto stroke
17  -5.0 fp moveto
18  -5.0 0.05 1.0 {fp lineto} for
19  stroke
20  /x {0.0} def
21  currentlinewidth 2 div setlinewidth
22  4 {xp moveto x fp lineto
23  x x f x g div sub /x exch def
24  xp lineto stroke
25  } repeat
26  showpage
```

演習問題(レポート)

4 二分法
- 二分法により $\tanh x + 0.2x + 0.3 = 0$ を解け。
- 収束までに必要な反復の回数はいくらと推定されるか? また、その値を実験結果と比較せよ。
- 1 次収束を確認せよ。

5 ニュートン法
- ニュートン法により $x^2 - e^x - 3 = 0$ を解け。
- 収束までに必要な反復の回数はいくらと推定されるか? また、その値を実験結果と比較せよ。
- 2 次収束を確認せよ。

*6 重根をニュートン法で求めるとき、収束性はどのようになるか?

—3. 非線形方程式の解法 III—

割線法 (Secant method)

- 非常に収束の早い方法 (1.6 次収束).
- ニュートン法における $f'(x)$ を差分により近似.

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} f(x_k) \text{ for } k = 1, 2, \ldots \quad (13)$$

- 出発点を 2 点 $x_0, x_1 \neq x_0$ 与える必要がある。

割線法の収束性

方程式の根を α, 近似値を x_k, 誤差を $\delta_k = x_k - \alpha$ とする.

$$\delta_{k+1} \sim L\delta_k\delta_{k-1}, \qquad L = \frac{f''(\alpha)}{2f'(\alpha)} \quad (14)$$

この数列を解くと,

$$|\delta_{k+1}| = \mathcal{O}(|\delta_k|^p), \qquad p = \frac{1+\sqrt{5}}{2} \sim 1.6 \quad (15)$$

休憩 — FORTRAN —

FORTRAN は科学技術計算用の言語として非常に長い歴史を持っており、その分、コンパイラの最適化や、スーパーコンピュータにおける並列計算のための技法なども発達している。長い歴史を持つだけに企業においては過去の資産の蓄積がありなかなか手放せない言語となっていた。情報関係の多くの分野ではもはや使うことはないと思われるが、このような歴史的事実を知っておくことは必要だろう。

休憩 — FORTRAN —

```
C     Secant Method
C     f(x) = tanh x + 0.2 x + 0.3
      PARAMETER (EPS = 1.0E-6)
      DATA X0/1.3/, X1/1.1/ K/0/
      F(X) = TANH(X)+0.2*X+0.3
      WRITE(6,200)
      F0 = F(X0)
      WRITE(6,100) K, X0, F0
      K = K + 1
   10 F1 = F(X1)
      WRITE(6,100) K, X1, F1
      IF (ABS(F1) .LT. EPS) GOTO 20
      X2 = X1 - (X1-X0)/(F1-F0)*F1
      X0 = X1
      X1 = X2
      F0 = F1
      K = K + 1
      GOTO 10
   20 STOP
  100 FORMAT(' ',I4,2E18.10)
  200 FORMAT('    K',8X,'X',17X,'F')
      END
```


割線法のプログラム

```
#include <stdio.h>
#include <math.h>

#define EPS 1.0E-15
#define X0  1.3
#define X1  1.1

double  f(double x) { return tanh(x)+0.2*x+0.3; }

main()
{
    double  f0, f1, f2;
    double  x0=X0, x1=X1, x2;
    int     k=0;

    f0 = f(x0);
    printf("%2d%16.12lf%16.8le\n",k++,x0,f0);
    do {
        f1 = f(x1);
        printf("%2d%16.12lf%16.8le\n",k++,x1,f1);
        x2 = x1 - (x1-x0)/(f1-f0)*f1;
        x0 = x1;
        x1 = x2;
        f0 = f1;
    } while(fabs(f0)>=EPS);
}
```


割線法の出力結果

k	x_k	f_k
0	1.300000000000	1.42172316e+00
1	1.100000000000	1.32049902e+00
2	-1.509059565607	-9.08583931e-01
3	-0.445595510593	-2.07390895e-01
4	-0.131056217236	1.43477749e-01
5	-0.259678111862	-5.93005751e-03
6	-0.254573055763	-1.27174662e-04
7	-0.254461174514	1.37164964e-07
8	-0.254461295054	-3.14802497e-12
9	-0.254461295051	-1.50162001e-17

演習問題(レポート)

7 割線法
- (14) 式において "$\sim$" の代わりに "$=$" が成り立っているとして数列を解き, (15) 式を導け. (ヒント: 両辺の log をとってみよ.)
- * (14) 式を導け.
- 方程式 $\tanh x + 0.2x + 0.3 = 0$ の根を割線法を用いて求めよ.
- 誤差がどのように減衰して行くかを示し, p 次収束を確認せよ.
- 数値微分は一般には, 桁落ちなどのために精度は悪くなる. 割線法では, 数値微分によって導関数を近似しているが, その影響はどうなっているか? 本手法の精度に関して考察せよ.

—4. 数値積分 I—

台形則 (Trapezoidal rule)

- **定積分** $I = \int_a^b f(x)dx$ **の数値解法**
- **積分区間** $[a, b]$ **を** N **等分して**, $h = (b-a)/N$ **とおき**, I **を和**

$$(16)\ I_N = h\left[\frac{1}{2}f(a) + \sum_{n=1}^{N-1} f(a+nh) + \frac{1}{2}f(b)\right]$$

で近似.

- N **を大きくしていき**, **十分な精度が得られた時点で終了**.

計算量の軽減

最も計算時間を費やす部分は, $f(x_n)$ **の計算**.
N **を大きくしていく際に**, **既に計算した値を再利用**.

$$(17)\quad I_{2N} = \frac{h}{2}\left[\frac{1}{2}f(a) + \sum_{n=1}^{2N-1} f\left(a+\frac{h}{2}nh\right) + \frac{1}{2}f(b)\right]$$

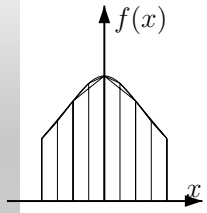
であるから,

$$(18)\qquad I_{2N} = \frac{1}{2}\left[I_N + J_N\right]$$

$$(19)\qquad J_N = h\sum_{n=1}^{N} f\left(a+\left(n-\frac{1}{2}\right)h\right)$$

によって求めればよい.

台形則の結果




```
    n    I_n           I_n-I_n/2
    2  3.00000000 -1.00000000e+00
    4  3.10000000 -1.00000000e-01
    8  3.13117647 -3.11764706e-02
   16  3.13898849 -7.81202390e-03
   32  3.14094161 -1.95311755e-03
   64  3.14142989 -4.88281134e-04
  128  3.14155196 -1.22070311e-04
  256  3.14158248 -3.05175781e-05
  512  3.14159011 -7.62939453e-06
 1024  3.14159202 -1.90734864e-06
 2048  3.14159249 -4.76837155e-07
 4096  3.14159261 -1.19209287e-07
 8192  3.14159264 -2.98023228e-08
16384  3.14159265 -7.45058371e-09
```


台形則のプログラム

```
 1  /*  Trapezoidal Rule    */
 2  #include <stdio.h>
 3  #include <math.h>
 4
 5  #define A   -1.0
 6  #define B    1.0
 7  #define KMAX    14
 8
 9  double f(double x) {return 2.0/(x*x+1.0);}
10
11  main()
12  {
13      int n = 1, j, k;
14      double  h, s, s1, x;
15
16      s = (B-A)*(f(A)+f(B))/2.0;
17      for (k=0; k<KMAX; k++) {
18          n *= 2;
19          h = (B-A)/n;
20          s1 = 0.0;
21          for (j=1; j<n; j+=2) {
22              x = A + h * j;
23              s1 += f(x);
24          }
25          s1 = s/2.0 + s1*h;
26          printf("%5d%12.8lf%16.8le\n",n,s1,s-s1);
27          s = s1;
28      }
29  }
```


演習問題 (レポート)

8 **台形則**

- * **離散化誤差は刻み幅の** 2 **乗に比例することを示せ。**
  ヒント: **オイラー・マクローリンの和公式を使え。**
  別解: **区間** $[c, c+h]$ **における** $f(x)$ **を** 2 **次式で近似しておき、**$(f(c)+f(c+h))/2$ **と**
  $f(c)/2 + f(c+h/2) + f(c+h)/2$ **を比較せよ。**
- (18), (19) **式をもとに台形則のプログラムを作り、**

$$I = \int_{-1}^{1} \frac{2}{1+x^2}dx$$

を求めよ。

- **離散化誤差** $I_N - I$ **が** h^2 **に比例することを確認せよ。**

—5. 数値積分 II—

Romberg **積分**

- **数列の収束の型がわかるとその知識を利用して収束を加速できることがある。**
- **数列の収束の加速方法としては、**Richardson **補外、**Aitken **法、**ε **算法などが有名である。**
- Romberg **積分は台形則に** Richardson **補外を適用したものである。**
- **刻み幅を**

$$h,\ h/2,\ h/2^2,\ \ldots$$

と減らしながら定積分を計算したり微分方程式を解いたりするときには、誤差が h **のべき級数で表されることが多い。そのような場合は、**Richardson **補外が有効である。**

Richardson 補外

$n \to \infty$ **のとき**

$$a_n = a + b_1\lambda_1^n + b_2\lambda_2^n + b_3\lambda_3^n + \cdots$$

と漸近的に表されるような数列 a_n **で、**$\lambda_1, \lambda_2, \ldots$ **の値が前もって判っているとき、**

$$a_n^{(1)} := \frac{a_n - \lambda_1 a_{n-1}}{1-\lambda_1} = a + b_2^{(1)}\lambda_2^n + \cdots$$

となって、a_n **よりは速く** a **に収束する。さらに、**

$$a_n^{(2)} := \frac{a_n^{(1)} - \lambda_2 a_{n-1}^{(1)}}{1-\lambda_2} = a + b_3^{(2)}\lambda_3^n + \cdots$$

と一層速く収束する。

Romberg 積分

刻み幅を $h = (b-a)/N$, **定積分と台形則による近似を**

$$I = \int_a^b f(x)dx$$

$$I_N = h\left[\frac{1}{2}f(a) + \sum_{n=1}^{N-1} f(a+nh) + \frac{1}{2}f(b)\right]$$

とすると, $f \in \mathrm{C}^{2p+2}[a, b]$ **ならば**, Euler-Maclaurin **の公式**

$$I_N - I = c_2h^2[f'(b) - f'(a)] + c_4h^4[f'''(b) - f'''(a)] + \cdots$$
$$(20)\qquad + c_{2p}h^{2p}[f^{(2p-1)}(b) - f^{(2p-1)}(a)] + \mathcal{O}(h^{2p+2})$$

が成り立つ. **ここで**, c_i **は** h **や** f **に依存しない定数である。**

Romberg 積分

(20) 式より, $I_N - I$ は h^2 で小さくなることがわかる。

$$I_N^{(1)} = \frac{I_N - I_{N/2}/4}{1 - 1/4}$$

とすれば、$I_N^{(1)}$ は I のより高精度な近似となり、(20) 式より $I_N^{(1)} - I$ は h^4 で小さくなることがわかる。
この操作を $k = 2, 3, \ldots$ と繰り返し行い、

$$I_N^{(k)} = \frac{I_N^{(k-1)} - I_{N/2}^{(k-1)}/4^k}{1 - 1/4^k}$$

を作れば、いくらでも高精度な公式を得る。

数列 $I_1, I_2, I_4, I_8, \ldots, I_{2^p}, \ldots$ に Richardson 補外を適用

Romberg 積分のプログラム

```
 1  /*  Romberg Integral  */
 2  #include <stdio.h>
 3  #include <math.h>
 4  #define KMAX    5
 5  double f(double x) {return exp(x);}
 6  main()
 7  {
 8      int     n = 1, j, k;
 9      double  h, s, t, r, x, sn[KMAX];
10      const double A=0.0, B=1.0;
11      sn[0] = s = (B-A)*(f(A)+f(B))/2.0;
12      printf("%3d %14.12lf\n",n,s);
13      for (k=1,n=2; k<KMAX; k++,n*=2) {
14          h = (B-A)/n;
15          for (j=1, s=0.0; j<n; j+=2) {
16              x = A + h * j;
17              s += f(x);
18          }
19          s = sn[0]/2.0 + s*h;
20          for (j=0, r=1.0; j<=k; j++) {
21              t = sn[j];
22              sn[j] = s;
23              r *= 4.0;
24              s = (r*s-t)/(r-1.0);
25          }
26          printf("%3d",n);
27          for (j=0; j<=k; j++)
28              printf(" %14.12lf",sn[j]);
29          printf("\n");
30      }
31  }
```


Romberg 積分の実行結果

(誤差 $I_{2^p}^{(p)} - I$ だけを表示)

```
 1 1.41e-01
 2 3.56e-02 5.79e-04
 4 8.94e-03 3.70e-05 8.59e-07
 8 2.24e-03 2.33e-06 1.38e-08 3.35e-10
16 5.59e-04 1.46e-07 2.16e-10 1.34e-12 3.33e-14
32 1.40e-04 9.10e-09 3.39e-12 5.77e-15 4.44e-16 4.44e-16
```


演習問題(レポート)

9 Romberg 積分

- $I_N^{(1)} - I$ は h^4 で小さくなることを示せ。
- Romberg 積分を用いて次の定積分を求めよ。

$$I = \int_0^1 e^x dx$$

(最低でも 1 段階は加速を行うこと。
余裕のあるものは $I_{2^p}^{(p)}$ を求めるプログラムを作れ。)

- 上の積分の収束性を確認せよ。

10* 数列の加速方法として Aitken 法というものがある。
Aitken 法について調べよ。

—6 常微分方程式の解法 I—

Euler 法

常微分方程式の初期値問題とは, 微分方程式:

(21) $$\frac{dx}{dt} = f(x,t) \qquad (a \le t \le b)$$

及び, 初期条件:

(22) $$x(a) = x_0$$

を満たす未知関数 $x(t)$ $(a \le t \le b)$ を求めるという問題。区間 $[a,b]$, 関数 $f(x,t)$, 及び, 初期値 $x(t)$ を与えると問題が定まる。
独立変数 t を時刻と呼ぶこともある。

連立微分方程式の場合は, $x(t)$, $f(x,t)$ はベクトル。高階の微分方程式も連立の 1 階微分方程式にすればよい。

Euler 法

- 最も原始的で簡単な方法
- 刻み幅を h として, 離散的な時刻:

  (23) $$t_n = a + nh \qquad (n = 0, 1, 2, \ldots)$$

  における (21) 式の解 $x(t_n)$ の近似値 x_n を,

  (24) $$x_{n+1} = x_n + hf_n, \quad f_n = f(x_n, t_n)$$

  によって逐次的に求める方法

Euler 法の精度

公式 (24) は (21) 式を Taylor の定理より,

(25) $x(t_{n+1}) = x(t_n) + hx'(t_n) + \frac{h^2}{2!}x''(\xi),\ t_n \le \exists\xi \le t_{n+1}$

と書き直したときの右辺第 2 項までをとったもの。
公式 (24) に微分方程式 (21) の真の解を代入したときの誤差,

(26) $$\tau_{n+1} = \frac{x(t_{n+1}) - x(t_n)}{h} - f_n(x(t_n), t_n)$$

を (24) の局所打ち切り誤差という.

Euler 法の精度

Euler 法の場合,

(27) $$\tau_{n+1} = \frac{1}{2}hx''(\xi) = \mathcal{O}(h)$$

となるので Euler 法の精度は 1 であるという。
近似解 x_n の真の解 $x(t_n)$ からの誤差は h に比例する。

高階微分方程式の場合

次式で表される 3 階の常微分方程式:

$$\frac{d^3}{dt^3}y(t)+2\frac{d^2}{dt^2}y(t)+2\frac{d}{dt}y(t)+y(t)=1 \tag{28}$$

を, 初期条件:

$$y(0)=\frac{d}{dt}y(0)=\frac{d^2}{dt^2}y(0)=0 \tag{29}$$

のもとで, 時刻 0 から 10 まで解きたい.

高階微分方程式の場合

(28) 式は高階の常微分方程式であるので, 1 階の連立常微分方程式にするため,

$$x_1(t)=y(t),\quad x_2(t)=\frac{d}{dt}y(t),\quad x_3(t)=\frac{d^2}{dt^2}y(t) \tag{30}$$

とおくと, (28) 式は,

$$\frac{d}{dt}\begin{pmatrix}x_1(t)\\x_2(t)\\x_3(t)\end{pmatrix}=\begin{pmatrix}x_2(t)\\x_3(t)\\1-x_1(t)-2x_2(t)-2x_3(t)\end{pmatrix} \tag{31}$$

と書き直すことができ, (21) 式の形となる.

高階微分方程式の場合

実は微分方程式 (28) は解析的に解くことができ, その解は,

$$y^*(t)=1-e^{-t}-\frac{2}{\sqrt{3}}e^{-\frac{1}{2}t}\sin\frac{\sqrt{3}}{2}t \tag{32}$$

で与えられる.

休憩 — TEX —

TEX は D. E. Knuth によって開発された組み版処理用の高機能言語である。数式などの出力の美しさや記述の容易さは他の組み版システムの追随を許さない。論理的な文章作成のための機能も持っており、投稿論文として TEX の原稿を標準としている学会も多い。TEX は強力なマクロ展開機能を持っており、プログラミング言語としても大変興味深い機能を備えている。次ページの図は、実際に Euler 法を用いて微分方程式 (28) を解いた結果である。決して効率的とは言えないけれども、四則演算からなる簡単な数値計算ならば可能である。

休憩 — TEX(つづき) —

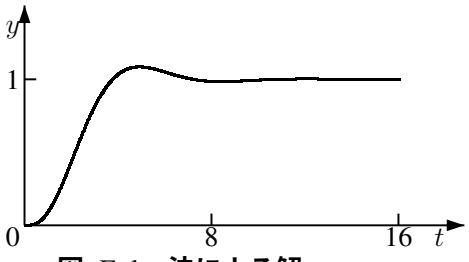


図. Euler 法による解

上図を表示させた LATEX のソースファイルを次ページに示す。

休憩 — TEX(つづき) —

```
\documentclass{slides}
\begin{document}
\newdimen\y\y=0pt\newdimen\v\v=0pt\newdimen\a\a=0pt
\newcount\t\newdimen\adot\newdimen\tp\newdimen\x\newbox\markbox
\setbox\markbox\hbox{\vrule height0.5pt depth0.5pt width1pt}
\def\mark{\raise\y\hbox to0pt{\hskip\x\unhcopy\markbox\hss}}
\def\whilenop#1{} % No operation
\def\whilenum#1\do#2{\ifnum#1\relax#2\relax
  \iwhilenum{#1\relax#2\relax}\fi}
\def\iwhilenum#1{\ifnum #1\let\nextwhile=\iwhilenum
  \else\let\nextwhile=\whilenop\fi\nextwhile{#1}}
\begin{slide}
\tiny\unitlength=1pt
\begin{picture}(316,166)
\put(16,16){\vector(1,0){300}}\put(16,16){\vector(0,1){150}}
\put(0,0){\makebox(16,16){0}}
\put(300,0){\makebox(0,16){$t$}}
\put(0,150){\makebox(16,0){$y$}}
\put(272,0){\makebox(0,16){16}}\put(272,16){\line(0,1){8}}
\put(144,0){\makebox(0,16){8}}\put(144,16){\line(0,1){8}}
\put(0,116){\makebox(16,0){1}}\put(16,116){\line(1,0){8}}
\put(16,16){\whilenum{\t<1024}\do{\x=0.25pt\multiply\x\t
\adot=100pt\advance\adot by-\y
\tp=\v\multiply\tp by-2\advance\adot by\tp
\tp=\a\multiply\tp by-2\advance\adot by\tp
\tp=\v\divide\tp by64 \advance\y by\tp
\tp=\a\divide\tp by64 \advance\v by\tp
\divide\adot by64 \advance\a by\adot\mark\advance\t by1}}
\end{picture}
\end{slide}
\end{document}
```


Euler 法のプログラム

次ページに示す Euler 法の主要部分を

```
    x0[0]=x0[1]=x0[2]=0.0;
    for (i=2; i<64; i*=2)
        loop(x0,i);
```

などと呼び出し、刻み幅を小さくしながら繰り返す。
プログラム中の plot, setplot は X Window 上に表示させるのであれば、

```
static int    cx, cy;
void setplot(y,t)
double y,t;
{
    t *= TSCALE;
    y *= YSCALE;
    cx = (int)t+TORG;
    cy = YORG-(int)y;
}
void plot(y,t)
double y,t;
{
    int    px=cx;
    int    py=cy;
    setplot(y,t);
    XDrawLine(disp,win,gc,px,py,cx,cy);
    XDrawLine(disp,pix,gc,px,py,cx,cy);
}
```

とでもしておけば良いだろう。
全プログラムは次の URL を参照せよ。

http://www-b2.is.tokushima-u.ac.jp/~ikeda/num/

Euler 法のプログラム

```
void func();

loop(x0,kmax)
double *x0;
int    kmax;
{
    double    x[N],f[N],t,h;
    int       k,i;

    for (i=0; i<N; i++)
        x[i]=x0[i];
    t = TMIN;
    h = 1.0/kmax;
    kmax *= TMAX-TMIN;
    setplot(x[0],t);
    for (k=0; k<kmax; k++) {
        t += h;
        func(x,f);
        for (i=0; i<N; i++)
            x[i]+=h*f[i];
        plot(x[0],t);
    }
}

void func(x,f)    /* f(x,t)=f(x) */
double    *x, *f;
{
    f[0]=x[1];
    f[1]=x[2];
    f[2]=1.0 - 1.0*x[0] - 2.0*x[1] - 2.0*x[2];
}
```

演習問題(レポート)

11 Euler 法

- (32) 式を導け。
- 刻み幅 h の Euler 法によって求めた数値解を $y_h(t)$ と表し、(32) 式で求めた真の解との誤差 E_h を次式で定義する。

$$E_h = \max |y^*(t_n) - y_h(t_n)|$$

プログラムを改良して各刻み幅における数値誤差 E_h を求め、h と E_h の関係を考察せよ。

- 真の解がわからない場合に数値計算によって得た解の誤差がどの程度であるかを知るにはどのようにしたらいいか?

—7 常微分方程式の解法 II,III—

- Euler 法の改良方法として、修正 Euler 法、Runge-Kutta 法を取り上げる。
- Euler 法と同様、(23) 式の離散的な時刻 t_n における解 $x(t_n)$ の近似値 x_n を次式で逐次計算する方法
- 修正 Euler 法は 2 次の公式、Runge-Kutta 法は 4 次の公式である。
- さらに高度な方法として、6 段 5 次のフェールベルグ公式で、ステップ幅を自動調節する方法などもある。

修正 Euler 法

$$\left.\begin{array}{rr} \tilde{x}_n{=}x_n + hf_n, & f_n{=}f(x_n,t_n), \\ x_{n+1}{=}x_n + \frac{h}{2}(f_n + \tilde{f}_n), & \tilde{f}_n{=}f(\tilde{x}_n,t_{n+1}), \end{array}\right\} \tag{33}$$

修正 Euler 法は 2 次の公式。すなわち, 誤差は $\mathcal{O}(h^2)$.

Runge-Kutta 法

$$\left.\begin{array}{rr} x_n^1{=}x_n + \frac{h}{2}f_n^1, & f_n^1{=}f(x_n,t_n), \\ x_n^2{=}x_n + \frac{h}{2}f_n^2, & f_n^2{=}f(x_n^1,t_n+\frac{h}{2}), \\ x_n^3{=}x_n + hf_n^3, & f_n^3{=}f(x_n^2,t_n+\frac{h}{2}), \\ x_{n+1}{=}x_n + \frac{h}{6}(f_n^1 + 2f_n^2 + 2f_n^3 + f_n^4). & f_n^4{=}f(x_n^3,t_{n+1}), \end{array}\right\} \tag{34}$$

Runge-Kutta 法の次数は 4 次。すなわち, 誤差は $\mathcal{O}(h^4)$.

2 重振子の振舞い

質量 μ 長さ l の一様な棒 0 及び棒 1 が図のように接続されている。各棒の重心回りの慣性モーメントは $I_0 = I_1 = \frac{1}{12}\mu l^2$ とし、摩擦や空気の抵抗は無視する。

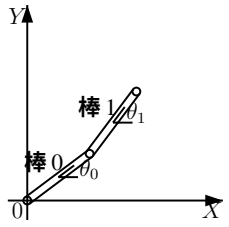


図　2 重振子

2 重振子の振舞い(つづき)

棒 0 及び棒 1 が水平面となす角をそれぞれ θ_0, θ_1 として運動方程式をたてると、以下の微分方程式が導出される。

$$\mu l^2 \left\{\frac{4}{3}\ddot{\theta}_0 + \frac{1}{2}\cos(\theta_1-\theta_0)\ddot{\theta}_1 - \frac{1}{2}\sin(\theta_1-\theta_0)\dot{\theta}_1^2\right\} + \frac{3}{2}\mu g l\cos\theta_0 = 0 \tag{35}$$

$$\mu l^2 \left\{\frac{1}{2}\cos(\theta_1-\theta_0)\ddot{\theta}_0 + \frac{1}{3}\ddot{\theta}_1 + \frac{1}{2}\sin(\theta_1-\theta_0)\dot{\theta}_0^2\right\} + \frac{1}{2}\mu g l\cos\theta_1 = 0 \tag{36}$$

休憩 — Java —

Java は C++に似たオブジェクト指向プログラミング言語である。プラットフォームに依存しない、マルチスレッド対応などの特徴をもち、WWW の普及とともにインターネット・アプリケーションのためのプログラミング言語として注目を浴びるようになった。ポインター操作に慣れてしまったプログラマにとってはまだるっこいと感じる部分もあるが、簡単なプログラムをインターネット上で手っ取り早く使ってもらうためには大変便利な言語である。

休憩 — Java(つづき) —

数理計画法の講義の Web ページ

`http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/`

の下に置いてある Dijkstra 法、Prim 法、Kruskal 法、Ford-Fulkerson 法、シンプレックス法、二段解法などの Java アプレットは、世界中からアクセスがあり好評を得ている。

前ページに紹介した 2 重振子の振る舞いは、下記をアクセスすれば見ることができる。

`http://www-b2.is.tokushima-u.ac.jp/~ikeda/num/Pendulum.html`

上記のアプレットでは、微分方程式を数値的に解くために Runge-Kutta 法を用いている。Java のソースファイルや運動方程式の導出が載っているので、興味のある人は調べてみよ。

演習問題(レポート)

12 修正 Euler 法による実験
- (28) 式を修正 Euler 法により微分方程式を解くプログラムを作れ.
- Euler 法の場合と同様に刻み幅と誤差の関係をグラフにプロットし, 傾きを求めよ.
- 刻み幅と誤差のグラフを比較し考察せよ.

13 Runge-Kutta 法による実験
- (28) 式を Runge-Kutta 法により微分方程式を解くプログラムを作れ.
- Euler 法の場合と同様に刻み幅と誤差の関係をグラフにプロットし, 傾きを求めよ.
- 刻み幅と誤差のグラフを比較し考察せよ.

14 Euler 法, 修正 Euler 法, Runge-Kutta 法において必要な計算量を比較せよ.

—8 連立1次方程式の解法— I

LU 分解

連立一次方程式 $Ax = b$, $A \in \mathrm{R}^{n\times n}$, $b \in \mathrm{R}^n$: given
計算の手間を如何に抑えるか?

cf. Cramer の公式 (A^{-1} を求める公式) 通りにやると計算の手間は $n!$ 回の乗除算と $(n-1)!$ 回の加減算 $\Rightarrow$ 使えない

cf. Gauss-Jordan の掃き出し法は比較的まとも
A^{-1} を求めるのに, n^3 回の乗除算と $n(n-1)^2$ 回の加減算
$A^{-1}b$ を求めるのに, n^2 回の乗除算と $n(n-1)$ 回の加減算

LU 分解: 行列 A を $A = LU$, L は下三角行列 (lower triangular), U は上三角行列 (upper triangular) と分解して、$Ly = b, Ux = y$ を解く方法。
$A = LU$ の分解に, 乗除算 : $(n-1)(n^2+n+3)/3$ 回、加減算 : $n(n-1)(2n-1)/6$ 回。$Ly = b$、$Ux = y$ を解くのに、乗除算 : n^2 回、加減算 : $n(n-1)$ 回。

しかも、作業領域をあまり必要としない。

LU分解を用いた連立一次方程式の解法

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} =$$
$$\begin{pmatrix} 1 & & & 0 \\ l_{21} & 1 & & \\ l_{31} & l_{32} & \ddots & \\ \vdots & \vdots & & 1 \\ l_{n1} & l_{n2} & \cdots & l_{n,n-1} \quad 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{2n} \\ & & \ddots & \vdots \\ 0 & & & u_{nn} \end{pmatrix}$$

もし，このように分解できたら、次ページのように連立一次方程式を解けばよい。

LU分解を用いた連立一次方程式の解法

$Ly = b$ の解.

$$\begin{aligned} y_1 &= b_1 \\ y_2 &= b_2 - l_{21}y_1 \\ y_3 &= b_3 - l_{31}y_1 - l_{32}y_2 \\ y_4 &= b_4 - l_{41}y_1 - l_{42}y_2 - l_{43}y_3 \\ &\vdots \\ y_{n-1} &= b_{n-1} - l_{n-1,1}y_1 - l_{n-1,2}y_2 - \cdots - l_{n-1,,n-2}y_{n-2} \\ y_n &= b_n - l_{n1}y_1 - l_{n2}y_2 - \cdots\cdots - l_{n,n-1}y_{n-1} \end{aligned}$$

$Ux = y$ の解.

$$\begin{aligned} x_n &= y_n/u_{nn} \\ x_{n-1} &= (y_{n-1} - u_{n-1,n}x_n)/u_{n-1,n-1} \\ x_{n-2} &= (y_{n-2} - u_{n-2,n}x_n - u_{n-2,n-1}x_{n-1})/u_{n-2,n-2} \\ &\vdots \\ x_2 &= (y_2 - u_{2,n}x_n - u_{2,n-1}x_{n-1} - \cdots - u_{23}x_3)/u_{22} \\ x_1 &= (y_1 - u_{1,n}x_n - u_{1,n-1}x_{n-1} - \cdots\cdots - u_{1,2}x_2)/u_{11} \end{aligned}$$

LU分解

$A = LU$ を具体的に書いてみる.

$$\begin{aligned} a_{11} &= u_{11} \\ a_{12} &= u_{12} & a_{22} &= l_{21}u_{12} + u_{22} \\ a_{13} &= u_{13} & a_{23} &= l_{21}u_{13} + u_{23} & a_{33} &= l_{31}u_{13} + l_{32}u_{23} + u_{33} \\ &\vdots & &\vdots & &\vdots & &\ddots \\ a_{1n} &= u_{1n} & a_{2n} &= l_{21}u_{1n} + u_{2n} & a_{3n} &= l_{31}u_{1n} + l_{32}u_{2n} + u_{3n} & &\cdots \end{aligned}$$

$$\begin{aligned} a_{21} &= l_{21}u_{11} \\ a_{31} &= l_{31}u_{11} & a_{32} &= l_{31}u_{12} + l_{32}u_{22} \\ a_{41} &= l_{41}u_{11} & a_{42} &= l_{41}u_{12} + l_{42}u_{22} & a_{43} &= l_{41}u_{13} + l_{42}u_{23} + l_{43}u_{33} \\ &\vdots & &\vdots & &\vdots & &\ddots \\ a_{n1} &= l_{n1}u_{11} & a_{n2} &= l_{n1}u_{12} + l_{n2}u_{22} & a_{n3} &= l_{n1}u_{13} + l_{n2}u_{23} + l_{n3}u_{33} & &\cdots \end{aligned}$$

LU分解

前ページの式を l_{ij}, u_{ij} に関して解くと、

$$\begin{aligned} u_{11} &= a_{11} \\ u_{12} &= a_{12} & u_{22} &= a_{22} - l_{21}u_{12} \\ u_{13} &= a_{13} & u_{23} &= a_{23} - l_{21}u_{13} & u_{33} &= a_{33} - l_{31}u_{13} - l_{32}u_{23} \\ &\vdots & &\vdots & &\vdots & &\ddots \\ u_{1n} &= a_{1n} & u_{2n} &= a_{2n} - l_{21}u_{1n} & u_{3n} &= a_{3n} - l_{31}u_{1n} - l_{32}u_{2n} & &\cdots \end{aligned}$$

$$\begin{aligned} v_{11} &= 1/u_{11} & v_{22} &= 1/u_{22} & v_{33} &= 1/u_{33} & &\cdots \end{aligned}$$

$$\begin{aligned} l_{21} &= a_{21}v_{11} \\ l_{31} &= a_{31}v_{11} & l_{32} &= (a_{32} - l_{31}u_{12})v_{22} \\ l_{41} &= a_{41}v_{11} & l_{42} &= (a_{42} - l_{41}u_{12})v_{22} & l_{43} &= (a_{43} - l_{41}u_{13} - l_{42}u_{23})v_{33} \\ &\vdots & &\vdots & &\vdots & &\ddots \\ l_{n1} &= a_{n1}v_{11} & l_{n2} &= (a_{n2} - l_{n1}u_{12})v_{22} & l_{n3} &= (a_{n3} - l_{n1}u_{13} - l_{n2}u_{23})v_{33} & &\cdots \end{aligned}$$

LU分解のプログラム(一部)

LU 分解と $Ly = b, Ux = y$ を解く部分は例えば以下のようになる。

```
#define N       5

ludecomp(double a[][N])
{
        int     i,j,k;
        double  w;

        for (k=0; k<N-1; k++) {
                w = 1/a[k][k];
                for (i=k+1; i<N; i++) {
                        a[i][k] *= w;
                        for (j=k+1; j<N; j++)
                                a[i][j] -= a[i][k]*a[k][j];
                }
        }
}

solve(double a[][N], double b[])
{
        int     i,k;

        for (k=1; k<N; k++)
                for (i=0; i<k; i++)
                        b[k] -= a[k][i]*b[i];
        for (k=N-1; k>=0; k--) {
                for (i=k+1; i<N; i++)
                        b[k] -= a[k][i]*b[i];
                b[k] /= a[k][k];
        }
}
```


3重対角行列のLU分解

LU 分解は A が特殊な構造をもつ場合に威力を発揮する.
例えば，3 重対角行列の逆行列をとると,

$$\begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}^{-1} = \begin{pmatrix} \frac{5}{6} & \frac{2}{3} & \frac{1}{2} & \frac{1}{3} & \frac{1}{6} \\ \frac{2}{3} & \frac{4}{3} & 1 & \frac{2}{3} & \frac{1}{3} \\ \frac{1}{2} & 1 & \frac{3}{2} & 1 & \frac{1}{2} \\ \frac{1}{3} & \frac{2}{3} & 1 & \frac{4}{3} & \frac{2}{3} \\ \frac{1}{6} & \frac{1}{3} & \frac{1}{2} & \frac{2}{3} & \frac{5}{6} \end{pmatrix}$$

のように，一般にはすべての要素が non-zero になる.

3重対角行列のLU分解

ところが、一般に、行列 L と U は A の構造を継承するという性質があり、構造的に 0 となる要素をあらかじめ考慮することによって、計算量、作業領域の大幅な節約が期待される。
3 重対角行列 A を LU 分解すると

$$\text{(37)} \quad \begin{pmatrix} \alpha_1 & \beta_1 & & 0 \\ \gamma_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ 0 & & \gamma_{n-1} & \alpha_n \end{pmatrix} = \begin{pmatrix} 1 & & & 0 \\ l_1 & 1 & & \\ & \ddots & \ddots & \\ 0 & & l_{n-1} & 1 \end{pmatrix} \begin{pmatrix} d_1 & u_1 & & 0 \\ & d_2 & \ddots & \\ & & \ddots & u_{n-1} \\ 0 & & & d_n \end{pmatrix}$$

3重対角行列のLU分解

実際、前ページの行列を LU 分解すると，

$$\begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix} \to \begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -\frac{1}{2} & \frac{3}{2} & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix} \to$$

$$\begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -\frac{1}{2} & \frac{3}{2} & -1 & 0 & 0 \\ 0 & -\frac{2}{3} & \frac{4}{3} & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix} \to \begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -\frac{1}{2} & \frac{3}{2} & -1 & 0 & 0 \\ 0 & -\frac{2}{3} & \frac{4}{3} & -1 & 0 \\ 0 & 0 & -\frac{3}{4} & \frac{5}{4} & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix} \to \begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -\frac{1}{2} & \frac{3}{2} & -1 & 0 & 0 \\ 0 & -\frac{2}{3} & \frac{4}{3} & -1 & 0 \\ 0 & 0 & -\frac{3}{4} & \frac{5}{4} & -1 \\ 0 & 0 & 0 & -\frac{4}{5} & \frac{6}{5} \end{pmatrix}$$

計算の途中においても，3 重対角行列のままである．
一般に逆行列を計算には n^2 の大きさのメモリが必要、3 重対角行列の LU 分解では $3n-2$ の大きさのメモリで十分。

演習問題(レポート)

16. LU 分解

- 次の方程式を LU 分解を用いて解け。

$$\begin{pmatrix} 2 & 1 & 2 & 1 & 2 \\ 4 & 5 & 5 & 5 & 5 \\ 6 & 9 & 10 & 10 & 10 \\ 8 & 13 & 15 & 18 & 16 \\ 10 & 17 & 20 & 26 & 24 \end{pmatrix} x = \begin{pmatrix} 6 \\ 14 \\ 26 \\ 39 \\ 54 \end{pmatrix}.$$

- 行列の大きさが $n \times n$ のとき, 本手法の計算の手間 (かけ算わり算の回数, 作業領域の大きさ) はそれぞれいくらか?

演習問題(レポート)

17. 3 重対角行列の LU 分解

- (37) 式を l_k, d_k, u_k に関して解け.
- (37) 式のように分解されたとして, $Ly = b, Ux = y$ を y, x に関して解け.
- ベクトル $\alpha \in \mathrm{R}^n, \beta \in \mathrm{R}^{n-1}, \gamma \in \mathrm{R}^{n-1}$, および, n を引数として渡すと, α, β, γ をそれぞれベクトル d, u, l に書き換えて返す C の関数 decomp を作れ。
- 上のプログラムを用いて次の連立 1 次方程式を解け.

$$\begin{pmatrix} 2 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 2 \end{pmatrix} x = \begin{pmatrix} 9 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

—9 連立1次方程式の解法 II—

LU 分解

- 特殊な構造をもつ行列を LU 分解する場合、その構造を考慮すると、計算の手間を軽減することができる。
- 対称行列を LU 分解する場合、修正 Cholesky 法を使うと、作業領域、計算量ともに約半分の手間で済む。
- 物理システムをモデル化すると正定対称行列がよく現れる。

対称行列のLU分解

対称行列 (symmetric matrix) を LU 分解すると

$$A = LU = LD\tilde{U}$$

ここで，$D = \mathrm{diag}[d_1, \ldots, d_n]$,

$$L = \begin{pmatrix} 1 & & \\ & \ddots & \\ l_{ij} & & 1 \end{pmatrix}, \ \tilde{U} = \begin{pmatrix} 1 & & \tilde{u}_{ij} \\ & \ddots & \\ & & 1 \end{pmatrix}$$

対称行列のLU分解

u_{ij} と $\tilde{u}_{ij}$ $(j > i)$ の関係は

$$\begin{pmatrix} u_{11} & & u_{ij} \\ & \ddots & \\ & & u_{nn} \end{pmatrix} = \begin{pmatrix} u_{11} & & \\ & \ddots & \\ & & u_{nn} \end{pmatrix} \begin{pmatrix} 1 & & \tilde{u}_{ij} \\ & \ddots & \\ & & 1 \end{pmatrix}$$

なので、$d_i = u_{ii}$.
$A = A^{\mathrm{T}}$ なら $\tilde{U}^{\mathrm{T}} = L$

$$\begin{aligned} A = A^{\mathrm{T}} &\Rightarrow LD\tilde{U} = \tilde{U}^{\mathrm{T}} D^{\mathrm{T}} L^{\mathrm{T}} = \tilde{U}^{\mathrm{T}} D L^{\mathrm{T}} \\ &\Rightarrow \tilde{U}^{\mathrm{T}} = L \\ &\Rightarrow A = LDL^{\mathrm{T}} \quad \text{修正 Cholesky 法} \end{aligned}$$

正定対称行列の場合

A が正定対称行列 (positive definite) であるための必用十分条件は $d_i > 0 \, \forall i \in \{1, \ldots, n\}$ となることである。

$$\begin{aligned} A = A^{\mathrm{T}} > 0 &\Leftrightarrow x^{\mathrm{T}} A x > 0, \quad \forall x \neq 0 \\ &\Leftrightarrow x^{\mathrm{T}} L D L^{\mathrm{T}} x > 0, \quad \forall x \neq 0 \\ &\Leftrightarrow y^{\mathrm{T}} D y > 0, \quad y = L^{\mathrm{T}} x, \quad \forall x \neq 0 \\ &\Leftrightarrow y^{\mathrm{T}} D y > 0, \quad \forall y \neq 0 \\ &\Leftrightarrow d_i > 0, \quad i = 1, \ldots, n \end{aligned}$$

修正 Cholesky 法の導出

便宜のため $l_{kk} = 1$ とおき，$u_{ij} = d_i l_{ji}$ であることを考慮し，$A = LU = LDL^{\mathrm{T}}$ を具体的に書き下すと，$k = 1, \ldots, n$ について，

$$a_{ki} = \sum_{j=1}^{i} l_{kj} u_{ji} = \sum_{j=1}^{i} l_{kj} d_j l_{ij}, \qquad i = 1, \ldots, k-1$$

$$a_{kk} = \sum_{i=1}^{k} l_{ki} u_{ik} = \sum_{i=1}^{k} l_{ki}^2 d_i$$

修正 Cholesky 法の導出

これを，解くと，

$$l_{ki} = \left(a_{ki} - \sum_{j=1}^{i-1} l_{kj} d_j l_{ij} \right) \Big/ d_i, \qquad i = 1, \ldots, k-1$$

$$d_k = a_{kk} - \sum_{i=1}^{k-1} l_{ki}^2 d_i$$

上の 2 式をプログラミングすれば修正 Cholesky 法となる。

修正 Cholesky 法のプログラム

次頁に示すプログラムの主要部分 decomp は，A 行列の左下部分を受取ると，L および D^{-1} を返す関数である．

$$\mathtt{decomp}: \begin{pmatrix} a_{11} & & \\ \vdots & \ddots & \\ a_{n1} & \ldots & a_{nn} \end{pmatrix} \longmapsto \begin{pmatrix} d_1^{-1} & & & \\ l_{21} & d_2^{-1} & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \ldots & d_n^{-1} \end{pmatrix}$$

修正 Cholesky 法の主要部分

```
#define A(i,j)  a[(i)*(i+1)/2+j]
decomp(double a[], int n)
{
    register int    i,j,k;
    double          *w = (double *)malloc(n*sizeof(double));
    for (k=0; k<n; k++) {
        for (i=0; i<k; i++) {
            w[i] = A(k,i);
            for (j=0; j<i; j++)
                w[i] -= w[j] * A(i,j);
            A(k,i) = w[i] * A(i,i);
        }
        for (i=0; i<k; i++)
                A(k,k) -= w[i] * A(k,i);
        A(k,k) = 1.0 / A(k,k);
    }
    free((void *)w);
}
solve(double a[], double b[], int n)
{
    register int    i,k;
    for (k=1; k<n; k++)
        for (i=0; i<k; i++)
            b[k] -= A(k,i) * b[i];
    for (k=n-1; k>=0; k--) {
        b[k] *= A(k,k);
        for (i=k+1; i<n; i++)
            b[k] -= A(i,k) * b[i];
    }
}
```


演習問題 (レポート)

18 対称行列の LU 分解

- 修正 Cholesky 法を用いて次の方程式を解け。

$$\begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix} x = \begin{pmatrix} 2 \\ 2 \\ 2 \\ 2 \\ 2 \end{pmatrix}.$$

- 行列の大きさが $n \times n$ のとき、本手法の計算の手間 (かけ算わり算の回数, 作業領域の大きさ) はいくらか?
- 上の行列は対称行列であり、かつ、3 重対角行列である。両方を考慮した LU 分解のプログラムを作成し、上の方程式を解け。また、そのときの計算の手間はいくらか?

—10 連立 1 次方程式の解法 III—

- $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ を LU 分解? $\Rightarrow$ 0 割り!!
- 0 割りを避けるため、行や列を適当に入れ換えながら LU 分解する。
- さらに、絶対値の大きな値でわり算を行うように行や列を入れ換えることによって、誤差の拡大を防ぐ効果がある。
- ここでは、操作が簡単な部分ピボット選択法 (Lu decomposition with partial pivoting) を取り上げる。

まずは、ガウスの消去法が LU 分解そのものであることから見てみよう。

cf. ガウスの消去法 (Gaussian Elimination)

$A \rightarrow$ 右上三角行列

$$A_k = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1k} & \ldots & a_{1n} \\ & a_{22}^{(2)} & a_{23}^{(2)} & \ldots & a_{2k}^{(2)} & \ldots & a_{2n}^{(2)} \\ & & a_{33}^{(3)} & \ldots & a_{3k}^{(3)} & \ldots & a_{3n}^{(3)} \\ & & & \ddots & & & \vdots \\ & & & & a_{kk}^{(k)} & \ldots & a_{kn}^{(k)} \\ & & & & \vdots & & \vdots \\ & & & & a_{nk}^{(k)} & \ldots & a_{n,n}^{(k)} \end{pmatrix}$$

のとき、A_{k+1} を次ページのように定義する。

cf. ガウスの消去法

$$A_{k+1} = M_k A_k$$

$$M_k = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -m_{k+1,k} & 1 & \\ & & \vdots & & \ddots \\ & & -m_{nk} & & 1 \end{pmatrix}$$

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \quad i = k+1, \ldots, n$$

ガウスの消去法とLU分解

$A_1 = A$ として $k = 1, \ldots, n-1$ について繰り返すと、

$$U = A_n = M_{n-1} \cdots M_2 M_1 A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ & a_{22}^{(2)} & \ldots & a_{2n}^{(2)} \\ & & \ddots & \vdots \\ & & & a_{n,n}^{(n)} \end{pmatrix}$$

$$L = M_1^{-1} M_2^{-1} \cdots M_{n-1}^{-1} = \begin{pmatrix} 1 & & & \\ m_{21} & 1 & & \\ \vdots & & \ddots & \\ m_{n1} & m_{n2} & \ldots & 1 \end{pmatrix}$$

となる。ガウスの消去法はLU分解そのものである。

LU decomp. with partial pivoting

さて、$m_{ik} = \dfrac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$ における0割りを避けるため、p_k を

$$a_{p_k k}^{(k)} = \max\left\{ \left|a_{kk}^{(k)}\right|, \left|a_{k+1,k}^{(k)}\right|, \ldots, \left|a_{n,k}^{(k)}\right| \right\}$$

となるように選び、P_k を次式で定義する。

$$P_k = \begin{array}{c} \\ k \\ p_k \end{array} \begin{array}{c} \begin{array}{cccccc} k & & & p_k & & \end{array} \\ \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 0 & 1 & & \\ & & 1 & 0 & & \\ & & & & \ddots & \\ & & & & & 1 \end{pmatrix} \end{array}$$

LU decomp. with partial pivoting

A_k の更新のとき、行の入れ換えを行ってから掃き出す。すなわち、置換行列 P_k を掛けてから M_k を掛けて、

$$A_{k+1} = M_k (P_k A_k)$$

とする (部分ピボット選択法 : LU decomposition with partial pivoting)
A が正則ならば，$a_{p_k k}^{(k)} \neq 0$ となる p_k が必ずとれる。

LU decomp. with partial pivoting

上の手順は A の行を適当に入れ換えた行列をLU分解している

$$U = A_n = M_{n-1} P_{n-1} M_{n-2} P_{n-2} \cdots M_2 P_2 M_1 P_1 A$$

は右上三角行列である．$P = P_{n-1} \cdots P_1$ とおいて

$$\begin{aligned} L &= P(M_{n-1} P_{n-1} \cdots M_1 P_1)^{-1} \\ &= P_{n-1} \cdots (P_3 (P_2 (P_1 P_1^{-1}) M_1^{-1} P_2^{-1}) M_2^{-1} P_3^{-1}) \cdots P_{n-1}^{-1} M_{n-1}^{-1} \end{aligned}$$

と定義すると，次ページの計算より、L は単位左下三角行列である。よって、

$$LU = PA$$

LU decomp. with partial pivoting

$$\begin{pmatrix} 1 & & & \\ m_{21}^{(k-1)} & \ddots & & \\ \vdots & & 1 & \\ m_{n1}^{(k-1)} & \ldots & m_{n,k-1}^{(k-1)} & 1 \end{pmatrix} \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & m_{k+1,k} & \ddots \\ & & m_{nk} & & 1 \end{pmatrix} = \begin{pmatrix} 1 & & & \\ m_{21}^{(k-1)} & \ddots & & \\ \vdots & & 1 & \\ \vdots & & m_{k+1,k} & \ddots \\ m_{n1}^{(k-1)} & \ldots & m_{nk} & & 1 \end{pmatrix}$$

$$P_{k+1} \begin{pmatrix} 1 & & & \\ m_{21}^{(k-1)} & \ddots & & \\ \vdots & & 1 & \\ \vdots & & m_{k+1,k} & 1 \\ m_{n1}^{(k-1)} & \ldots & m_{nk} & & 1 \end{pmatrix} P_{k+1}^{-1} = \begin{pmatrix} 1 & & & \\ m_{21}^{(k)} & \ddots & & \\ \vdots & & 1 & \\ \vdots & & m_{k+1,k}^{(k)} & 1 \\ m_{n1}^{(k)} & \ldots & m_{nk}^{(k)} & & 1 \end{pmatrix}$$

部分ピボット選択法の実際

連立1次方程式は $PAx = Pb$ を解く．プログラミングをするときは、実際に行の入れ換えをするのではなく、何行目と何行目を入れ換えたかを覚えておけば十分である。

実際のプログラムは web page を参考にせよ。

`http://www-b2.is.tokushima-u.ac.jp/~ikeda/num`

演習問題(レポート)

19 部分ピボット選択法を用いたLU分解

- 次の連立一次方程式をLU分解を用いて解け。

$$\begin{pmatrix} 0 & 12 & 12 & 6 & 18 \\ 8 & 12 & 22 & 22 & 20 \\ 10 & 23 & 37 & 28 & 33 \\ 4 & 10 & 30 & 30 & 20 \\ 12 & 18 & 24 & 12 & 6 \end{pmatrix} x = \begin{pmatrix} 24 \\ 12 \\ 6 \\ 12 \\ 30 \end{pmatrix}$$

  をLU分解せよ。
- 実際のプログラムの中では行の入れ換えはどのようにして実現されているか? 説明せよ。
- * 部分ピボット選択法は0割りを避けるだけでなく、数値誤差の拡大を防ぐ効果がある。その原理を説明せよ。

—11 QR分解 I—

- QR分解は、最小2乗法や固有値問題の数値解法などで利用される方法である。
- QR分解とは、行列 A を直交行列 Q と右上三角行列 R の積に分解することである。
- 直交行列は誤差を拡大しないという“数値的によい”性質を持っている。
- QR分解はGram-Schmidtの正規直交化法のそものである。

–準備– ユニタリ行列、直交行列

$AA^* = A^*A = I$ となる行列をユニタリ行列という。(* は共役転置) また、$AA^{\mathrm{T}} = A^{\mathrm{T}}A = I$ となる行列を直交行列という。

ユニタリ変換 $y = Ax$ に対して, ベクトルのなす角, ベクトルの長さは不変である. (合同変換)

A をユニタリ行列, $\langle \cdot, \cdot \rangle$ を内積とすると、

$\langle Ax_1, Ax_2 \rangle = x_1^* A^* A x_2 = x_1^* x_2 = \langle x_1, x_2 \rangle$

ユニタリ行列の固有値の絶対値は 1

$Ax = \lambda x$ となるとき, λ: 固有値, x: 固有ベクトル

$x^*x = x^*A^*Ax = \bar{\lambda}\lambda x^*x$ より $|\lambda| = 1$

–準備– ユニタリ行列、直交行列

A : ユニタリ行列 $\Leftrightarrow AA^* = A^*A$ かつ

$$|\lambda_i(A)| = 1 \text{ for } i = 1, \ldots, n$$

U, V はユニタリ行列とする. ベクトル 2-norm $|\cdot|_2$ はユニタリ変換 $y = Vx$ に対して不変. $|x|_2^2 = \langle x, x \rangle$ よりあきらか.

行列 2-norm $\|\cdot\|_2$ はユニタリ変換 $B = U^*AV$ に対して不変.

$$\begin{aligned}\|U^*AV\|_2^2 &= \sup_{|z|_2=1} |U^*AVz|_2^2 \\ &= \sup_{|z|_2=1} z^*V^*A^*UU^*AVz = \sup_{|x|_2=1} x^*A^*Ax = \|A\|_2^2\end{aligned}$$

–準備– 特異値分解 (SVD)

任意の $\forall A \in C^{m\times n}$ に対して, ユニタリ行列 $\exists U, \exists V$ が存在して, 次式が成り立つ。(SVD:Singular Value Decomposition)

$$\begin{aligned}U^*AV &= \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_p), \\ &\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0, \quad p = \min\{m, n\}\end{aligned}$$

σ_i を A の特異値という. (singular value of A)

$\sigma_1 = \sigma_{\max}$ 最大特異値

$\sigma_p = \sigma_{\min}$ 最小特異値, A:正則 $\Rightarrow \sigma_{\min} > 0$

SVD は一意ではない。もし、$A = U^*DV$ なら $V' = Ve^{j\theta}$, $U' = Ue^{j\theta}$ とおくと、$U'^*DV' = U^*DV = A$ も SVD。ただし、D は一意に決まる。

–準備– 特異値分解の証明

[証明] $\sigma_1 = \|A\|_2 = \sup_{|x|_2=1} \sqrt{x^*A^*Ax}$ とする.

$\sigma_1 y = Ax$ となる x, y をとってくる. ($\|x\|_2 = \|y\|_2 = 1$)

$V = (\,x \quad V_1\,), U = (\,y \quad U_1\,)$ がそれぞれユニタリ行列になるように V_1, U_1 を選ぶ.

$$\begin{aligned}U^*AV &= \begin{pmatrix} y^* \\ U_1^* \end{pmatrix} A \,(\,x \quad V_1\,) = \begin{pmatrix} y^* \\ U_1^* \end{pmatrix} (\,\sigma_1 y \quad AV_1\,) \\ &= \begin{pmatrix} \sigma_1 & w^* \\ 0 & U_1^*AV_1 \end{pmatrix}\end{aligned}$$

–準備– 特異値分解の証明

ところで、$(\sigma_1^2 + w^*w)^2$ を計算すると、

$$\begin{aligned}(\sigma_1^2 + w^*w)^2 &\leq \left| \begin{pmatrix} \sigma_1 & w^* \\ 0 & U_1^*AV_1 \end{pmatrix} \begin{pmatrix} \sigma_1 \\ w \end{pmatrix} \right|_2^2 \\ &\leq \|A\|_2^2 \left| \begin{pmatrix} \sigma_1 \\ w \end{pmatrix} \right|_2^2 = \sigma_1^2(\sigma_1^2 + w^*w)\end{aligned}$$

なので、$w = 0$ でなければならない。故に,

$U^*AV = \begin{pmatrix} \sigma_1 & 0 \\ 0 & A_2 \end{pmatrix}$.

以下同様に

$$U^*AV = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_p) = D, \qquad A = UDV^*.$$

–準備– 特異値分解の続き

A を正則行列とする.

$$A = U^*DV = U^* \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{pmatrix} V, \ \sigma_1 \geq \cdots \geq \sigma_n > 0$$

$$A^{-1} = V^*D^{-1}U = V^* \begin{pmatrix} 1/\sigma_1 & & \\ & \ddots & \\ & & 1/\sigma_n \end{pmatrix} U$$

よって,

$\|A\|_2 = \sigma_{\max}$ A の最大特異値

$\|A^{-1}\|_2 = 1/\sigma_{\min}$ A の最小特異値の逆数

–準備– 条件数

行列 A の条件数 (condition number) は次式によって定義される。

$$\mathrm{cond}(A) = \|A\| \cdot \|A^{-1}\|$$

一般に $\mathrm{cond}(A) \geq 1$. 1 より大きくなるにしたがって行列の性質が悪くなり、連立一次方程式を解くときに、A や b の誤差の影響を受けやすくなる。

2-norm を用いた場合の条件数は以下の通りとなる.

$$\mathrm{cond}_2(A) = \|A\|_2 \cdot \|A^{-1}\|_2 = \frac{\sigma_{\max}}{\sigma_{\min}} \geq 1.$$

$Ax = b$ を $UAx = Ub$ としても, $\mathrm{cond}_2(UA) = \mathrm{cond}_2(A)$ なので, 問題の解き難さは変わらない.

–準備– Gram-Schmidt の正規直交化法

- 線形独立なベクトルの組み $\{a_1, a_2, \ldots, a_m\}$ が与えられたとき、互いに直交する長さ 1 のベクトル $\{q_1, q_2, \ldots, q_m\}$ を求める方法。
- Gram-Schmidt:

$$A = (\,a_1 \quad a_2 \quad \ldots \quad a_m\,) \mapsto Q = (\,q_1 \quad q_2 \quad \ldots \quad q_m\,)$$

ここで、$\langle q_i, q_j \rangle = \delta_{ij}$. すなわち、$Q$ は直交行列 (ユニタリ行列)

- QR 分解と本質的に同じである。

–準備– Gram-Schmidt の正規直交化法

a'_k, q_k の定義	a_k を $q_1, \ldots, q_k$ で表現
$q_1 = a_1/\|a_1\|_2$	$a_1 = \|a_1\|_2 q_1 \in \mathrm{span}\{q_1\}$
$a'_2 = a_2 - \langle q_1, a_2\rangle q_1$ $q_2 = a'_2/\|a'_2\|_2$	$a_2 = \langle q_1, a_2\rangle q_1 + \|a'_2\|_2 q_2$ $a_2 \in \mathrm{span}\{q_1, q_2\}$
$\vdots$	$\vdots$
$a'_k = a_k - \sum_{i=1}^{k-1}\langle q_i, a_k\rangle q_i$ $q_k = a'_k/\|a'_k\|_2$	$a_k = \sum_{i=1}^{k-1}\langle q_i, a_k\rangle q_i + \|a'_k\|_2 q_k$ $a_k \in \mathrm{span}\{q_1, \ldots, q_k\}$
$\vdots$	$\vdots$

–準備– Gram-Schmidt の正規直交化法

[証明] 帰納法で証明。$\{q_1, \ldots, q_{k-1}\}$ が直交しているとする．
$q_j (j = 1, \ldots, k-1)$ に対して $\langle q_j, q_k\rangle$ を計算する．

$$\begin{aligned}\|a'_k\|_2\langle q_j, q_k\rangle &= \langle q_j, a'_k\rangle = \langle q_j, a_k\rangle - \sum_{i=1}^{k-1}\langle q_i, a_k\rangle\langle q_j, q_i\rangle \\ &= \langle q_j, a_k\rangle - \langle q_j, a_k\rangle = 0\end{aligned}$$

よって，q_k は $\{q_1, \ldots, q_{k-1}\}$ と直交する．$\|q_k\|_2 = 1$ なので
$\langle q_i, q_j\rangle = \delta_{ij}$ すなわち $Q^*Q = I$.

最小2乗法

cf. データ $\{(x_i, y_i) : i = 1, \ldots, n\}$ を 1 次式 $y = \alpha x + \beta$ にあてはめたい．
$\sum_{i=1}^n (\alpha x_i + \beta - y_i)^2 \to \min$

$$A = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix}, \qquad x = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}, \qquad b = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

とおくと，$\|Ax - b\|_2^2 \to \min$.
連立一次方程式 $Ax = b$ ($A \in \mathrm{R}^{n\times m}, x \in \mathrm{R}^m, b \in \mathrm{R}^n$, A は列 full rank または 行 full rank をもつ.) は
$n = m \Rightarrow x$ は一意に定まる.
$n < m \Rightarrow$ 不定.
* $n > m \Rightarrow$ 不能. 残差を最小に

最小2乗法

$A = (a_1, \ldots, a_m)$ を $A = Q_1 R$ と分解する．
$q = (Q_1, \bar{Q}_1)$ がユニタリ行列となるように $\bar{Q}_1$ を選ぶ．

$$\begin{aligned}J &= \|Ax - b\|_2^2 = \|Q^*Ax - Q^*b\|_2^2 \\ &= \left\| \begin{pmatrix} Q_1^*Q_1Rx - Q_1^*b \\ \bar{Q}_1^*Q_1Rx - \bar{Q}_1^*b \end{pmatrix} \right\|_2^2 = \left\| \begin{pmatrix} Rx - Q_1^*b \\ -\bar{Q}_1 b \end{pmatrix} \right\|_2^2\end{aligned}$$

よって，$Rx = Q_1^*b$ を解けば，$J \to \min$ となる．
注 1. $\bar{Q}_1$ を計算する必要はない．
注 2. $\mathrm{cond}_2(Q) = 1, \mathrm{cond}_2(R) = \mathrm{cond}_2(A)$ である．
注 3. $A^{\mathrm{T}}Ax = A^{\mathrm{T}}b$ を解いても代数的には正しいが，
$\mathrm{cond}_2(A^{\mathrm{T}}A) = \mathrm{cond}_2(A)^2$ なので誤差が入りやすい。

Householder 変換 (鏡像変換)

$P = I - \dfrac{2vv^{\mathrm{T}}}{v^{\mathrm{T}}v}$ とおくと P は直交行列. なぜならば、

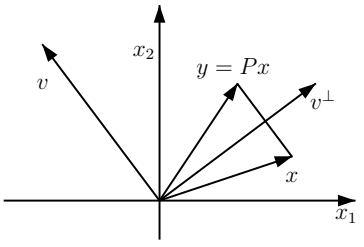
$$P^{\mathrm{T}}P = (I - \frac{2vv^{\mathrm{T}}}{v^{\mathrm{T}}v})(I - \frac{2vv^{\mathrm{T}}}{v^{\mathrm{T}}v}) = I - \frac{4vv^{\mathrm{T}}}{v^{\mathrm{T}}v} + \frac{4v(v^{\mathrm{T}}v)v^{\mathrm{T}}}{(v^{\mathrm{T}}v)^2} = I$$

変換 $Px = x - 2\dfrac{x^{\mathrm{T}}v}{v^{\mathrm{T}}v}v$ は x を軸 $v^\perp$ に対して対称な位置に変換する．(鏡像変換)

Householder 変換 (鏡像変換)

Householder 変換を用いて $x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ を

$e_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$ 軸に変換するには v をどのように選べばよいか?



Householder 変換 (鏡像変換)

$Px = x - 2\dfrac{x^{\mathrm{T}}v}{v^{\mathrm{T}}v}v \in \mathrm{span}\{e_1\}$ より，$v \in \mathrm{span}\{x, e_1\}$.
$v \neq 0$ の大きさは任意なので, $v = x + \alpha e_1$ とおく.

$$Px = \left(1 - 2\frac{x^{\mathrm{T}}x + \alpha x_1}{x^{\mathrm{T}}x + 2\alpha x_1 + \alpha^2}\right)x - 2\alpha\frac{v^{\mathrm{T}}x}{v^{\mathrm{T}}v}e_1 \text{ より}$$

$$\left(1 - 2\frac{x^{\mathrm{T}}x + \alpha x_1}{x^{\mathrm{T}}x + 2\alpha x_1 + \alpha^2}\right) = 0$$

でなければならない．よって，$\alpha = \pm\|x\|_2$.
$v = x \pm \|x\|_2 e_1 \Rightarrow Px = \mp\|x\|_2 e_1$
桁落ちを考慮すると, $v = x + \mathrm{sign}(x_1)\|x\|_2 e_1$

QR 分解のアルゴリズム

$$R = P_{m-1}P_{m-2}\cdots P_2P_1A = QA$$

ここで, P_k は, $A_1 = A$ とおいて、

$$A_k = P_{k-1}\cdots P_1A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1k} & \ldots & a_{1m} \\ & a_{22}^{(2)} & a_{23}^{(2)} & \ldots & a_{2k}^{(2)} & \ldots & a_{2m}^{(2)} \\ & & a_{33}^{(3)} & \ldots & a_{3k}^{(3)} & \ldots & a_{3m}^{(3)} \\ & & & \ddots & & & \vdots \\ & & & & a_{kk}^{(k)} & \ldots & a_{km}^{(k)} \\ & & & & \vdots & & \vdots \\ & & & & a_{nk}^{(k)} & \ldots & a_{nm}^{(k)} \end{pmatrix}$$

のとき，次ページを満たすユニタリ (直交) 行列である。

QR 分解のアルゴリズム

$$P_k \begin{pmatrix} a_{1k} \\ \vdots \\ a_{k-1,k}^{(k-1)} \\ a_{kk}^{(k)} \\ a_{k+1,k}^{(k)} \\ \vdots \\ a_{nk}^{(k)} \end{pmatrix} = \begin{pmatrix} a_{1k} \\ \vdots \\ a_{k-1,k}^{(k-1)} \\ -s_k \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

$$s_k = \mathrm{sign}(a_{kk}^{(k)}) \sqrt{{a_{kk}^{(k)}}^2 + \cdots + {a_{nk}^{(k)}}^2}$$

QR 分解のアルゴリズム

P_k を Householder 変換を用いて求めると，

$$P_k = I - \frac{2 v_k v_k^{\mathrm{T}}}{v_k^{\mathrm{T}} v_k}, \qquad v_k = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ a_{kk}^{(k)} + s_k \\ a_{k+1,k}^{(k)} \\ \vdots \\ a_{nk}^{(k)} \end{pmatrix}$$

となる．

演習問題 (レポート)

20 QR 分解

- 次の行列 A に Gram-Schmidt の正規直交化法を適用して、正規直交基底 $Q = (\, q_1 \quad q_2 \quad q_3 \,)$ を (手計算で) 求めよ。

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} = (\, a_1 \quad a_2 \quad a_3 \,)$$

- 上で求めた Q がユニタリ行列になっていることを確認せよ。

演習問題 (レポート)

- 次の式を最小にする x を求めよ。

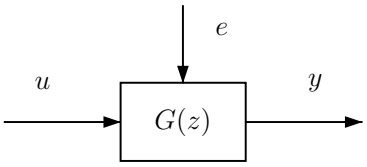
$$\left\| \begin{pmatrix} 2 & 1 \\ 1 & 1 \\ 0 & 1 \\ -1 & 1 \\ -2 & 1 \end{pmatrix} x - \begin{pmatrix} 2.6 \\ 2.4 \\ 1.1 \\ -0.1 \\ -1.0 \end{pmatrix} \right\|_2 \to \min$$

- s_k を求めるときに $\mathrm{sign}(a_{kk}^{(k)})$ を掛けるのはなぜか?
- プログラムで、計算量に関して工夫している点はどこか? 説明せよ。

—12 QR 分解 II 応用編—

システム同定

実システムの入出力データにもとづいて与えられたクラスの中から実システムに最も近いモデルを求める問題



u: 入力、y: 出力、e: 外乱 (測定できない入力)

T 時間単位のサンプリング周期の等間隔で信号を記録

システム同定

- 倒立振子の安定化補償器を設計するためには、振り子が立った状態での伝達関数モデルがあると便利である。
- 倒立振子は不安定系なので、最初はシステムの物理方程式に基づいて伝達関数を求める。
- 倒立振子も一旦安定化できれば、その入力と出力のペアから伝達関数を求めることができる。

線形差分方程式

$$\begin{aligned} &y(t) + a_1 y(t-1) + \cdots + a_{n_a} y(t-n_a) \\ &= b_1 u(t-n_k) + \cdots + b_{n_b} u(t-n_k-n_b+1) + e \end{aligned}$$

で表されるシステムを次の線形回帰モデル (linear regression model) で表す。

$$y = \Phi\theta + e$$

$\theta = (a_1, \ldots, a_{n_a}, b_1, \ldots, b_{n_b})^{\mathrm{T}}$ は求めるべきパラメータ

線形差分方程式

Φ, y, e は、$y_i = y(i), e_i = e(i)$ $(y_i = e_i = 0 \text{ if } i \le 0)$ とおいて、

$$\Phi = \begin{pmatrix} -y_0 & -y_{-1} & \cdots & -y_{1-n_a} & u_{-n_k} & \cdots & u_{1-n_k-n_b} \\ -y_1 & -y_0 & \cdots & -y_{2-n_a} & u_{1-n_k} & \cdots & u_{2-n_k-n_b} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ -y_{N-1} & -y_{N-2} & \cdots & -y_{N-n_a} & u_{N-1-n_k} & \cdots & u_{N-n_k-n_b} \end{pmatrix}$$

$$y = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}, \quad e = \begin{pmatrix} e_1 \\ \vdots \\ e_N \end{pmatrix}, \quad \left(\Phi = \begin{pmatrix} \phi_1^{\mathrm{T}} \\ \vdots \\ \phi_N^{\mathrm{T}} \end{pmatrix} \text{ とおいておく}\right)$$

線形差分方程式

θ を求めるためには、次の最小 2 乗問題を解けばよい。

$$J(\theta) = \sum_{i=1}^{N} (y_i - \phi_i^{\mathrm{T}}\theta)^2 = \|y - \Phi\theta\|^2 \to \min$$

この最小化問題は、QR 法を用いて解けばよい。

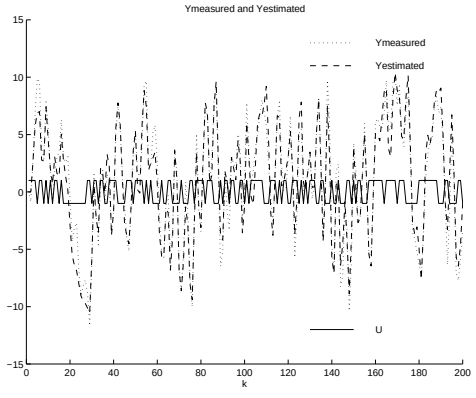
休憩 — MATLAB —

MATLAB は高性能な数値計算機能と多彩なグラフ機能を備えた技術計算ソフトである。会話型のインタプリタ言語であり、問題を数学で表現するように書けば実行してくれる。特定分野の解法を集めたツールボックスも多数用意されており、特に制御系設計や信号処理の分野では標準的に使われるシステムとなっている。離散時間システム (z^{-1} は遅れ作用素、すなわち、$z^{-i}y(t) = y(t-i)$)、

$$y(t) = \frac{2.5z^{-2} + 0.9z^{-3}}{1 - 0.9z^{-1} - +0.7z^{-2} - 0.5z^{-3}}u(t) + e(t) \quad (38)$$

に対して、適当な入力 u と外乱 e を作成し、その応答 y と u のペアから伝達関数を推定するなどは MATLAB を用いれば簡単にできる。結果は次ページ。

休憩 — MATLAB —



演習問題 (レポート)

21 QR 分解の応用 —システム同定—

- 適当な入力 u と外乱 e を作成し、(38) 式のシステムの出力 y を計算するプログラムを作り、入力 u と出力 y のペアを作れ。
- 上で作成した入力 u と出力 y のペアから伝達関数の係数を推定するプログラムを作り、伝達関数を推定せよ。

—12 行列の固有値問題—

- システムの安定性
- 最適化
- 物理システムの特性解析 (柔軟構造物の振動解析など)

など、工学の様々な問題は、行列の固有値問題に帰着される。

固有値問題

- $Av = \lambda v$ となる λ を行列 A の固有値、v を固有ベクトルという。
- 固有値は特性多項式 $\det(sI - A)$ の零点である。
- 5 次以上の代数方程式は、一般には有限回の四則演算と開平では解けないので、5×5 以上の大きさの行列の固有値は、繰り返し計算によって求めるしかない。
- 行列の特性方程式を求めるのは手間がかかる上に、数値的に不安定なので、数値解法では、特性方程式経由で固有値は求めない。
- 固有値の数値解法としては、べき乗法、QR 法、二分法などが有名である。

線形代数の復習 2

相似変換, ユニタリ相似変換

- $B = T^{-1}AT$ (T は正則) を相似変換といい、A と B は相似であるという。
- 固有値は相似変換に対して不変である。
- 任意の行列 $A \in C^{n\times n}$ は Jordan 標準形に相似である。
- $B = V^*AV$, (V はユニタリ ($V^*V = VV^* = I$)) をユニタリ相似変換といい、A と B はユニタリ相似であるという。
- $X^{-1}AX = \Lambda$ (Λ はジョルダン標準形) とし、$X = Q_X R_X$ と QR 分解すると、

$$A = X\Lambda X^{-1} = Q_x(R_X \Lambda R_X^{-1})Q_X^{-1}$$

より、任意の行列 $A \in C^{n\times n}$ は上三角行列にユニタリ相似である。

線形代数の復習 2 Jordan 標準形

- $\forall A \in C^{n\times n}, \lambda_1, \ldots, \lambda_s$ は A の相異なる固有値、$h_1, \ldots, h_s$ はその重複度とする。このとき、適当な複素行列 P を選ぶことによって,

$$P^{-1}AP = \begin{pmatrix} \Lambda_1 & & & \\ & \Lambda_2 & & \\ & & \ddots & \\ & & & \Lambda_s \end{pmatrix} = \Lambda \qquad (n \times n)$$

$$\Lambda_i = \begin{pmatrix} J_{i1} & & & \\ & J_{i2} & & \\ & & \ddots & \\ & & & J_{it_i} \end{pmatrix} \qquad (h_i \times h_i, \sum_{i=1}^{s} h_i = n)$$

$$J_{ij} = \begin{pmatrix} \lambda_i & 1 & & \\ & \lambda_i & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_i \end{pmatrix} \qquad (n_{ij} \times n_{ij}, \sum_{j=1}^{t_i} n_{ij} = h_i)$$

というブロック対角行列の形に表すことができる．さらに，λ_i, h_i, n_{ij} は A に対して一意的に定まる．

線形代数の復習 2

同値変換、ユニタリ変換、相似変換、ユニタリ相似変換のまとめ

	P, Q : 正則	U, V : ユニタリ
$A, B \in C^{n\times m}$	同値変換 $B = PAQ$ ランク不変 $\sim \begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix}$	ユニタリ変換 $B = U^*AV$ 特異値不変 $\sim \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{pmatrix}$
$A, B \in C^{n\times n}$	相似変換 $B = P^{-1}AP$ 固有値不変 → Jordan 標準形	ユニタリ相似変換 $B = V^*AV$ 固有値不変 → Schur Form (三角行列)

べき乗法 (Power Method)

- 絶対値最大の固有値 λ_1 とその固有ベクトル q_1 だけを求める方法
- $A^k q^{(0)} \to \lambda_1^k q_1 \quad \text{as } k \to \infty \text{ where } \quad q^{(0)} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$
- べき乗法のプログラム

  $q^{(0)} = [1, 0, \ldots, 0]^{\mathrm{T}}$

  for $k = 1, 2, 3, \ldots$

  $z^{(k)} = Aq^{(k-1)}$

  $q^{(k)} = z^{(k)}/\|z^{(k)}\|_2$ Normalization

  $\lambda^{(k)} = [q^{(k)}]^* A q^{(k)}$

  end

QR 法

- QR 法のプログラム

  $A_1 = A$

  for $k = 1, 2, 3, \ldots$

  $A_k = Q_k R_k$ と QR 分解

  $A_{k+1} = R_k Q_k$

  end
- $A_k \to \begin{pmatrix} \lambda_1 & & * \\ & \ddots & \\ & & \lambda_n \end{pmatrix}$ となる。すなわち、対角成分が固有値に収束する。

QR 法

- $R_k = Q_k^* A_k$ であるから、$A_{k+1} = Q_k^* A_k Q_k$、すなわちユニタリ相似変換になっている。
- $A_{k+1} = Q_k^* Q_{k-1}^* \cdots Q_1^* A Q_1 Q_2 \cdots Q_k = Q^{(k)*} A Q^{(k)}$

  ここで, $Q^{(k)} = Q_1 Q_2 \cdots Q_k$ とおいた。
- $R^{(k)} = R_k R_{k-1} \cdots R_1$ を仮想的に考えてみると、

  $Q^{(k)}$, $R^{(k)}$ は A^k の QR 分解になっている。
- Q_k は毎回 normalize されているので (ユニタリ行列なので)、誤差を拡大しない。
- A^k, $R^{(k)}$ は直接計算していない.

QR 法の実際

実用的な QR 法のプログラムは、前述の原理だけではなく、つぎに挙げるようないくつかの工夫を組み合わせた、大変大がかりなものになっている。

- Hessenberg form の利用
- 原点移動 (Origin shift)
- 減次 (deflation)
- 絶対値が等しい固有値をもつ場合への対応

  (Double Shifted QR など)

Hessenberg form の利用

- つぎのような形を Hessenberg form という。

$$\begin{pmatrix} * & \cdots & \cdots & * \\ * & \ddots & & \vdots \\ & \ddots & \ddots & \vdots \\ 0 & & * & * \end{pmatrix} \qquad a_{ij} = 0 \quad \text{if } i > j+1$$

- Hessenberg form の行列 A_1 を $A_1 = Q_1 R_1$ と QR 分解する手間は $\mathcal{O}(n^2)$。
- $A_2 = R_1 Q_1$ も Hessenberg で、その手間は $\mathcal{O}(n^2)$。
- 通常の行列の QR 分解/積の手間は $\mathcal{O}(n^3)$。
- 任意の正方行列はユニタリ相似変換 (Householder 変換など) によって Hessenberg form にできる。

原点移動 (Origin shift)

- $H = Q_H^{\mathrm{T}} A Q_H$ {Hessenberg Reduction};

  for $k = 1, 2, \ldots$

  determine a scalar $\mu (= h_{nn})$ {called shift};

  $H - \mu I = QR$ {QR factorizaion};

  $H = RQ + \mu I$

  end.
- $H_{new} = RQ + \mu I = Q^*(H_{old} - \mu I)Q + \mu I = Q^* H_{old} Q$
- H_{new} は H_{old} にユニタリ相似である。
- H の p 番目の subdiagonal 成分 $h_{p+1,p}$ は $\left(\frac{\lambda_{p+1}-\mu}{\lambda_p - \mu}\right)^k$ で収束する。$|\lambda_1 - \mu| > |\lambda_2 - \mu| > \cdots > |\lambda_n - \mu|$.
- μ を λ_{p+1} の近くに取れば、収束は加速される。

減次 (Deflation)

- $h_{n,n-1}$ が十分小さくなったら, $(n-1) \times (n-1)$ 行列の部分の固有値を求めればよい.
- 次数が小さい方が計算の手間が少なくて済む.

絶対値が等しい固有値をもつ場合

- $A \to \begin{pmatrix} A_1 & & * \\ & \ddots & \\ 0 & & A_s \end{pmatrix}$ A_i は絶対値が等しい固有値をもつブロック。(重複固有値, 共役複素固有値, etc)
- 共役複素固有値をもつ場合は、Double Shift QR を使うのが一般的である。
- 絶対値が等しい、あるいは、近い固有値をもつ場合は、数値計算的にみてより困難な問題となる。

共役複素固有値をもつ場合

- Double Shift QR a_1, a_2 は互いに共役とする。

$$\begin{aligned} H - a_1 I &= Q_1 R_1 \\ H_1 &= R_1 Q_1 + a_1 I \to H_1 = Q_1^* H Q_1 \\ H_1 - a_2 I &= Q_2 R_2 \\ H_2 &= R_2 Q_2 + a_2 I \to H_2 = Q_2^* H_1 Q_2 = Q_2^* Q_1^* H Q_1 Q_2 \end{aligned}$$

- 2 回 Shift QR を行ったと思えば，実数の範囲で計算できる。G は H の右下隅 2×2 行列とすると、

$$\begin{aligned} (Q_1 Q_2)(R_2 R_1) &= (H - a_1 I)(H - a_2 I) = M \\ &= H^2 - sH + tI \\ s &= a_1 + a_2 = h_{n-1,n-1} + h_{nn} = \operatorname{trace} G \in R \\ t &= a_1 a_2 = h_{n-1,n-1} h_{nn} - h_{n-1,n} h_{n,n-1} = \det G \in R \end{aligned}$$

共役複素固有値をもつ場合

- $M = H^2 - sH + tI$ を計算する.
  M を QR 分解する.
  $H_{new} = Q^* H_{old} Q$ とする.
- しかし，H^2 を計算するのに $\mathcal{O}(n^3)$ の手間が掛かる!!
- Implicit Q Theorem を使えば、$\mathcal{O}(n^2)$ の手間で済む。
- Implicit Q Theorem $H = Q^* A Q$: Hessenberg,
  $Q = (q_1, \ldots, q_n)$: unitary, q_1:givin, $h_{i+1,i} > 0$ ならば Q は一意に定まる．$q_2, \ldots, q_n$ がわからなくても H は完全に決まる.