物理学情報処理演習
10. 数値計算

2015年6月26日
ver20150626

本日の推奨作業directory

lesson10

10.1 方程式の解法
10.2 数値積分
10.3 数値微分
10.4 連立方程式の解法

参考文献

・やさしいC++ 第4版 高橋 麻奈 (著)
ソフトバンククリエイティブ
・プログラミング言語C++第4版
ビャーネ・ストラウストラップ, Bjarne Stroustrup, 柴田 望洋
・Numerical Recipes: The Art of Scientific Computing, Third Edition in C++

身内賢太朗
レポート提出:fsci-phys-jouhou@edu.kobe-u.ac.jp

数値計算（演習第二回の復習）

- 計算機が行える演算は、数値計算のみ。
- 具体的な解なしでは計算できない。
- 数値解は、解析的な解ではなく、数値を入れて解に最も近い値を探す。

10.1 方程式の解法

$$f(x)=0$$

の解は

$$y=f(x)$$

がx軸と交わる点のx座標である。数値計算での解放は非常に多くあるが、次の3つの方法を紹介する。(いずれも関数が連続ならば解の前後で関数の値の符号は異符号(+/-)であることを利用)

- a) 逐次法

  適当に決めた区間$[x_1,x_2]$を十分細かい精度で分割し、その分割点における関数 y の値を調べ、十分ゼロに近い値を解とする。ただし十分の判断が難しいので前後でyの値の符号が変化することでを判別する

- b) 二分法

  「ある区間$[x_1,x_2]$の2つ端点で関数 y が異符号である」ことが判っている場合、その区間の中点((x_1+x_2) / 2)での関数の値を調べ、同符号の端点と中点を入れ替える。入れ替えた新しい区間で同様の手続きを繰り返す。図では1-2区間、1-3区間と区間を縮める。

- c) 挟み撃ち法

  二分法の中点の代わりに、端点を結ぶ直線がx軸と交わる点を中点の代わりに採用する(関数を1次式に近似する方法)。

10.1a：逐次法

・逐次法

適当に決めた区間$[x_1, x_2]$を十分細かく分割し、その分割点における関数yの値を調べ、十分ゼロに近い値を解とする。

quadra_1は

$$ax^2 + bx + c = 0$$

の解を求めるプログラムである。

使用法は

>quadra_1 a b c x最小値　x最大値 xステップ

アルゴリズムは

・a*x*x + b*x + c をequ()として関数で定義
・係数、区間、その間の刻み幅を入力
・xが区間内の間loopを繰り返す
・y==0.0 あるいは　一つ前のyの値(y_pre) * y が負ならば解として解の配列 ans[] に代入
・yを一つ前のyの値として y_pre に代入
・loopが終わったら答えを出力

quadra_1.cxx

```
#include <string.h>
#include <math.h>
#define MIN_DEFAULT -100
#define MAX_DEFAULT 100
#define STEP_DEFAULT 0.1
using namespace std;
double equ(double x,double a, double b, double c){
  return a*x*x+b*x+c;
}
//quadratic equation 1
int main(int argc, char *argv[] ){
  double a,b,c;
  double min,max,step;
  double x,y,y_pre;
  double ans[16];
  int i;
  int ans_num=0;
  if(argc<4){
    cerr<< "quadra_1 a b c min max step"<< endl;
    return 1;
     }
  else{
    a=atof(argv[1]);
    b=atof(argv[2]);
    c=atof(argv[3]);
  }
  if(argc<6){
    min=MIN_DEFAULT;
    max=MAX_DEFAULT;
  }
  else{
    min=atof(argv[4]);
    max=atof(argv[5]);
  }
  if(argc<7){
    step=STEP_DEFAULT;
  }
  else{
    step=atof(argv[6]);
  }
```

関数の定義

パラメータセット

quadra_1.cxx

```
// output the parameters
  cerr << "quadra_1 for solving quadratic equation"<<endl;
  cerr <<"equation: "<<a<<"*x^2 + "<< b << "*x + " << c << " = 0"<<endl;
  cerr <<"ans search range: "<<min<<" : "<< max  << " (step= " << step << ")"<<endl<<endl;

  //initialization
  x=min;
  y_pre=equ(x,a,b,c);

  //answer search                                          loop
  while(x<max){
    y=equ(x,a,b,c);          比較・判定
    if(y==0){                （例外処理）
      cerr << "exact answer candidate found: "<<x<<endl;
      cerr << "¥t¥t y="<<y <<endl;
      ans[ans_num]=x;
      ans_num++;
    }
    else if(y*y_pre<0){      比較・判定
      cerr << "answer candidate found in "<<x-step<<" " <<x<<endl;
      cerr << "¥t¥t y="<<y_pre<<" : " <<y <<endl;
      ans[ans_num]=x-step/2;
      ans_num++;
    }
    y_pre=y;
    x+=step;
  }
  for(i=0;i<ans_num;i++){
    cerr<<"y("<<ans[i]<<")="<<equ(ans[i],a,b,c)<<endl;
  }                          解出力
}
```

演習10.1a（提出不要）quadra_1を実行、因数分解可能な係数を代入し、解析解と結果を比較せよ。

quadra_1の引数の扱いを確認せよ。

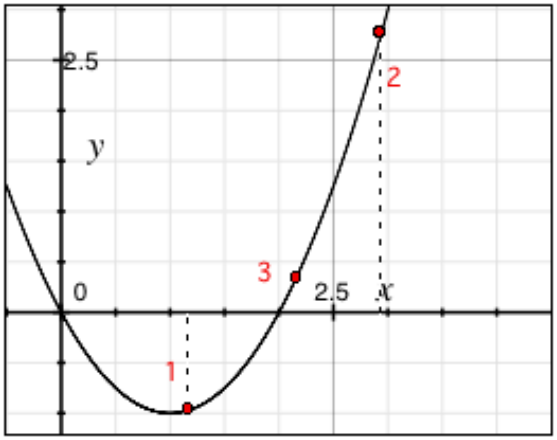
このloopを最小→最大　から　最大→最小　になるように変更してみよ。

10.1b:2分法
(bisection method)

・2分法
適当に決めた2点x_1,x_2に対する関数の値f(x_1), f(x_2)の符号が異符号なら解が必ず存在する。x_1, x_2の中点を新しい点として解に近づけて行く。
quadra_2は

$$ax^2 + bx + c = 0$$

を2分法で解くプログラムである。
使用法:
>quadra_2 a b c x_1 x_2 解の精度



quadra_2.cxx

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define MIN_DEFAULT -100
#define MAX_DEFAULT 100
#define ACCURACY_DEFAULT 0.1
#define ITERATION_MAX 100

using namespace std;
double equ(double x,double a, double b, double c){
  return a*x*x+b*x+c;
}

int output_ans(double ans,double a, double b, double c,int
itt){
  cout << "answer: x="<<ans<<" y="<<equ(ans,a,b,c)<<"
(after "<<itt<<" iterations)"<<endl;
  return 0;
}
//quadratic equation 1
int main(int argc, char *argv[] ){
  double a,b,c;
  double min,max,acc;
  double x_min,x_max,y_min,y_max;
  double x_mean,y_mean;
  double ans;
  int itt, itt_max; if(argc<4){
    cerr<< "quadra_1 a b c min max step"<< endl;
    return 1;
      }
  else{
    a=atof(argv[1]);
    b=atof(argv[2]);
    c=atof(argv[3]);
  }
```

繰り返し回数の上限

パラメータセット
(中略)

quadra_2.cxx

```
x_min=min;
 x_max=max;
 y_min=equ(x_min,a,b,c);
 y_max=equ(x_max,a,b,c);
 itt=0;
 if(fabs(y_min)<acc){ //fabs for absolute value
  ans=x_min;
  output_ans(ans,a,b,c,itt);
  return 0;
 }
 if(fabs(y_max)<acc){
  ans=x_max;
  output_ans(ans,a,b,c,itt);
  return 0;
 }
 if(y_min*y_max>0){
  cerr <<"Signs of y(x_min) and y(x_max) are same."<<endl;
  cerr <<"Bisection method cannot be started."<<endl;
  return 1;
 }
  x_mean=(x_min+x_max)/2.;
 y_mean=equ(x_mean,a,b,c);
 //answer search
 while(fabs(y_mean)>acc&&itt<itt_max){
  if(y_min*y_mean>0){
   x_min=x_mean;
  }
  else{
   x_max=x_mean;
  }
  y_max=equ(x_max,a,b,c);
  y_min=equ(x_min,a,b,c);
  x_mean=(x_min+x_max)/2.;
  y_mean=equ(x_mean,a,b,c);
  itt++;
 }
 ans=x_mean;
 output_ans(ans,a,b,c,itt);
 return 0;
}
```

初期化

例外処理①
いきなり解

例外処理②
挟めていない

loop

次の点を決定

二分法のアルゴリズム

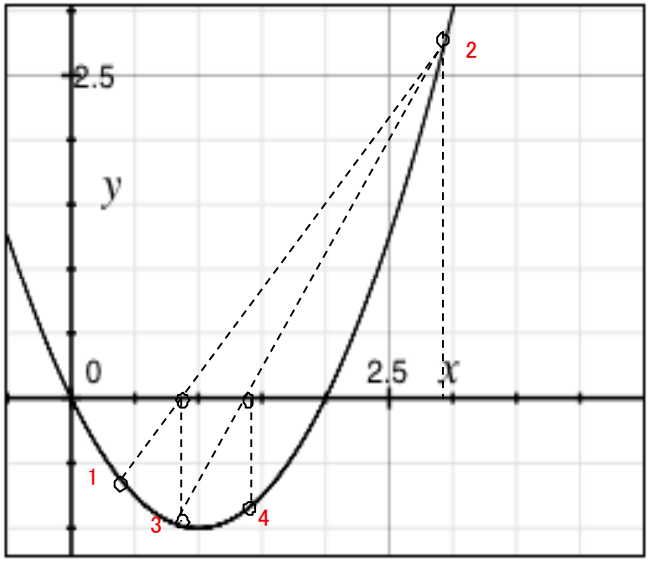
解が存在しないときのために繰り返し上限を定数で定めている。

- 例外処理
- 二分点とその関数値を求める(x_mean,ymean)
- y_meanの絶対値が解の精度(acc)より大きいかつ繰り返し回数が上限値以下の間、繰り返す。
- 二分点の関数の値y_meanと端点関数の値y_min, y_max の符号を調べ、同符号の端点を二分点に置き換える
- 解の出力

演習10.1b (提出不要) quadra_2を実行、quadra_1と比較してみよ。2分法では、関数の値の符号が違う2点を与える必要があることに注意。また、その間で解を一つのみ見つけることに特化していることにも注意する。

10.1c：挟み撃ち法

(false position method)



・挟み撃ち法

二分法では、区間の中点を考えたが、その区間の端点を結ぶ直線がx軸と交わる点

$$x = a - \frac{(b-a)f(a)}{f(b)-f(a)}$$

を用いる方法を挟み撃ち法という。

一般に挟み撃ち法の方が速く収束する。

演習10.1c（提出不要）quadra_2を変更して、quadra_3として挟み撃ち法で解を求めるプログラムとして、動作を確認せよ。いくつかの例で2分法と比較して収束が速い(iterationの回数が少ない)ことを確認せよ。

数値計算による方程式解法の注意

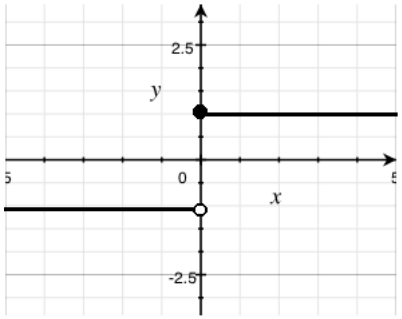
二分法、挟み撃ち法では次のような場合は解けない；

不連続関数

例）$f(x) = -1$ $\quad x < 0$

$f(x) = +1$ $\quad x \geq 0$

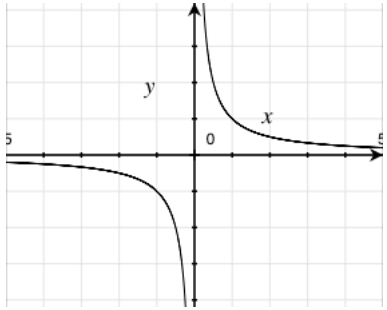
といった関数では、実際には解がないが、反復法（二分法、挟み撃ち法）では解は0に収束する。この場合では、0近傍で急激に-1から1に変化する連続関数と区別ができない。



特異点を持つ場合

例）$f(x) = x^{-1}$

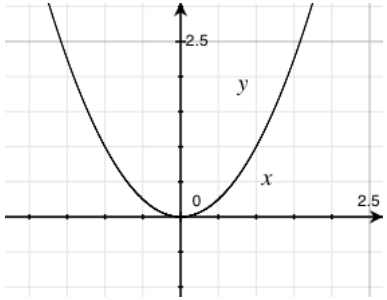
といった関数では、0の前後で、$f(x)$の符号は変化するが、解はない。このような場合は、x=0近傍でも$f(x)$の絶対値は小さくならないので、解がないことはわかる。



重解

例）$f(x) = x^2$ の解は2重解 x=0 である。

この場合、解の前後で$f(x)$の符号は正のままで変化しない。このような問題では二分法、挟み撃ち法などの囲い込み法では解けない。



数値計算による方程式解法:その他の方法

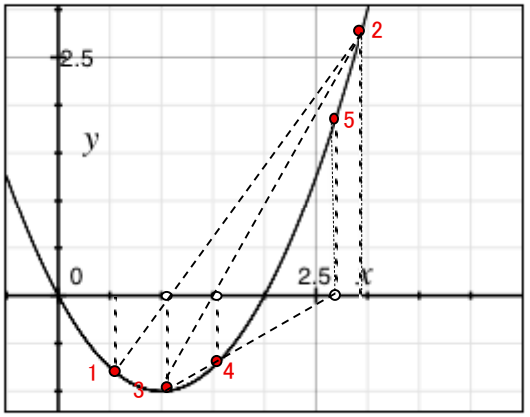
- 二分法、挟み撃ち法の他
  - 割線法

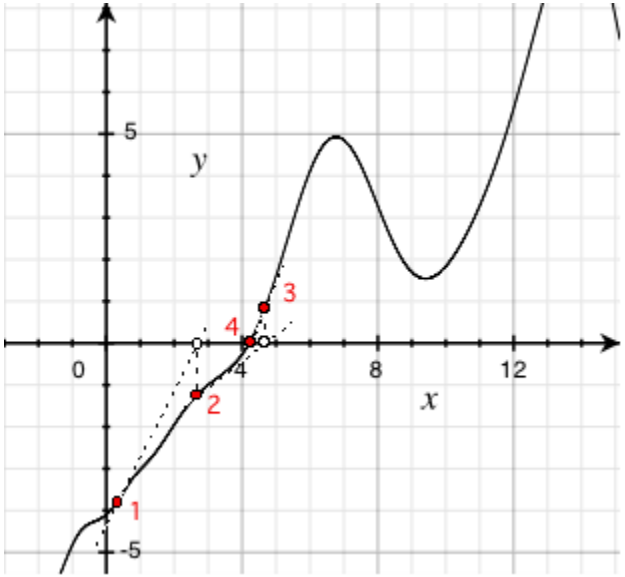
    挟み撃ち法では、中点と異符号の端点を残したが、割線法では必ず新しい端点を残す方法。

  - Newton法

    関数$f(x)$の導関数も判明している場合、Newton法が使用できる。
    割線法では、次の近似解を求めるのに、区間の端点を結んだ直線を使用したが、これを現在の解x_iでの接線を使用する方法。

- 以上述べてきた方法は、2次方程式のみならず、非線形方程式にも適用できる。



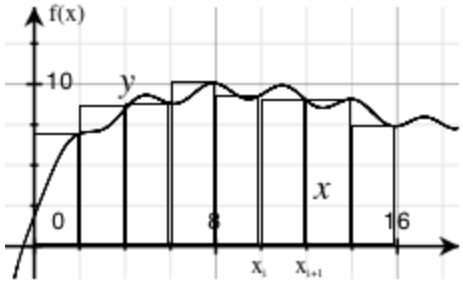


Newton法

10.2 数値積分

関数 $f(x)$ の区間 $[a, b]$ の積分をすることを考えよう。

計算機は一般には積分公式を知らないので、数学の定義に立ち返って数値的に積分を行う。



$$I = \int_a^b f(x)dx = \sum_{i=1}^{N} f(x_i)\cdot\Delta x = \Delta x\left[f(x_0) + f(x_1) + \cdots + f(x_N)\right]$$

この和を数値的に行えばよい。

便宜的に、積分区間$[a, b]$を等間隔でN等分し、その幅を$h=(b-a)/N$とすると、各点は

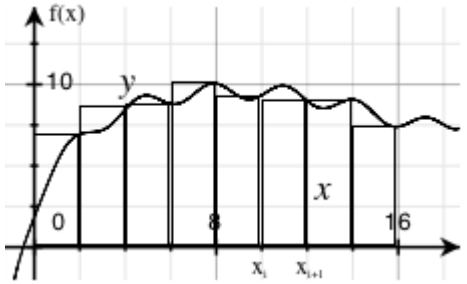
$$x_0=a,\ x_1=a+h,\ x_2=a+2h\ldots,\ x_N=b$$

と表される。

これらを関数$f(x)$ に代入し、足し合わせればよい。

10.2a 数値積分：矩形近似

数値積分を行う際、関数$f(x)$をN等分し、関数の積分を矩形で近似する方法。右図参照。
関数の変化に比べ、分割する数が多ければ近似として成り立つが、分割が少なければ悪い近似となる。



$$I = \int_a^b f(x)dx = \sum_{i=1}^{N} f(x_i)\cdot\Delta x = h\left[f(x_0) + f(x_0+\Delta x) + \cdots + f(x_N)\right]$$

この和を数値的に行えばよい。

10.2b 数値積分：台形近似

数値積分を行う際、関数$f(x)$をN等分し、関数をの積分を台形で近似する方法。

分割区間の両端で直線近似するので、矩形近似よりよい近似になっているが、関数の変化に比べ分割点が十分にあることが必要。

$$I=\int_a^b f(x)dx=\sum_{i=1}^{N}\left[\frac{1}{2}f(x_i)+\frac{1}{2}f(x_{i+1})\right]\cdot\Delta x=h\left[\frac{1}{2}f(x_0)+\sum_{i=1}^{N-1}f(x_i)+\frac{1}{2}f(x_N)\right]$$

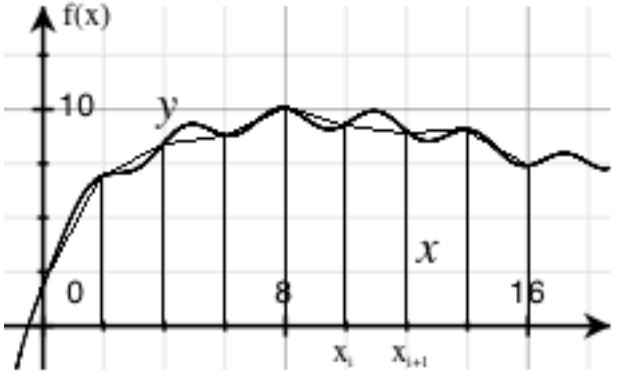
この和を数値的に行えばよい。この式を台形公式と呼ぶ。

演習10-2（提出不要） 演習第7回の提出コード2015_jouhou_07_学籍番号の下4桁.cxx の累積分布関数の計算を台形近似に書き換えて図で比較してみよう。

台形近似は、各区間で$f(x)$を直線近似したことになる。各区間で$f(x)$を2次曲線で近似すると積分は

$$I=\int_a^b f(x)dx=\sum_{i=1}^{N}\left[\frac{1}{3}f(x_i)+\frac{4}{3}f(x_{i+1})+\frac{1}{3}f(x_{i+2})\right]\cdot\Delta x$$

となる。これをSimpson則という。



演習10−3：数値微分

数値積分と同様、微分の概念に立ち返って、

$$\frac{df(x)}{dx} = \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

の定義より、数値的に微分を行うことが可能。

計算は、区間$[a, b]$をN等分に分け（$\Delta x=(b - a)/N$）を計算するだけである。

演習10−3（提出不要） 演習第7回の提出コード2015_jouhou_07_学籍番号の下4桁.cxx のガウシアンの微分を計算、図示してみよう。

演習10−3（提出不要） 解析的に微積分可能な関数（三角関数や指数関数）を数値積分・数微分して、正しく計算されることを確認しよう。

10.4 連立1次方程式の解法(参考)

連立方程式の解法として以下のものがある。

- Gauss-Jordan法（ガウス・ジョルダン法)
- Gaussの消去法（ガウスの消去法)
- Gauss-Seidel法（ガウス・ザイデル法)
- 共役傾斜法
- Cholesky法（コレスキー法)
- Crout法（クラウト法)

N 個の未知数x_j（j=1，2，…．，N）に対して、M 個の方程式

$$a_{11}\cdot x_1 + a_{12}\cdot x_2 + \cdots\cdots + a_{1N}\cdot x_N = b_1$$
$$a_{21}\cdot x_1 + a_{22}\cdot x_2 + \cdots\cdots + a_{2N}\cdot x_N = b_2$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$a_{M1}\cdot x_1 + a_{M2}\cdot x_2 + \cdots\cdots + a_{MN}\cdot x_N = b_M$$

があるとき、その解を求めることを考えよう。
ここで、a_{ij}，b_j は既知の数である。
　これらの方程式は、行列・ベクトルを用いると

$$\mathbf{A}\cdot\mathbf{x} = \mathbf{b}$$

と書き表すことができる。
これらM個の方程式の内、N個が線形独立なら（変数の数Nと同じか、それ以下ならば）、解が一意に定まる。

連立1次方程式の解法

行列$\mathbf{A}$の逆行列$\mathbf{A}^{-1}$は

$$\mathbf{A}\cdot\mathbf{A}^{-1}=\mathbf{A}^{-1}\cdot\mathbf{A}=\mathbf{I}$$

（$\mathbf{I}$は単位行列）となる。これを用いると解は

$$\mathbf{x}=\mathbf{A}^{-1}\cdot\mathbf{b}$$

と書き表される。

また、逆行列$\mathbf{A}^{-1}$のi列目のベクトルを$\boldsymbol{e}^{(i)}$とすると、

$\boldsymbol{e}^{(i)}$は方程式

$$\mathbf{A}\cdot\mathbf{x}=\mathbf{e}^{(\mathbf{i})}$$

$$\mathbf{e}_j^{(i)}=\delta_{ij}$$

の解となる。

以下では、簡単のためN=3として話を進めていく。

連立1次方程式の解法:Gauss-Jordan法

方程式を行列・ベクトルを用いてGauss-Jordan法で解く際に、行列式は以下の操作に普遍である性質を使う;

1. 任意の２行を入れ替える。
2. 任意の行をその行と別の行の線形接合で置き換える。

連立1次方程式の解法:Gauss-Jordan法

Gauss-Jordan法は、いわゆる消去法による連立１次方程式の解法。
次の連立一次方程式を考える;

$$a_{11}\cdot x_1+a_{12}\cdot x_2+a_{13}\cdot x_3=b_1\cdots\cdots\cdots\cdots\cdots\cdots(1)$$

$$a_{21}\cdot x_1+a_{22}\cdot x_2+a_{23}\cdot x_3=b_2\cdots\cdots\cdots\cdots\cdots\cdots(2)$$

$$a_{31}\cdot x_1+a_{32}\cdot x_2+a_{33}\cdot x_3=b_3\cdots\cdots\cdots\cdots\cdots\cdots(3)$$

(1)式をa_{11}で割り、x_1の係数を１にする((1)’)。
(2)式から(1)’式をa_{21}倍したものを引く。
(3)式から(1)’式をa_{31}倍したものを引く。
すると次のように(2)式、(3)式のx_1の係数は０になる。

$$x_1+a'_{12}\cdot x_2+a'_{13}\cdot x_3=b'_1\cdots\cdots\cdots\cdots\cdots\cdots(1)'$$

$$a'_{22}\cdot x_2+a'_{23}\cdot x_3=b'_2\cdots\cdots\cdots\cdots\cdots\cdots(2)'$$

$$a'_{32}\cdot x_2+a'_{33}\cdot x_3=b'_3\cdots\cdots\cdots\cdots\cdots\cdots(3)'$$

連立1次方程式の解法：Gauss−Jordan法

同様に

(2)’をa_{22}’で割る（(2)”）。

a_{12}’を(2)”にかけて(1)’から引く

a_{32}’を(2)”にかけて(3)’から引く

$$x_1 + a''_{13} \cdot x_3 = b''_1 \cdots\cdots\cdots\cdots\cdots\cdots (1)''$$

$$x_2 + a''_{23} \cdot x_3 = b''_2 \cdots\cdots\cdots\cdots\cdots\cdots (2)''$$

$$a''_{33} \cdot x_3 = b''_3 \cdots\cdots\cdots\cdots\cdots\cdots (3)''$$

さらに

(3)”をa_{33}”で割る（(3)’’’）。

a_{13}”を(3)’’’にかけて(1)”から引く

a_{23}”を(3)’’’にかけて(2)”から引く

$$x_1 = b'''_1 \cdots\cdots\cdots\cdots\cdots\cdots (1)'''$$

$$x_2 = b'''_2 \cdots\cdots\cdots\cdots\cdots\cdots (2)'''$$

$$x_3 = b'''_3 \cdots\cdots\cdots\cdots\cdots\cdots (3)'''$$

連立1次方程式の解法：Gauss−Jordan法

これで、

$$x_1 = b_1''' \qquad x_2 = b_2''' \qquad x_3 = b_3'''$$

と解が求められた。

上記の連立方程式は、以下のようにも表せる；

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \cdot \begin{bmatrix} y_{11} & y_{12} & y_{13} & x_1 \\ y_{21} & y_{22} & y_{23} & x_2 \\ y_{31} & y_{32} & y_{33} & x_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & b_1 \\ 0 & 1 & 0 & b_2 \\ 0 & 0 & 1 & b_3 \end{bmatrix}$$

ここで、

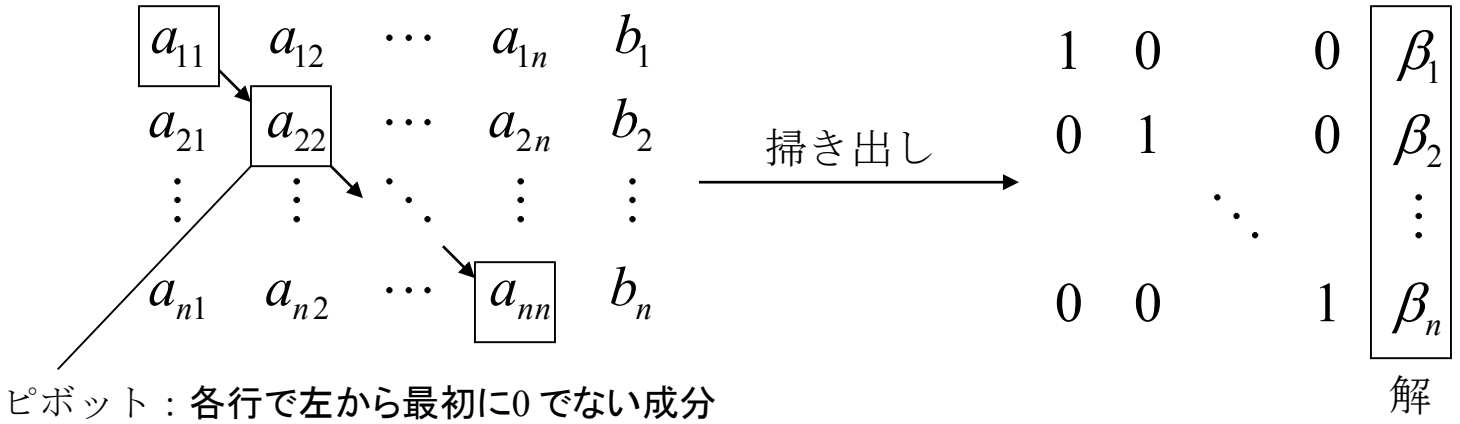
$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

x(=x_j)は、 **A・x=b** の解

Y(=y_{ij})は、**A**の逆行列　　である。

連立1次方程式の解法:Gauss-Jordan法

実際のプログラムは、次のような係数行列を作り、これが単位行列になるように掃き出し演算を行う。



アルゴリズムは以下の通り

ピボットを１行１列からn行n列に移しながら以下を繰り返す。

1. ピボットのある行の要素($a_{k\,k}, a_{k\,k+1}, \ldots, a_{k\,n}, b_k$)をピボット係数($a_{k\,k}$)で割る。結果としてピボットは１となる。なおピボット以前の要素($a_{k\,1}, a_{k\,2}, \ldots, a_{k\,k-1}$)はすでに０。
2. ピボット行以外の各行について以下を繰り返す。

（各行)ー(ピボット行）x（係数)。なお、この操作についてもピボット以前の列要素についてはすでに０になっている。

Gauss−Jordan法の実例

次の連立１次方程式を解く実例

$$2x_1 + 3x_2 + 1x_3 = 4$$
$$4x_1 + x_2 - 3x_3 = -2$$
$$-x_1 + 2x_2 + 2x_3 = 2$$

これを先の行列で表すと

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 4 \\ 4 & 1 & -3 & -2 \\ -1 & 2 & 2 & 2 \end{bmatrix}$$

4ren_1.cxx

```
#include <iostream>
#include <iomanip>
#include <stdlib.h>
#include <string.h>
#include <math.h>
using namespace std;
#define N 3    // 3x3 matrix
int main(int argc, char *argv[] ){
 double a[N][N+1] = {{2.0, 3.0, 1.0, 4.0}, {4.0,1.0, -3.0, -
2.0}, {-1.0, 2.0, 2.0, 2.0}};
 double p, d;
 int i, j, k;
 for (k=0; k<N; k++){
  p = a[k][k];    // pivot coefficient
  for (j=k; j<N+1; j++){
   a[k][j]=a[k][j]/p;      /* substract pivot gyou by p */
  }
  for (i=0; i<N; i++){    /* pivot sweep out */
   if (i!=k){
    d = a[i][k];
    for (j=k; j<N+1; j++){
                a[i][j]=a[i][j]-d*a[k][j];
    }
   }
  }
  for (k=0; k<N; k++){
   cout << "x" << k+1<<"="<<a[k][N]<<endl;
  }
 }
 return 0;
}
```

係数の定義

ピボットで割る

レポート提出：fsci-phys-jouhou@edu.kobe-u.ac.jp
提出期限　2015年7月3日13:00

ソースコードファイル名：2015_jouhou_10_学籍番号の下4桁.cxx
出力ファイル名：2015_jouhou_10_学籍番号の下4桁.dat
出力画像ファイル名：2015_jouhou_10_学籍番号の下4桁.pdf

課題10：　第二宇宙速度（地球から脱出するのに必要な速度）を数値積分を用いて以下の要領で求めよ。

① 考え方：重力による加速度は$1/R^2$で落ちる。重力による位置エネルギーは力を距離で積分したものに等しい。無限遠に脱出するためには、無限遠までに必要な位置エネルギーと等しい運動エネエルギーを初速度で持つ必要がある。

② 最低限必要なパラメータは$g=9.8\mathrm{m/s^2}$, 地球の半径$=6000\mathrm{km}$

③ 出力ファイルには　地球の半径で規格化した地球中心からの距離(無次元)　そこまで到達するのに必要な初速度　その場所での重力加速度　を出力する。

④ 横軸に地球の半径で規格化した地球中心からの距離(無次元)、縦軸にそこまで到達するのに必要な地上での速度[km/s]のグラフを描く。横軸の範囲は収束が確認される程度に各自調整すること。