

# 固有値計算法

RIKEN AICS HPC Spring School

今村俊幸　理化学研究所AICS

2014/3/6 9:00～12:00

K computer

# 本日の講義内容

- 固有値(線形代数)と応用問題
  - 振動問題
  - ネットワーク定常問題
- 固有値計算アルゴリズム
  - 密行列
    - べき乗法
    - ヤコビ法
    - ハウスホルダー三重対角＋分割統治法＋逆変換
  - 疎行列
    - ランチョス法
    - ヤコビ・デビッドソン法
    - その他
- 固有値計算ソフトウェア
  - ScaLAPACK
  - EigenExa
  - PETSc

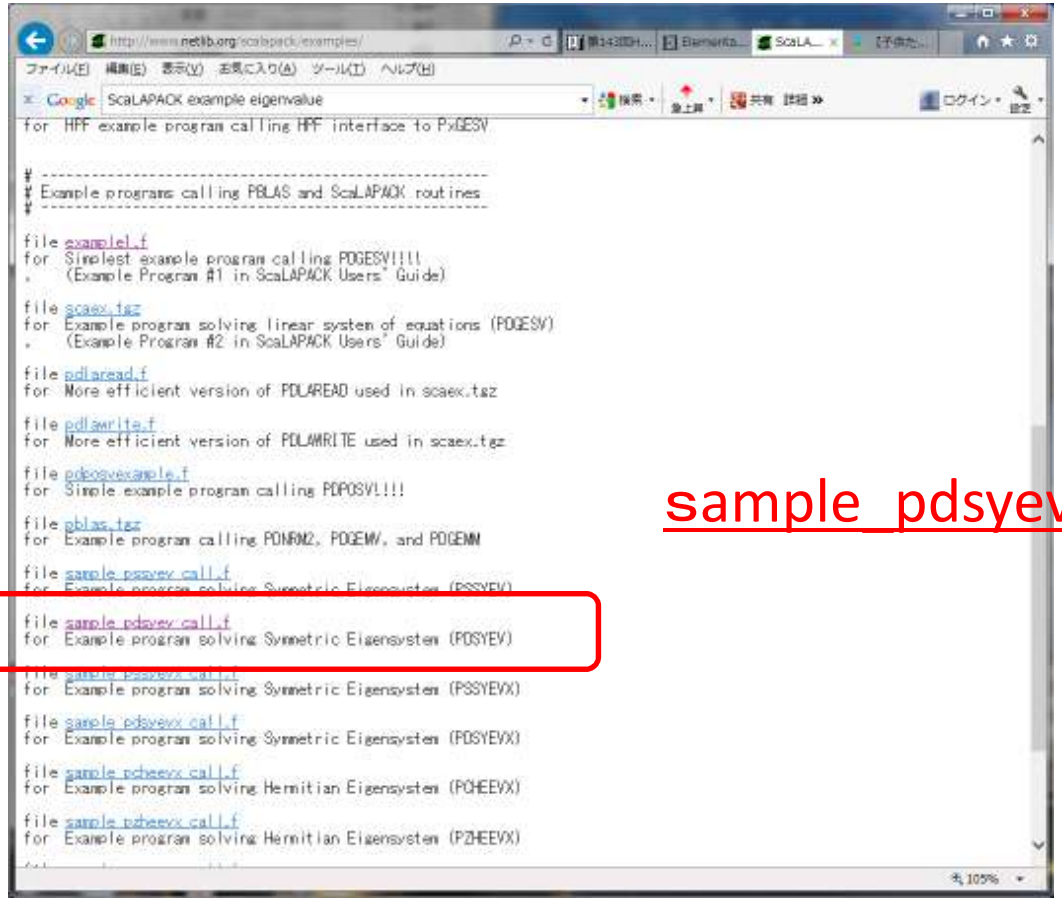
# ライブラリ使用法

ScaLAPACK, EigenExa, SLEPCを実際に使ってみる


# ScaLAPACKの利用方法

# ScaLAPACKサイトの例題より

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




sample_pdsyev_call.fをダウンロードする.

# How to use ScaLAPACK

- Link方法, KもしくはFX10の環境で

```
% mpifrt –o exe sample_pdsyev_call.f –SCALAPACK –SSL2BLAMP
```

- 実行:

```
#!/bin/bash
#PJM -L "rscgrp=school"
#PJM -L "node=2x2"
#PJM -L "elapse=00:05:00"
#PJM –mpi “proc=4”
#PJM -j

export OMP_NUM_THREADS=16

mpiexec ./exe
```

# 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

- 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
```

# やってみよう！

- PDLAMODHILBを改良して、自由な対称行列を入力として固有値計算をしてみよう。

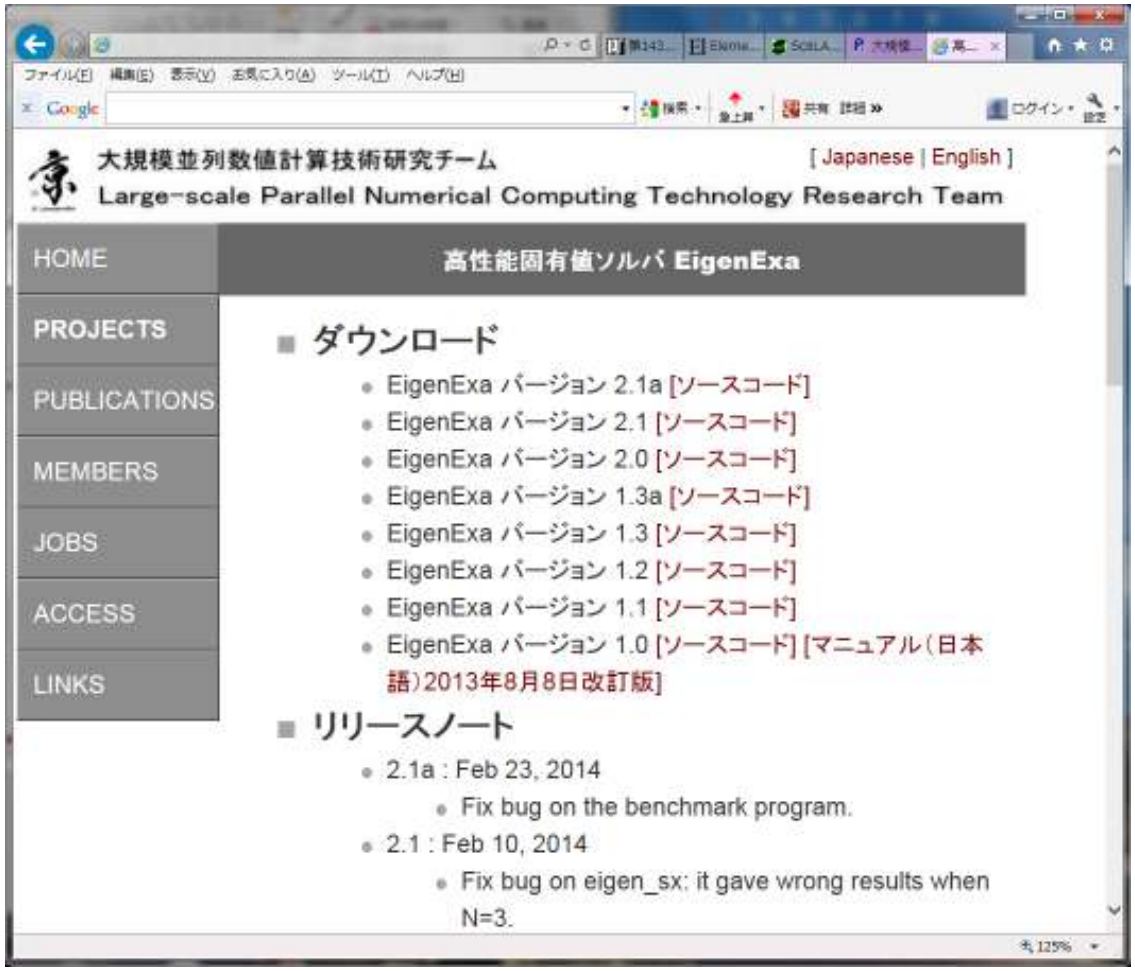
- また、固定化されている行列次元(N), プロセス数（NPROW, NPCOL）を変えて、計算時間の変化を見てみよう！

- 更に、余裕のある人は固有値求解関数をpdsyevdなどに変更してどうなるかを調べてみよう。


# EigenExaの使用方法

# EigenExaのダウンロード

http://www.aics.riken.jp/labs/lpnctrt/EigenExa.html




最新版をダウンロードしてください

# How to use EigenExa

- Build & Link方法, KもしくはFX10の環境で

```
% make test
自身でリンクする場合は
% mpifrt –o exe foo.f –lEigenExa –SCALAPACK –SSL2BLAMP
```

- 実行:

```
#!/bin/bash
#PJM -L "rscgrp=school"
#PJM -L "node=2x2"
#PJM -L "elapse=00:05:00"
#PJM –mpi “proc=4”
#PJM -j

export OMP_NUM_THREADS=16

mpiexec ./eigenexa_benchmark
```

# Let's learn the sample code

- 行列生成関数: mat_set
- 固有値計算関数: eigen_sx

ScaLAPACK同様に、初期化(eigen_init)や終了(eigen_free)必要。

EigenExaは2次元サイクリックサイクリック分割（具体的にはCOL,ROW共にNB=1固定の場合）の範囲ではScaLAPACKとコンパチであるので、相互の関数を利用し合うことができる。

従って、NB=1で行列のdescriptorが宣言されているので、PDLMODHILBで作成した配列をEigenExaに渡して解くこともできる。

# やってみよう！

- 行列次元(N), プロセス数（NPROW, NPCOL）を変えて、計算時間の変化を見てみよう！

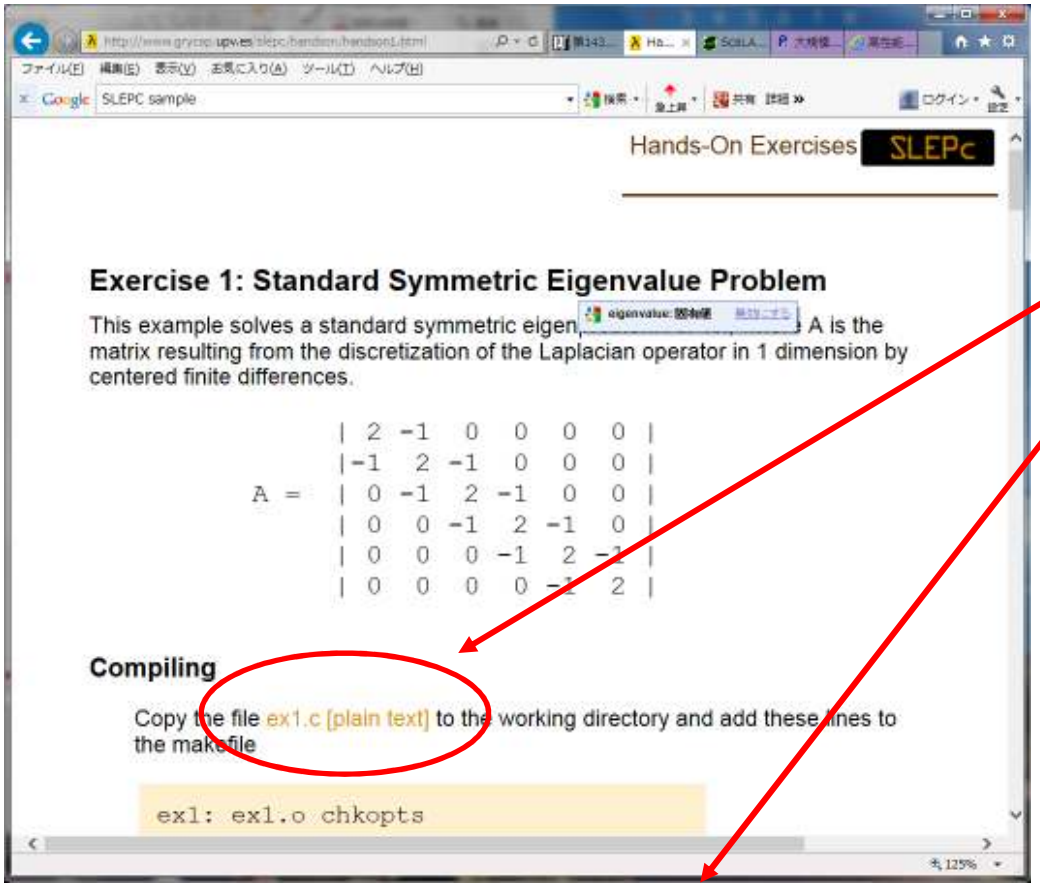
- 更に、余裕のある人はScaLAPACKのサンプルコード内のPDLAMODHILBを組み込んで、様々な行列の固有値求解をEigenExaにさせてみよう。

# PETSc(SLEPC)の使い方

SLEPCの公式Hands-on exercises
サイトの題材を利用する

神大FX10にSLEPCがなければこの章はとりやめ

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




ex1.cを使用します

ex1f.Fを使用します

（スクロールした下の方にあります）

**/opt/aics-ss/examplesの下にも一通りおいていますので、そちらをアクセスしてください**

# コンパイル&リンク

Makefile

```
ARCH = arch-fujitsu-sparc64fx-opt
PETSC_DIR = /opt/aics-ss/petsc-3.3-p5
SLEPC_DIR = /opt/aics-ss/slepc-3.3-p4
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 ¥
      -SSL2BLAMP

all: ex1 ex1f
ex1: ex1.o
      mpifccpx -o ex1 ex1.o $(LDFLAGS)
ex1f: ex1f.o
      mpifrtpx -o ex1f ex1f.o $(LDFLAGS)
ex1.o: ex1.c
      mpifccpx -c ex1.c -Xg $(INCPATH)
ex1f.o: ex1f.F
      mpifrtpx -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

- 実行

```
#!/bin/bash
#PJM -L "rscgrp=school"
#PJM -L "node=2x2"
#PJM -L "elapse=00:05:00"
#PJM –mpi “proc=4”
#PJM -j

export OMP_NUM_THREADS=16

echo ‘C-version’
mpiexec ./ex1 –n 100
echo ‘F90-version’
mpiexec ./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

- 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?

- 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?

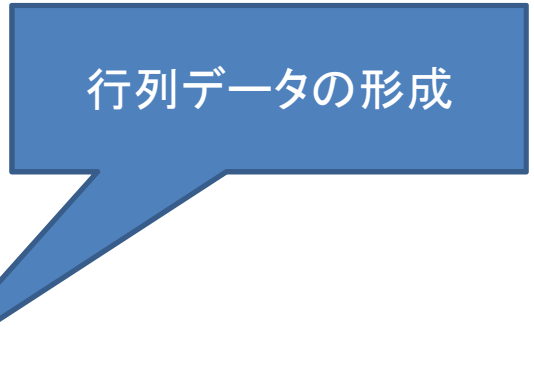
- PETScは内部データを柔軟に制御している。PETScが管理するため、使用者からは実態は見えない。行列ハンドラ変数 A を使ってアクセスする。内部フォーマットはいくつか存在する。

```
! 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?

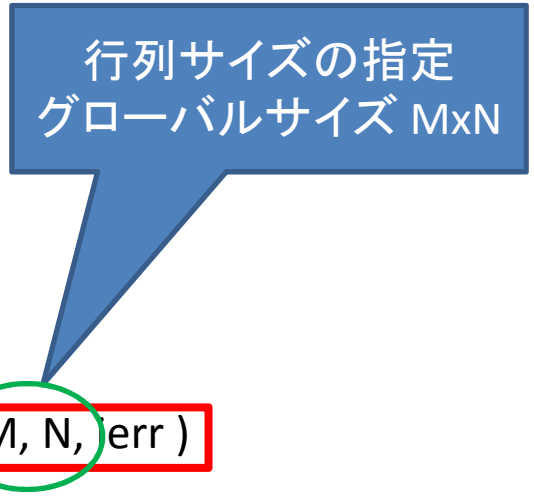
- PETScは内部データを柔軟に制御している。PETScが管理するため、使用者からは実態は見えない。行列ハンドラ変数 A を使ってアクセスする。内部フォーマットはいくつか存在する。

```
! 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?

- 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?

- 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)
```

セットされた配列データをアセンブルする

# やってみよう！余裕ある人向

- 少し難しくなりますが、長方領域板振動を表現する方程式、重調和関数の方程式

$$D\left(\frac{\partial^4 u}{\partial x^4}+2\frac{\partial^4 u}{\partial x^2\partial y^2}+\frac{\partial^4 u}{\partial y^4}\right)+\nu\frac{\partial^2 u}{\partial t^2}=0$$

$$D=\frac{Eh^3}{12(1-\mu^2)},\ \nu=\rho h$$

- (E: ヤング率, h: 板厚, ρ: 密度, μ: ポアソン比)
- 変数分離法により次の固有方程式を得る

$$\frac{\partial^4 u}{\partial x^4}+2\frac{\partial^4 u}{\partial x^2\partial y^2}+\frac{\partial^4 u}{\partial y^4}=\lambda u$$

# やってみよう！

- 固定境界条件(ディリクレ条件)のもと差分化して

$$X_m = \begin{bmatrix} 7 & -4 & 1 & & & \\ -4 & 6 & -4 & 1 & & \\ & & & & & \\ & & 1 & -4 & 6 & -4 \\ & & & 1 & -4 & 7 \end{bmatrix}_{m\times m}$$

$$Y_m = \begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & & & & & \\ & & 1 & -2 & 1 & \\ & & & 1 & -2 & \end{bmatrix}_{m\times m}$$

# やってみよう！

- テンソル積の表示を使って

$$A = \frac{1}{h_x^4}(I_{N_y-1} \otimes A_{N_x-1}) + \frac{2}{h_x^2 h_y^2}(B_{N_y-1} \otimes B_{N_x-1}) + \frac{1}{h_y^4}(A_{N_y-1} \otimes I_{N_x-1})$$

- 簡単化のため正方形状として、Nx=Ny=m, hx=hy=1とする.

$$A = \begin{bmatrix} X_m & & & \\ & X_m & & \\ & & \ddots & \\ & & & X_m \end{bmatrix} + 2 \begin{bmatrix} -2Y_m & Y_m & & & \\ Y_m & -2Y_m & Y_m & & \\ & & & \ddots & \\ & & & Y_m & -2Y_m \end{bmatrix}$$

$$+ \begin{bmatrix} 7I & -4I & I & & \\ -4I & 6I & -4I & I & \\ & & & \ddots & \\ & & I & -4I & 7I \end{bmatrix}$$

# やってみよう！

- この様に作られる行列Aの固有値と固有ベクトルを計算して、得られた固有値の大きい方から順に選んで対応する固有ベクトルをm×mに並べ換えたデータとして可視化するとどうなるだろうか？

- 可視化にはgnuplotのplot3dを使ってみましょう。