Computer simulations create the future

数値計算ライブラリの使用方法

「実習編」

Kobe HPC Spring School 2019

今村俊幸　理化学研究所　計算科学研究センター

Toshiyuki Imamura, RIKEN Center for Computational Science

2019/3/13～ 15

本日の講義(2)

- 代表的な3ソフトウェアを使った演習
  - ScaLAPACK
  - EigenExa
  - PETSc
- ScaLAPACK
  - 密行列・ベクトルの基本計算
  - 連立一次方程式・固有値計算
- EigenExa
  - 密行列固有値計算
- PETSc/SLEPc
  - 疎行列、連立一次方程式、固有値計算
  - 他に開発環境としても

ScaLAPACKの利用方法

ScaLAPACKサイトの例題より

- **http://www.netlib.org/scalapack/examples/**

S ample_pdsyev_call.fをダウンロードする.

How to use ScaLAPACK

Computer simulations create the future

- **Link方法, Intel compiler+MPIの環境で**

```
% mpiifort –o exe  S ample_pdsyev_call.f -I${MKLROOT}/include -
L${MKLROOT}/lib/intel64 -lmkl_scalapack_lp64 -lmkl_intel_lp64 -lmkl_intel_thread -
lmkl_core -lmkl_blacs_intelmpi_lp64 -liomp5 -lpthread -lm -ldl
```

- **実行:**

```
#!/bin/bash
#PBS -q XXXX    (S, SMP1, SMP2などから選ぶ)
#PBS -l select=1:ncpus=4:mpiprocs=4
#PBS -N job_MPI
#PBS -o ex.out
#PBS -j oe

source /etc/profile.d/modules.sh
module load intel intelmpi

cd ${PBS_O_WORKDIR}
mpirun dplace ./a.out
```

Let’s learn the sample code

- 行列生成関数: PDLAMODHILB
- 固有値計算関数: PDSYEV
- 結果出力: PDLAPRNT

他に、初期化(BLACS_XXX)や終了(BLACS_EXIT)、行列データを扱うためのdescriptorの宣言(DESCINIT)などが必要。

BLACS: 通信回りの関数系、通常利用者からはプロセスの2次元配置の仕方などの管理系と思えばよい

PDSYEV: QR法による固有値計算ルーチン

他に、PDSYEVX, PDSYEVD, PDSYEVRなど存在する

行列データはプロセス間で「2次元ブロック分割」されており、行列要素ごとに格納されるプロセスが決まっている。

行列設定関数を読む

Computer simulations create the future

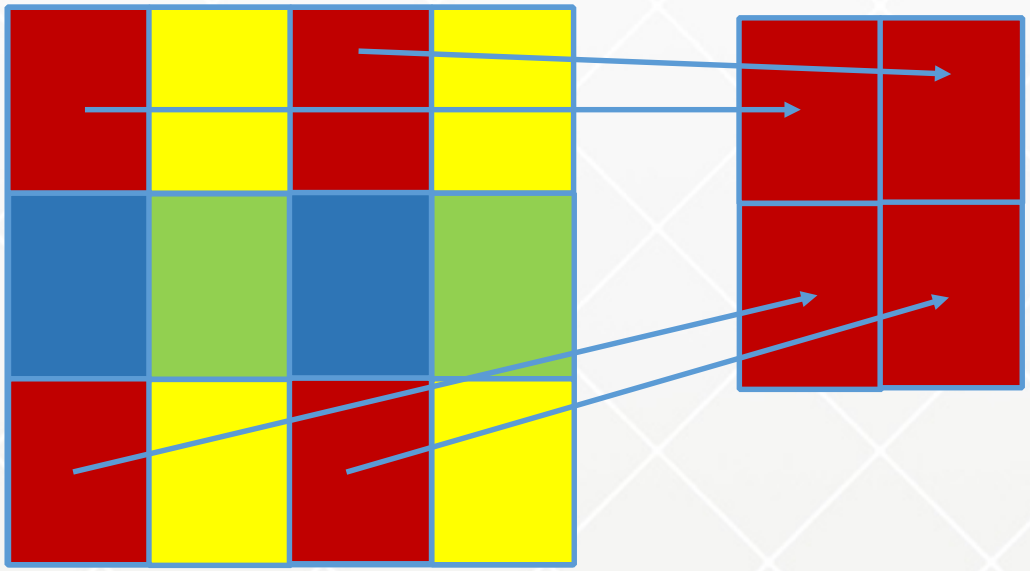
- **PDLAMODHILBを読んで分かるように**
  - 並列用の特別な変更は，代入操作を関数呼び出しPDELSETにしている点。PDELSETはもし、呼び出しプロセスがオーナーであれば指定された値をローカルメモリ上の配列データにストアする

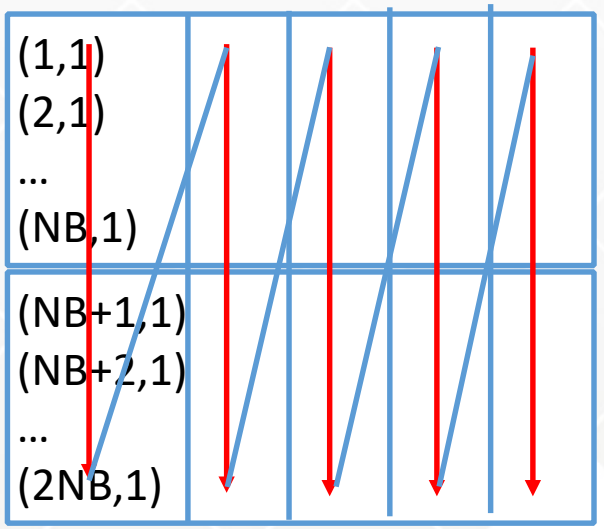
```
SUBROUTINE PDLAMODHILB( N, A, IA, JA, DESCA, INFO )
DO 20 J = 1, N
DO 10 I = 1, N
IF( I.EQ.J ) THEN
        CALL PDELSET( A, I, J, DESCA,  ¥
        ( DBLE( N-I+1 ) ) / DBLE( N )+ONE /  ( DBLE( I+J )-ONE ) )
ELSE
        CALL PDELSET( A, I, J, DESCA, ONE / ( DBLE( I+J )-ONE ) )
ENDIF

END
```

さらなる活用法

- プロセスグリッドとデータ分割
  - プロセスは2次元に配置（MPIのrankと関連付けられる）
    - プロセスグリッド
  - 基本データは2次元配列として、プロセスグリッド上に分散配置
    - ブロックサイズ(NB)を基本単位として、大きな行列の対応箇所のみ保有
    - FORTRANのC-Major方式に則って、同一列のブロックは1次元目が連続メモリアクセスとなるようにメモリ上に格納される





簡単な行列積

Computer simulations create the future

- **これまで行列データは形状と開始インデックスで指定(A(1,1) or A(1,2))**
  - **配列Aと分散情報を保持するデスクリプタの組で**

A(1,1) ➔ A, 1, 1, descA

注意: プログラム上の配列Aはローカルに確保された配列を代表しているので　A(1,1)は現在のプロセス上に分散格納されている部分成分の第1,1成分を示している。配列のA(1,1)をアクセスするにはownerを確認し, MPIに相当する通信でデータを前もって移動する必要がある。

Context: **プロセスグリッドを管理するハンドラ**

```
integer :: ICTXT
call  BLACS_GET( -1, 0, ICTXT )
 call BLACS_GRIDINIT( ICTXT, 'Row-major', NPROW, NPCOL )
 call BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
```

Descriptor: **行列情報を管理するハンドラ**

行列のサイズ（行、列）、ブロックサイズなどの情報を格納する。

特に分散方法を固定するなら以下のものを１つ用意して利用する

```
integer, dimension(9) :: DESCA
call DESCINIT( DESCA, n, n, NB, NB, 0, 0, ICTXT, lda, INFO )
```

簡単な行列積

- 行列Aの設定を先のPDLAMODHILBを使わずに
  - A(I,J)=(N+1)-max(I,J)

```
! Setup A
    do jl=1,NQ
      do il=1,NP
        ib=(il-1)/NB
        jb=(jl-1)/NB
        i0=MOD(il-1,NB)
        j0=MOD(jl-1,NB)
        i=(ib*NPROW+MYROW)*NB+i0+1
        j=(jb*NPCOL+MYCOL)*NB+j0+1
        if(i<=n.AND.j<=n)then
          a(il,jl)=(n+1)-MAX(i,j)
        else
          a(il,jl)=0
        endif
      end do
    end do
```

簡単な行列積

Computer simulations create the future

- 実際の行列積部分 (PDGEMM)
  - C=A*B

```
call MPI_Barrier ( MPI_COMM_WORLD, ierr )
z1 = MPI_Wtime()

CALL PDGEMM ( "N", "N", n, n, n, alpha, a, 1, 1, DESCA, &
              b, 1, 1, DESCA, beta, c, 1, 1, DESCA )

call MPI_Barrier ( MPI_COMM_WORLD, ierr )
z2 = MPI_Wtime()
```

  - 並列版でないDGEMMを参考に
    - “N”,”N”は転置するかしないかを指定できる
    - 1,1を指定している部分は部分行列のとき開始インデックスを指定できる
    - 今回行列の形状は同じなのでDESCAをすべてで使用(それぞれ変更可能)

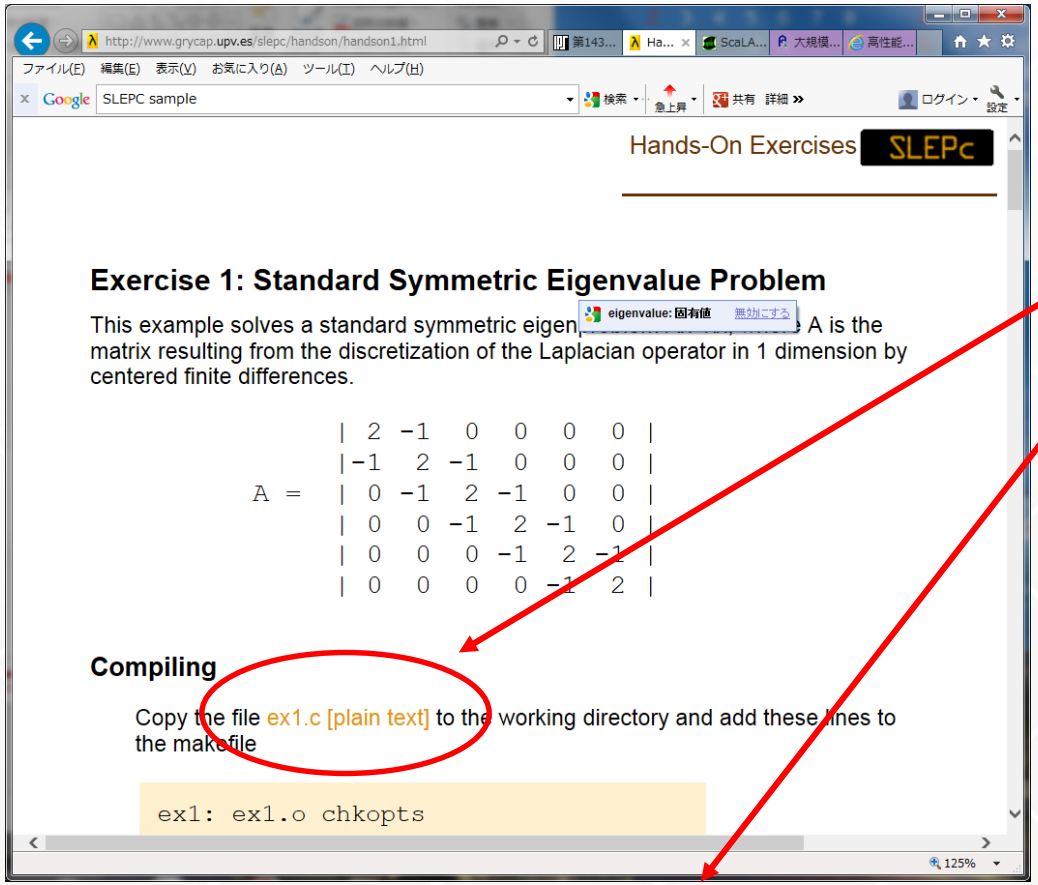
PETSc(SLEPC)の使い方

PETSc/SLEPC

公式Hands-on exercisesサイトを活用

＋サンプルの改良を目指す

http://slepc.upv.es/handson/handson1.html



ex1.cをダウンロードしてください

ex1ｆ.Ｆをダウンロードしてくだ

（スクロールした下の方にあります

コンパイル&リンク

Computer simulations create the future

Makefile

```
HOME=/home/guest39
ARCH = arch-linux2-c-debug
PETSC_DIR = $(HOME)/petsc
SLEPC_DIR = $(HOME)/slepc
INCPATH= -I$(PETSC_DIR)/include -I$(PETSC_DIR)/$(ARCH)/include ¥
      -I$(SLEPC_DIR)/include -I$(SLEPC_DIR)/$(ARCH)/include
LDFLAGS= -L$(SLEPC_DIR)/$(ARCH)/lib -lslepc ¥
      -L$(PETSC_DIR)/$(ARCH)/lib -lpetsc

all: ex1 ex1f
ex1: ex1.o
        mpiicc -o ex1 ex1.o $(LDFLAGS)
ex1f: ex1f.o
        mpiifort -o ex1f ex1f.o $(LDFLAGS)
ex1.o: ex1.c
        mpiicc -c ex1.c $(INCPATH)
ex1f.o: ex1f.F
        mpiifort -c ex1f.F $(INCPATH)
clean:
        ¥rm ex1 ex1.o ex1f ex1f.o
```

makeコマンドで
コンパイルする
→ ex1, ex1fが作成される

プログラムの実行


```
#!/bin/bash
#PBS –q XXXX (S, SMP1, SMP2などから選ぶ)
#PBS -l select=1:ncpus=4:mpiprocs=4
#PBS -N job_MPI
#PBS -o ex1.out
#PBS -j oe

source /etc/profile.d/modules.sh
module load intel intelmpi

export PETSC_DIR=/home/guest39/petsc
export SLEPC_DIR=/home/guest39/slepc
export PETSC_ARCH=arch-linux2-c-debug
#
export LD_LIBRARY_PATH=$PETSC_DIR/$PETSC_ARCH/lib:$SLEPC_DIR/$PETS
C_ARCH/lib:$HOME/lib64:$LD_LIBRARY_PATH

cd ${PBS_O_WORKDIR}
mpirun dplace ./ex1 -n 100

cd ${PBS_O_WORKDIR}
mpirun dplace ./ex1f -n 100
```

C version

1-D Laplacian Eigenproblem, n=100

Number of iterations of the method: 19
Solution method: krylovschur

Number of requested eigenvalues: 1
Stopping condition: tol=1e-08, maxit=100
Number of converged eigenpairs: 2

k	||Ax-kx||/||kx||
3.999033	4.02784e-09
3.996131	4.31174e-09

Cバージョンの結果

F90 version

1-D Laplacian Eigenproblem, n =100 (Fortran)

Number of iterations of the method: 19
Solution method: krylovschur
Number of requested eigenvalues: 1
Stopping condition: tol=1.0000E-08, maxit= 100
Number of converged eigenpairs: 2

k	||Ax-kx||/||kx||
3.9990E+00	4.0278E-09
3.9961E+00	4.3117E-09

F90バージョンの結果

Play with SLEPC

Computer simulations create the future

- チュートリアルページから

$ ./ex1 -n 400 -eps_nev 3 -eps_tol 1e-7

1-D Laplacian Eigenproblem, n=400

Number of iterations of the method: 100
Solution method: krylovschur

Number of requested eigenvalues: 3
Stopping condition: tol=1e-07, maxit=100
Number of converged eigenpairs: 1

k	||Ax-kx||/||kx||
3.999939	9.48781e-08

$ ./ex1 -n 400 -eps_nev 3 -eps_ncv 24

1-D Laplacian Eigenproblem, n=400

Number of iterations of the method: 60
Solution method: krylovschur

Number of requested eigenvalues: 3
Stopping condition: tol=1e-08, maxit=100
Number of converged eigenpairs: 5

k	||Ax-kx||/||kx||
3.999939	9.48494e-09
3.999754	7.19493e-09
3.999448	1.18552e-09
3.999018	6.43926e-10
3.998466	1.04213e-09

$ ./ex1 -n 100 -eps_nev 4 -eps_type lanczos

1-D Laplacian Eigenproblem, n=100

Number of iterations of the method: 62
Solution method: lanczos

Number of requested eigenvalues: 4
Stopping condition: tol=1e-08, maxit=100
Number of converged eigenpairs: 4

k	||Ax-kx||/||kx||
3.999033	9.95783e-09
3.996131	1.97435e-09
3.991299	9.15231e-09
3.984540	3.55339e-09

Learn the sample code


```
SlepcInitialize( PETSC_NULL_CHARACTER, ierr )

MatCreate( PETSC_COMM_WORLD, A, ierr )
MatSetSizes( A, …., n, n, ierr )
MatSetUp( A, ierr )


(行列やベクトルデータの宣言とデータ設定)


ESPCreate( PETSC_COMM_WORLD, eps, ierr )
ESPSetOperators( eps, A, PETSC_NULL_OBJECT, ierr )
EPSSetProblemType( eps, EPS_HEP, ierr )

EPSSolve( eps, ierr )

EPSGetEigenPair( eps, …… )

EPSDestroy( eps, ierr )
SlepcFinalize( ierr )
```

How to setup a matrix?

Computer simulations create the future

- **PETScは内部データを柔軟に制御している。PETScが管理するため、使用者からは実態は見えない。行列ハンドラ変数 A を使ってアクセスする。内部フォーマットはいくつか存在する。**

```
! Simple matrix format
Mat         A
EPS         eps
EPSType     tname
PetscReal   tol, error, values(:)

MatCreate( PETSC_COMM_WORLD, A, ierr )
MatSetSizes( A, PETSC_DECIDE, PETSC_DECIDE, M, N, ierr )

MatGetOwnershipRange(A, Istart, Iend, ierr )
MatSetValues(  A, m, idxm, n, idxn, values, INSERT_VALUES|ADD_VALUES, ierr)

MatAssemblyBegin( A, MAT_FINAL_ASSEMBLY, ierr)
MatAssemblyEnd( A, MAT_FINAL_ASSEMBLY, ierr)
```

How to setup a matrix?

Computer simulations create the future

- **PETScは内部データを柔軟に制御している。PETScが管理するため、使用者からは実態は見えない。行列ハンドラ変数 A を使ってアクセスする。内部フォーマットはいくつか存在する。**

```
! Simple matrix format
Mat         A
EPS         eps
EPSType     tname
PetscReal   tol, error, values(:)

MatCreate( PETSC_COMM_WORLD, A, ierr )
MatSetSizes( A, PETSC_DECIDE, PETSC_DECIDE, M, N, ierr )

MatGetOwnershipRange(A, Istart, Iend, ierr )
MatSetValues(  A, m, idxm, n, idxn, values, INSERT_VALUES|ADD_VALUES, ierr)

MatAssemblyBegin( A, MAT_FINAL_ASSEMBLY, ierr)
MatAssemblyEnd( A, MAT_FINAL_ASSEMBLY, ierr)
```

How to setup a matrix?

Computer simulations create the future

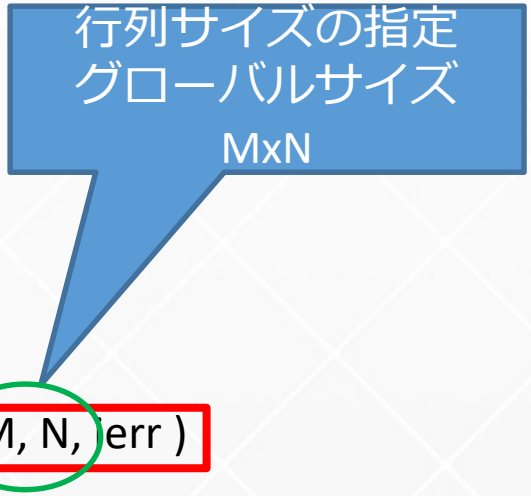
- **PETScは内部データを柔軟に制御している。PETScが管理するため、使用者からは実態は見えない。行列ハンドラ変数 A を使ってアクセスする。内部フォーマットはいくつか存在する。**

```
! Simple matrix format
Mat         A
EPS         eps
EPSType     tname
PetscReal   tol, error, values(:)

MatCreate( PETSC_COMM_WORLD, A, ierr )
MatSetSizes( A, PETSC_DECIDE, PETSC_DECIDE, M, N, ierr )

MatGetOwnershipRange(A, Istart, Iend, ierr )
MatSetValues(  A, m, idxm, n, idxn, values, INSERT_VALUES|ADD_VALUES, ierr)

MatAssemblyBegin( A, MAT_FINAL_ASSEMBLY, ierr)
MatAssemblyEnd( A, MAT_FINAL_ASSEMBLY, ierr)
```



How to setup a matrix?

Computer simulations create the future

- **PETScは内部データを柔軟に制御している。PETScが管理するため、使用者からは実態は見えない。行列ハンドラ変数 A を使ってアクセスする。内部フォーマットはいくつか存在する。**

```
! Simple matrix format
Mat          A
EPS          eps
EPSType      tname
PetscReal    tol, error, values(:)

MatCreate( PETSC_COMM_WORLD, A, ierr
MatSetSizes( A, PETSC_DECIDE, PETSC_DECIDE, M, N, ierr )

MatGetOwnershipRange(A, Istart, Iend, ierr )
MatSetValues(  A, m, idxm, n, idxn, values, INSERT_VALUES|ADD_VALUES, ierr)

MatAssemblyBegin( A, MAT_FINAL_ASSEMBLY, ierr)
MatAssemblyEnd( A, MAT_FINAL_ASSEMBLY, ierr)
```

mxnのブロック行列に対して配列valuesをセットする

How to setup a matrix?

Computer simulations create the future

- **PETScは内部データを柔軟に制御している。PETScが管理するため、使用者からは実態は見えない。行列ハンドラ変数 A を使ってアクセスする。内部フォーマットはいくつか存在する。**

```
! Simple matrix format
Mat         A
EPS         eps
EPSType     tname
PetscReal   tol, error, values(:)

MatCreate( PETSC_COMM_WORLD, A, ierr )
MatSetSizes( A, PETSC_DECIDE, PETSC_DECIDE, N, ierr )

MatGetOwnershipRange(A, Istart, Iend, ierr)
MatSetValues(  A, m, idxm, n, idxn, values, INSERT_VALUES|ADD_VALUES, ierr)

MatAssemblyBegin( A, MAT_FINAL_ASSEMBLY, ierr)
MatAssemblyEnd( A, MAT_FINAL_ASSEMBLY, ierr)
```

少し複雑な問題にチャレンジ

- **2次元問題の固有値分析を**

$$A = \begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & & & -1 & 2 \end{bmatrix}$$

：１次元の場合

$$A_0 = \begin{bmatrix} 4 & -1 & & & & \\ -1 & 4 & -1 & & & \\ & -1 & 4 & -1 & & \\ & & & & -1 & 4 \end{bmatrix}$$

とおくと、２次元の場合は次のように
ただしＩは対応する大きさの単位行列

$$A = \begin{bmatrix} A_0 & -I & & & & \\ -I & A_0 & -I & & & \\ & -I & A_0 & -I & & \\ & & & & -I & A_0 \end{bmatrix}$$

少し複雑な問題にチャレンジ

- 行列のsetup方法が変わる

```
MatGetOwnershipRange(A,&Istart,&Iend);
for (i=Istart;i<Iend;i++) {
  if (i>0) { MatSetValue(A,i,i-1,-1.0,INSERT_VALUES); }
  if (i<n-1) { MatSetValue(A,i,i+1,-1.0,INSERT_VALUES); }
  MatSetValue(A,i,i,2.0,INSERT_VALUES);
}
MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);
```

【ポイント】
他に,
Mat Aのサイズを(n*n)x(n*n)に変更する
また、
長さn*nのベクトルだが必要に応じて
(n,n)の２次元配列と処理する。

```
MatGetOwnershipRange(A,&Istart,&Iend);
for (i=Istart;i<Iend;i++) {
  ib=i/n; i0=i%n;
  if (ib>0) { MatSetValue(A,i,i-n,-1.0,INSERT_VALUES); }
  if (ib<n-1) { MatSetValue(A,i,i+n,-1.0,INSERT_VALUES); }
  if ( i0>0 ) { MatSetValue(A,i,i-1,-1.0,INSERT_VALUES); }
  if ( i0<n-1 ) { MatSetValue(A,I,i+1,-1.0,INSERT_VALUES); }
  MatSetValue(A,i,i,4.0,INSERT_VALUES);
}
MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);
```

Play with SLEPC

Computer simulations create the future

- **変更したプログラム(出力部も変更済み)**

$ ./ex1-2d -n 10 -eps_nev 3 -eps_tol 1e-7

2-D Laplacian Eigenproblem, n=10

Number of iterations of the method: 4
Solution method: krylovschur

Number of requested eigenvalues: 3
Stopping condition: tol=1e-07, maxit=100
Number of converged eigenpairs: 4

k	||Ax-kx||/||kx||
7.837972	5.4621e-09
7.601493	6.44101e-08
7.365014	1.2931e-08
7.228707	1.40955e-08

$ ./ex1-2d -n 10 -eps_nev 3 -eps_ncv 24

2-D Laplacian Eigenproblem, n=10

Number of iterations of the method: 3
Solution method: krylovschur

Number of requested eigenvalues: 3
Stopping condition: tol=1e-08, maxit=100
Number of converged eigenpairs: 5

k	||Ax-kx||/||kx||
7.837972	2.40527e-14
7.601493	1.39769e-13
7.365014	3.14186e-11
7.228707	6.57291e-11
6.992229	1.14673e-09

$ ./ex1-2d -n 10 -eps_nev 4 -eps_type lanczos

2-D Laplacian Eigenproblem, n=10

Number of iterations of the method: 10
Solution method: lanczos

Number of requested eigenvalues: 4
Stopping condition: tol=1e-08, maxit=100
Number of converged eigenpairs: 4

k	||Ax-kx||/||kx||
7.837972	1.60317e-09
7.601493	1.72603e-10
7.601493	3.30641e-09
7.365014	2.12916e-09

上級編：Stencilを使う

Computer simulations create the future

- Stencilは反復法系統の解法でAxのような計算を任意の場所で考えるとき

v(i,j)=
Top(i,j)*v(i,j+1)+Center(i,j)*v(i,j)+Bottom(i,j)*v(i,j-1) :5点ステンシル
+Left(i,j)*v(i-1,j)+Right(i,j)*v(i+1,j)

行列の替わりに、「上記係数行列を渡す」か「上記計算を行うルーチンそのものを渡す」ことでAxの計算は実現できる

PETScはstencil用の特別な計算手順が用意されている。

ただし、連立一次方程式を解く場合にのみ提供

固有値計算(SLEPc)の場合はShellという別の行列構造(MatCreateShell)とと関数登録(MatShellSetOperation)の機構を利用する。

PETSc のKSP solverでStencil計算

- **何をすべきか**
  - サンプルコードをダウンロード
http://www.mcs.anl.gov/petsc/petsc-current/src/ksp/ksp/examples/tutorials/ex50.c
  - コンパイル
  - キーポイントは i) PETScの初期化過程と行列生成方法, ii) ソルバールーチンの呼び出し方

PETScはDMDA (distributed multi-dimentional array format)をサポートする

- ３つのコールバック関数と１ユーティリティ関数を用意する必要がある
  - **ComputeRHS,**
  - **ComputeStencil,**
  - **SetInitialGuess,** and
  - **GetComputedResult.**

- 必要に応じて DMハンドラに上記関数情報を登録.
- KSPSolve() によって連立一次方程式が解かれる

PETSc のKSP solverでStencil計算

Computer simulations create the future

1. **ComputeStencil**
   - ２次元ヤコビ問題の場合5点ステンシル

$$u_{i,j}^{new} := au_{i-1,j} + bu_{i,j-1} + cu_{i,j} + du_{i+1,j} + eu_{i,j+1} + f_{i,j}$$

   - 上記式を行列 x ベクトルの表現と考える.

2. **ComputeRHS**
   - 反復法の中で右辺ベクトルを更新するときの手続き

3. **SetInitialGuess**
   - 初期値を設定する

4. **GetComputedResult.**
   - 結果を取り出す方法

Play with PETSc

Computer simulations create the future

- **プログラム実行例**

$ ./ex50 –ksp_monitor

```
0 KSP Residual norm 8.461650784989e-02
 1 KSP Residual norm 2.651391244119e-02
 2 KSP Residual norm 7.393505623135e-03
 3 KSP Residual norm 2.716741858701e-03
 4 KSP Residual norm 1.227939334146e-03
 5 KSP Residual norm 7.992911579280e-04
 6 KSP Residual norm 4.352343260567e-04
 7 KSP Residual norm 1.220838982121e-04
 8 KSP Residual norm 1.986831847167e-05
 9 KSP Residual norm 5.498079664434e-06
10 KSP Residual norm 1.706579187483e-06
11 KSP Residual norm 2.864431046042e-07
```

./ex50 -da_grid_x 120 -da_grid_y 120 -ksp_type cg -pc_type none -ksp_monitor

```
0 KSP Residual norm 4.166666666667e-03
 1 KSP Residual norm 1.397528003165e-15
```

$ ./ex50 -da_grid_x 120 -da_grid_y 120 -ksp_type gmres -pc_type mg -ksp_monitor

```
0 KSP Residual norm 3.644879484287e-02
 1 KSP Residual norm 2.829277489238e-02
 2 KSP Residual norm 1.108114249011e-02
 3 KSP Residual norm 4.923852014037e-03
 4 KSP Residual norm 2.675987120928e-03
 5 KSP Residual norm 1.645371666521e-03
 6 KSP Residual norm 1.078492505017e-03
 7 KSP Residual norm 7.453947736021e-04
 8 KSP Residual norm 5.348142516896e-04
 9 KSP Residual norm 3.896019613535e-04
10 KSP Residual norm 2.853075689015e-04
11 KSP Residual norm 1.950277383544e-04
12 KSP Residual norm 6.674151272888e-05
13 KSP Residual norm 1.247668851444e-05
14 KSP Residual norm 9.491976417721e-06
15 KSP Residual norm 7.982091088757e-06
16 KSP Residual norm 6.666275824338e-06
17 KSP Residual norm 5.576472192563e-06
```

まとめ

- **代表的な並列数値計算ライブラリ**
  - ScaLAPACK
  - EigenExa (時間の都合で割愛)
  - PETSc

  他、逐次版でLAPACK, BLASなど

- **ScaLAPACK (Fortranのみ説明)**
  - 2次元プロセスグリッド、ブロック分割された行列とベクトル
  - ハンドラを使ってさまざまな情報を管理（PETScも同様）
- **PETSc**
  - 疎行列だけでなくstencil型データ構造にも対応
  - 手続きが決まっているので、サンプルコードを参考にして必要な部分を追加拡張していく。
  - 実行時の引数やプログラムコード内で設定変更可能