CPPLapack Tutorial

東京工業大学 情報理工学研究科 情報環境学専攻
上島 正史
e-mail:ueshima@a.mei.titech.ac.jp

CPPLapack を用いたプログラミングをこのチュートリアルにて解説していきます．このチュートリアルで CPPLapack の全ての機能を網羅しているわけではありませんが，見てもらえれば CPPLapack の主たる部分は使えるようになると思います．

1 CPPLapackが使えるようになるまで

1.1 概要

CPPLapack のインストールから実行まで，linux 環境の場合を例に順を追って解説します．

1.2 インスト ール

最近の linux 環境であれば，

```
apt-get install blas
```

と入力して lapack のインストールを，

```
apt-get install lapack
```

と入力して BLAS のインストールができます．(yum 等他の方法でも構いません)

LAPACK と BLAS のインストールが終わったら CPPLapack のインストールを行います．インストールといってもファイルを展開するだけです．
手順は以下の通りです．

- `mkdir ~/local`，`mkdir ~local/lib` と入力し，ディレクトリ`~/local/lib` を作成する．
- `cd ~/local/lib` と入力する．
- `http://sourceforge.net/projects/cpplapack/`から CPPLapack の最新版 `cpplapack-XXXX_XX_XX.tar.gz` をダウンロードする．
- `tar xzf cpplapack-XXXX_XX_XX.tar.gz` と入力し，CPPLapack を展開する．
- `lu -s cpplapack-XXXX_XX_XX cpplapack` と入力し，シンボリックリンクを作成する．

以上でインストール完了です．

もし Cygwin や VC++環境で CPPLapack を利用したい場合は
を参考にしてください．

1.3 コードの生成

ユーザーは `cpplapack.h` をインクルードすれば CPPLapack の全ての機能を利用することができます．また，iostream と cmath など幾つかのヘッダファイルは内部で読み出し済みなので，これらを改めてインクルードする必要はありません．

例として，$A = \begin{bmatrix} 1 & 7 & 5 \\ 3 & 4 & 6 \end{bmatrix}$ という行列を表示するプログラムを作ります．コードは以下の通りです．

```
#include ‘‘cpplapack.h’’

void main(void)
{
  CPPL::dgematrix A(2,3);
  A(0,0)=1; A(0,1)=7; A(0,2)=5;
  A(1,0)=3; A(1,1)=4; A(1,2)=6;

  std::cout << A << std::endl;
}
```

これを main.cpp として保存することにします.

1.4 コンパイル

まず，~/local/lib/cpplapack/makefiles から Makefile.g++を main.cpp と同じディレクトリにコピーしてきて下さい．そして，

```
>mv Makefile.g++ Makefile
```

と入力してファイル名を変えて下さい．その次に Makefile の中身を開いて，

```
OBJECTS = main.o
```

となっていることを確認してください.

これらの確認作業が終わったら，コンパイルを行います．次の様に入力してください.

```
>make
```

これでコンパイルが行われ，実行ファイル A.OUT が生成されます．実際にコンパイルすると，次の様な感じになります.

```
>make
g++ -c main.cpp -O2 -Wall -Wno-unknown-pragmas -I/home/hoge/local/lib/cppla
pack/include
g++ main.o -O2 -Wall -Wno-unknown-pragmas -L/usr/local/lib/LAPACK -L/usr/local/
lib/BLAS -llapack -lblas -lg2c -lm  -o A.OUT
```

1.5 実行

コンパイルしてできあがった A.OUT を実行するには次の様に入力してください.

```
>./A.OUT
```

実行結果は次の様になります.

```
>./A.OUT
 1 7 5
 3 4 6
```

2 (基本演算) 連立一次方程式の解法

2.1 概要

CPPLapack では，連立一次方程式を解くための関数が各行列クラスのメンバ関数として用意されています.

```
dgematrix.dgesv
dgbmatrix.dgbsv
dsymatrix.dsysv
zgematrix.zgesv
zgbmatrix.zgbsv
zhematrix.zhesv
```

2.2 使い方

ここでは `double` 型の実一般行列を例に紹介し ます.
連立一次方程式 $Ax{=}y$ を解くには下記の様に記述し ます．解 x は y に上書きされます.

```
A.dgesv(y);
```

2.3 注意点

- これらの関数は LU 分解を用いて解いています．その為行列 A は正方行列でなくてはなりません.
- 計算後，行列 A は因子 L,U に (但し L の単位対角要素は格納し ない)，引数 y は解 x として 上書きされます．元の値を保持し たい場合は，関数を呼び出す前にコピーを作っておいて下さい.

2.4 例題

連立一次方程式

$$\begin{bmatrix} 1 & 1 & -2 \\ -3 & 2 & 1 \\ 3 & -1 & 2 \end{bmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

を解く.

コードを以下に示し ます.

```
//=========================================================================[include]
#include ‘‘cpplapack.h’’

//===========================================================================[main]
/*! main */
int main(int argc, char** argv)
{
  //// make dgematrix A  ////
  CPPL::dgematrix A(3,3);
  A(0,0)=1;  A(0,1)=1;  A(0,2)=-2;
  A(1,0)=-3; A(1,1)=2;  A(1,2)=1;
  A(2,0)=3;  A(2,1)=-1; A(2,2)=2;

  //// make dcovector y ////
  CPPL::dcovector y(3);
  y(0)=1;
  y(1)=2;
  y(2)=3;

  //// solve Ax=y ////
  A.dgesv(y);

  //// print ////
  std::cout << ‘‘x=\n’’ << y << std::endl;

  return 0;
}
```

実行結果は以下の様になります.

```
x=
 1
 2
 1
```

]

3 (基本演算) 線形最小二乗問題

3.1 概要

CPPLapack では，線形最小二乗問題を解くための関数が各行列クラスのメンバ関数として用意されています．

```
dgematrix.dgels
dgematrix.dgelss
zgematrix.zgels
zgematrix.zgelss
```

3.2 使い方

ここでは double 型の実一般行列を例に紹介します．
$Ax=y$ 形の線形最小二乗問題を解く時，以下の様に記述します．A は M 行 N 列行列であり，解 x は y に上書きされます．

```
A.dgels(y,r);
A.dgelss(y,S,RANK,RCOND);
```

dgels は $m \geq n$ の場合には最小二乗解を，$m \leq n$ の場合は最小ノルム解を求めます．r には計算残差が入ります．r は引数に取らなくてもでも構いません．
dgelss は最小ノルム解を求めます．S は特異値を出力する dcovector 型，RANK は A のランクを出力する double 型です．RCOND は A の実行精度を決める double 型で，デフォルト引数で *-1.0*(機種精度) に設定されているので，ここは空欄でもかまいません．
詳しくはマニュアルを御覧下さい．

3.3 注意点

- 計算後，行列 A は dgels では QR,LQ 分解の詳細に，dgelss では右特異ベクトルへと上書きされ，引数 y は解 x として上書きされます．元の値を保持したい場合は，関数を呼び出す前にコピーを作っておいて下さい．
- dgels ではフルランクを仮定しています．行列 A がランク落ちの可能性がある場合は dgelss を用いてください．

3.4 例題

連立一次方程式

$$\begin{bmatrix} 4 & 1 & 3 \\ 5 & 1 & -1 \\ 2 & 2 & 0 \\ -3 & 1 & 5 \end{bmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 8 \\ 2 \\ 0 \\ 5 \end{pmatrix}$$

を解く．

コードを以下に示します．

```
//=============================================================================[include]
#include ‘‘cpplapack.h’’

//=============================================================================[main]
/*! main */
int main(int argc, char** argv)
{
  //// make dgematrix A  ////
  CPPL::dgematrix A(4,3);
  A(0,0)=4;  A(0,1)=1;  A(0,2)=3;
  A(1,0)=5;  A(1,1)=1;  A(1,2)=-1;
  A(2,0)=2;  A(2,1)=2;  A(2,2)=0;
  A(3,0)=-3; A(3,1)=1;  A(3,2)=5;

  //// make dcovector y ////
  CPPL::dcovector y(4);
  y(0)=8;
  y(1)=2;
  y(2)=0;
  y(3)=5;

  //// solve Ax=y ////
  A.dgels(y);

  //// print ////
  std::cout << ‘‘x=\n’’ << y << std::endl;

  return 0;
}
```

実行結果は以下の様になります.

```
x=
 0.928934
 -0.92422
 1.74003
```

4 (基本演算) 固有値問題

4.1 概要

CPPLapack では，固有値問題を解くための関数が各行列クラスのメンバ関数として用意されています.

```
dgematrix.dgeev
dgematrix.dggev
dsymatrix.dsyev
zgematrix.zgeev
zgematrix.zggev
zsymatrix.zsyev
```

4.2 使い方

ここでは double 型の実一般行列を例に紹介します.
正方行列 A の固有値を求めるには，以下の様にします.

```
A.dgeev(wr,wi);
```

ここで，wr,wi は固有値の実部と虚部を出力する double 型ベクトルコンテナです.
同時に右固有値と右固有ベクトルを求めたい時は，dcovector 型ベクトルコンテナ，vrr,vri を用意して，

```
A.dgeev(wr,wi,vrr,vri);
```

と記述します.
左固有値と左固有ベクトルを求めたい時は，drovector 型ベクトルコンテナ，vlr,vli を用意して，

```
A.dgeev(wr,wi,vli,vri);
```

と記述します．また，一般化固有値，一般化固有ベクトルが欲しい場合は，dggev を利用してください.

4.3 注意点

計算後，行列 A は上書きされてしまいます．元の値を保持したい場合は，関数を呼び出す前にコピーを作っておいて下さい.

4.4 例題

行列 $A = \begin{bmatrix} 3 & 2 & 1 \\ 2 & 0 & 0 \\ 4 & 2 & 1 \end{bmatrix}$ の固有値，固有ベクトルを求める.

コードを以下に示します.

```
//=============================================================================[include]
#include ‘‘cpplapack.h’’

//=============================================================================[main]
/*! main */
int main(int argc, char** argv)
{
  //// make dgematrix A  ////
  CPPL::dgematrix A(3,3);
  A(0,0)=3; A(0,1)=2; A(0,2)=1;
  A(1,0)=2; A(1,1)=0; A(1,2)=0;
  A(2,0)=4; A(2,1)=2; A(2,2)=1;

  //// make wr wi vr ////
  std::vector<double> wr, wi;
  std::vector<CPPL::dcovector> vrr, vri;

  //// dgeev ////
  A.dgeev(wr, wi ,vrr, vri);

  //// print ////
  for(int i=0; i<3; i++){
    std::cout << ‘‘#### ‘‘ << i << ‘‘th eigen ####’’ << std::endl;
    std::cout << ‘‘wr=’’ << wr[i] << std::endl;
    std::cout << ‘‘wi=’’ << wi[i] << std::endl;
    std::cout << ‘‘vrr=\n’’ << vrr[i] << std::endl;
    std::cout << ‘‘vri=\n’’ << vri[i] << std::endl;
  }
}
```

実行結果は以下の様になります.

```
#### 0th eigen ####
wr=5
wi=0
vrr=
 0.620174
 0.248069
 0.744208

vri=
 0
 0
 0

#### 1th eigen ####
wr=-1
wi=0
vrr=
 0.447214
 -0.894427
 1.23361e-16

vri=
 0
 0
 0

#### 2th eigen ####
wr=1.36434e-16
wi=0
vrr=
 -5.22952e-17
 -0.447214
 0.894427

vri=
 0
 0
 0
```

5 (基本演算) 特異値分解

5.1 概要

CPPLapack では，特異値分解を行うための関数が各行列クラスのメンバ関数として用意されています．

```
dgematrix.dgesvd
zgematrix.zgesvd
```

5.2 使い方

ここでは double 型の実一般行列を例に紹介します．
行列 A の特異値分解を行うには下記の様に記述します．A は M 行 N 列行列です．

```
A.dgesvd(S,U,VT);
```

S は特異値を出力する dcovector 型，U は左特異ベクトル，VT は右特異ベクトルを出力する dgematrix 型です．

5.3 注意点

計算後，行列 A は上書きされてしまいます．元の値を保持したい場合は，関数を呼び出す前にコピーを作っておいて下さい．

5.4 例題

行列 $A = \begin{bmatrix} 4 & 1 & 3 \\ 5 & 1 & -1 \\ 2 & 2 & 0 \\ -3 & 1 & 5 \end{bmatrix}$ を特異値分解する．

コードを以下に示します．

```
//=====================================================================[include]
#include ‘‘cpplapack.h’’

//=====================================================================[main]
/*! main */
int main(int argc, char** argv)
{
  //// make dgematrix A  ////
  CPPL::dgematrix A(4,3);
  A(0,0)=4;  A(0,1)=1;  A(0,2)=3;
  A(1,0)=5;  A(1,1)=1;  A(1,2)=-1;
  A(2,0)=2;  A(2,1)=2;  A(2,2)=0;
  A(3,0)=-3; A(3,1)=1;  A(3,2)=5;

  //// make S, U and VT ////
  CPPL::dcovector S;
  CPPL::dgematrix U, VT;

  //// SVD A ////
  A.dgesvd(S,U,VT);

  //// print S, U, and VT ////
  std::cout << ‘‘S=\n’’ << S << std::endl;
  std::cout << ‘‘U=\n’’ << U << std::endl;
  std::cout << ‘‘VT=\n’’ << VT << std::endl;

  return 0;
}
```

実行結果は以下の様になります.

```
S=
 7.61305
 5.94552
 1.64083

U=
 -0.404476 -0.672423 0.445651 -0.430862
 -0.678796 -0.0902972 -0.0493761 0.727079
 -0.287353 -0.181365 -0.872916 -0.350075
 0.541353 -0.711899 -0.192289 0.403933

VT=
 -0.947144 -0.146673 0.285317
 -0.230125 -0.309031 -0.922791
 0.22352 -0.939674 0.258944
```

6　(応用例) 観測点の多項式近似

6.1　概要

ここからはCPPLapackの機能の応用例を示していきます．最初の例は観測点の多項式近似です．一般に最小二乗法の方が通りがよいかもしれません．

6.2　アルゴリズムの解説

m個の観測点(x_1, y_1)〜(x_n, y_n)をm次の多項式近似する場合，

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_n^n \end{bmatrix} \begin{pmatrix} k_0 \\ k_1 \\ \vdots \\ k_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

という$Ax{=}y$型の方程式を解けば，多項式の係数k_0〜k_nを求められます．より詳しい解説は他を御覧下さい．

6.3　例題

4つの観測点(0,-4.04), (1,-1.98), (2,2.02), (3,13.86)を3次の多項式で近似する．

コードを以下に示します．

```
//=============================================================================[include]
#include ‘‘cpplapack.h’’

//=============================================================================[solve]
void solve(int N,
           CPPL::dcovector& x, CPPL::dcovector& y)
{
  //// make dgematrix A ////
  CPPL::dgematrix A(x.l,N+1);
  for(int i=0; i<A.m; i++){for(int j=0; j<A.n; j++){
      A(i,j) = std::pow(x(i), (double)j);
  }}

  //// solve Ak=y ////
  A.dgels(y);
}

//=============================================================================[main]
/*! main */
int main(int argc, char** argv)
{
  //// make dcovector x,y ////
  CPPL::dcovector x(4), y(4);
  x(0)=0; y(0)=-4.04;
  x(1)=1; y(1)=-1.98;
  x(2)=2; y(2)=2.02;
  x(3)=3; y(3)=13.86;

  //// solve ////
  solve(3,x,y);

  //// print ////
  for(int i=0; i<3; i++){
    std::cout << ‘‘k’’ << i << ‘‘=’’ << y(i) << std::endl;
  }

  return 0;
}
```

実行結果は以下の様になります.

```
k0=-4.04
k1=3.05667
k2=-1.98
```

7　(応用例) 離散複素フーリエ変換

7.1　概要

二つめの例は離散複素フーリエ変換です.

7.2　アルゴリズムの解説

離散フーリエ変換の定義式は次のようになります.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi kn}{N}}$$

そこで,

$$W = e^{-j\frac{2\pi}{N}}$$

とし,

$$\begin{bmatrix} W^0 & W^0 & W^0 & \cdots & W^0 \\ W^0 & W^1 & W^2 & \cdots & W^n \\ W^0 & W^2 & W^4 & \cdots & W^{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ W^0 & W^{N-1} & W^{(N-1)2} & \cdots & W^{(N-1)n} \end{bmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ \vdots \\ X_{N-1} \end{pmatrix}$$

という $Ax{=}y$ 型の方程式を解けば，n 次の離散フーリエ変換を求められます．より詳しい解説は他を御覧下さい.

7.3　例題

data.txt から離散データを読み込み，10 次のフーリエ変換をして dft.txt に出力する.

コードを以下に示します.

```
//=============================================================================[include]
#include ‘‘cpplapack.h’’

//=============================================================================[DFT]
void DFT(CPPL::zcovector& X, int n)
{
  //// make zgematrix A ////
  CPPL::zgematrix A(X.l,n+1);
  for(int i=0; i<A.m; i++){ for(int j=0; j<A.n; j++){
    A(i,j) = std::complex<double>(std::cos((i*j)*2*MPI/(X.l))
                                 ,-std::sin((i*j)*2*MPI/(X.l)));
  }}

  //// solve Ax=X ////
  A.zgels(X);
}

//=============================================================================[main]
/*! main */
int main(int argc, char** argv)
{
  //// make zcovector X ////
  CPPL::zcovector X;
  X.read(‘‘data.txt’’);

  //// solve ////
  DFT(X,10);

  //// write ////
  X.write(‘‘dft.txt’’);

  return 0;
}
```

data.txt の中身です.

```
zcovector 100
(1,3.97901e-10)
(0.998027,-0.0627905)
(0.992115,-0.125333)
(0.982287,-0.187381)
(0.968583,-0.24869)
(0.951057,-0.309017)
(0.929776,-0.368125)
(0.904827,-0.425779)
(0.876307,-0.481754)
(0.844328,-0.535827)
    ⋮
```

dft.txt の中身です.

```
zcovector 11
(2.07977e-07,6.53992e-09)
(1,-6.4705e-07)
(-2.07977e-07,6.54019e-09)
(-1.03885e-07,6.54021e-09)
(-6.91412e-08,6.5401e-09)
(-2.80132e-08,6.53977e-09)
(-4.12676e-08,6.54009e-09)
(-3.42625e-08,6.54017e-09)
(-2.92385e-08,6.54009e-09)
(-3.86341e-08,6.53992e-09)
(-2.24983e-08,6.54008e-09)
```

]

8 (応用例) ポアソン方程式 直接法

8.1 概要

今回は，ポアソン方程式を直接法で解きます.

8.2 アルゴリズムの解説

1 次のポアソン方程式の一般系を

$$\frac{d^2\phi}{dx^2} = \rho$$

と書くとします．この時，2 次精度の差分式は，

$$\frac{\phi_{j-1} - 2\phi_j + \phi_{j+1}}{\Delta x^2} = \rho_j$$

となります.
そして，$\phi_1 = 0$ と $\phi_N = 0$ が境界条件として与えられた時，

$$\begin{bmatrix} -2 & 1 & & & & & \\ 1 & -2 & 1 & & & & \\ & \cdot & \cdot & \cdot & & & \\ & & 1 & -2 & 1 & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & 1 & -2 & 1 \\ & & & & & 1 & -2 \end{bmatrix} \begin{pmatrix} \phi_2 \\ \phi_3 \\ \vdots \\ \phi_j \\ \vdots \\ \phi_{N-2} \\ \phi_{N-1} \end{pmatrix} = \Delta x^2 \begin{pmatrix} \rho_2 \\ \rho_3 \\ \vdots \\ \rho_j \\ \vdots \\ \rho_{N-2} \\ \rho_{N-1} \end{pmatrix}$$

という $Ax{=}y$ 型の方程式を解けば，分布 $\boldsymbol{\phi}$ が求められます．より詳しい解説は他を御覧下さい.

8.3 例題

$$\frac{d^2\phi}{dx^2} = -k^2 \sin(kx) \qquad (k = 2\pi)$$

を解く．但し，境界条件を $\phi(0) = 0$,$\phi(1) = 0$ とし，$0 \leq x \leq 1$ の範囲で 201 個に離散化する.

コードを以下に示します.

```
//=====================================================================[include]
#include ‘‘cpplapack.h’’

//=====================================================================[main]
/*! main */
int main(int argc, char** argv)
{
  //// declare objects ////
  const int N(201);
  const double dx(1/(double)(N-1));
  const double k(2*MPI);
  CPPL::dgbmatrix A(N-2,N-2,1,1);
  CPPL::dcovector rho(N-2);

  //// make dgbmatrix A ////
  A.identity();
  A *= -2;
  for(int i=0; i<A.n-1; i++){
    A(i,i+1) = 1;
    A(i+1,i) = 1;
  }

  //// make dcovector rho ////
  for(int i=0; i<rho.l; i++){
    rho(i) = -k*k*sin((i+1)*k*dx);
  }
  rho *= dx*dx;

  //// solve A*phi=dx^2*rho ////
  A.dgbsv(rho);

  //// print ////
  std::cout << ‘‘x       phi(x)      Ans’’ << std::endl;
  std::cout << ‘‘0 0 0’’ << std::endl;
  for(int i=0; i<rho.l; i++){
    std::cout << (i+1)*dx << ‘‘ ‘‘ << rho(i) << ‘‘ ‘‘ << sin((i+1)*k*dx)
              << std::endl;
  }
  std::cout << ‘‘1 0 0’’ << std::endl;

}
```

実行結果は以下の様になります.

```
x       phi(x)      解析解
0 0 0
0.005 0.0314133 0.0314108
0.01 0.0627957 0.0627905
0.015 0.0941161 0.0941083
0.02 0.125344 0.125333
0.025 0.156447 0.156434
0.03 0.187397 0.187381
0.035 0.218161 0.218143
0.04 0.24871 0.24869
0.045 0.279014 0.278991
0.05 0.309042 0.309017
0.055 0.338766 0.338738
0.06 0.368155 0.368124
0.065 0.397181 0.397148
0.07 0.425814 0.425779
0.075 0.454028 0.45399
0.08 0.481793 0.481754
0.085 0.509083 0.509041
0.09 0.535871 0.535827
0.095 0.56213 0.562083
0.1 0.587834 0.587785
0.105 0.612957 0.612907
0.11 0.637476 0.637424
0.115 0.661366 0.661312
0.12 0.684603 0.684547
    ⋮
```

9 (応用例) ポアソン方程式 緩和法

9.1 概要

今度はポアソン方程式を緩和法 (PointJacobi 法) で解きます.

9.2 アルゴリズムの解説

計算の繰り返し数を n とし，n 回目の値と $n+1$ 回目の値が 1 つに収束して行く時,

$$\phi_j^{n+1} = \frac{1}{2}\left(\phi_{j+1}^n + \phi_{j-1}^n - \Delta x^2 \rho_j^n\right)$$

はポアソン方程式の差分式となります.
これを，境界条件 $\phi_1(0) = 0$，$\phi_N(1) = 0$ を考慮して行列表現すると，次の様になります.

$$\begin{pmatrix} \phi_2^{n+1} \\ \phi_3^{n+1} \\ \vdots \\ \phi_j^{n+1} \\ \vdots \\ \phi_{N-2}^{n+1} \\ \phi_{N-1}^{n+1} \end{pmatrix} = \frac{1}{2}\left(\begin{bmatrix} 0 & 1 & & & & & \\ 1 & 0 & 1 & & & & \\ & \cdot & \cdot & \cdot & & & \\ & & 1 & 0 & 1 & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & 1 & 0 & 1 \\ & & & & & 1 & 0 \end{bmatrix} \begin{pmatrix} \phi_2^n \\ \phi_3^n \\ \vdots \\ \phi_j^n \\ \vdots \\ \phi_{N-2}^n \\ \phi_{N-1}^n \end{pmatrix} - \Delta x^2 \begin{pmatrix} \rho_2 \\ \rho_3 \\ \vdots \\ \rho_j \\ \vdots \\ \rho_{N-2} \\ \rho_{N-1} \end{pmatrix} \right)$$

これを初期分布 $\boldsymbol{\phi^0}$ を適当に決め，$\| \boldsymbol{\phi^{n+1}} - \boldsymbol{\phi^n} \| \leq \epsilon$ となるまで繰り返し計算を行えば，分布 $\boldsymbol{\phi}$ が求められます.
より詳しい解説は他を御覧下さい.

9.3 例題

section8 と同じ問題を解く.

コードを以下に示します.

```
//=====================================================================[include]
#include ``cpplapack.h''

//=====================================================================[main]
/*! main */
int main(int argc, char** argv)
{
  //// declare objects ////
  const int N(201);
  const double dx(1/(double)(N-1));
  const double pai(3.141592);
  const double k(2*pai);
  const double eps(1.0e-6);
  CPPL::dgbmatrix A(N-2,N-2,1,1);
  CPPL::dcovector rho(N-2);
  CPPL::dcovector phi(N-2),phi_old(N-2);

  //// make dgbmatrix A ////
  A.zero();
  for(int i=0; i<A.n-1; i++){
    A(i,i+1) = 1;
    A(i+1,i) = 1;
  }

  //// make dcovector rho ////
  for(int i=0; i<rho.l; i++){
    rho(i) = -k*k*sin((i+1)*k*dx);
  }
  rho *= dx*dx;

  //// solve ////
  do{
    phi_old = phi;
    phi = A*phi_old-rho;
    phi *= 0.5;
  }while(nrm2(phi-phi_old)>eps);

  //// print ////
  std::cout << ``x       phi(x)      解析解'' << std::endl;
  std::cout << ``0 0 0'' << std::endl;
  for(int i=0; i<rho.l; i++){
    std::cout << (i+1)*dx << `` `` << phi(i) << `` `` << sin((i+1)*k*dx)
              << std::endl;
  }
  std::cout << ``1 0 0'' << std::endl;

}
```

実行結果は以下の様になります.

```
x       phi(x)      解析解
0 0 0
0.005 0.031407 0.0314108
0.01 0.062783 0.0627905
0.015 0.094097 0.0941083
0.02 0.125318 0.125333
0.025 0.156416 0.156434
0.03 0.187359 0.187381
0.035 0.218117 0.218143
0.04 0.24866 0.24869
0.045 0.278958 0.278991
0.05 0.30898 0.309017
0.055 0.338697 0.338738
0.06 0.36808 0.368124
0.065 0.3971 0.397148
0.07 0.425728 0.425779
0.075 0.453936 0.45399
0.08 0.481696 0.481754
0.085 0.50898 0.509041
0.09 0.535762 0.535827
0.095 0.562016 0.562083
0.1 0.587715 0.587785
0.105 0.612833 0.612907
0.11 0.637347 0.637424
0.115 0.661232 0.661312
0.12 0.684465 0.684547
    ⋮
```