数値計算ライブラリの使い方

疎行列向けライブラリを中心に学習

KOBE HPC スプリングスクール（中級）2021【Zoom開催】
講師：今村俊幸 (理化学研究所計算科学研究センター)

OBCXでの実行時の注意事項

- 環境変数LANGを適切に設定（抜けていることがあるので）

```
% export LANG=C
```

- 3日目午前は, PETScというライブラリを利用しますので、ログインノード上で以下を実行して、モジュールを有効にしてください。

```
% module load impi petsc phdf5
```

- makeコマンド実行時に ‘depend_c.inc:1: *** multiple target patterns. Stop.’, といったメッセージがあった場合、上記文モジュールが正しく読み込まれていない可能性があります。”module list”とタイプしてモジュールとそのバージョンを確認して下さい。

```
Currently Loaded Modulefiles:
  1) impi/2019.5.281    3) phdf5/1.10.5
  2) intel/2019.5.281   4) petsc/3.11.2
```

ほとんどのソースファイルとヒントは、

Computer simulations create the future

/work/gt65/t65021/share/

PETSc/ → PETSc sample code, a job-script, and Makefile
SLEPc/ → SLEPc sample code, a job-script, and Makefile

- **Please ‘cd’ your work directory, then ‘cp -R’ the directory (‘%’ is a prompt, and do not type). You will see**

```
% cd /work/gt65/tXXXXX
% cp -R /work/gt65/t65021/share ./third_day
% cd third_day
% ls
PETSc  SLEPc
```

本日の講義

- **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
- **簡単な固有値計算アルゴリズム紹介**
  - 密行列
  - 疎行列
- **一般的な数値計算ライブラリの紹介**
- **固有値計算ソフトウェアPETSc+SLEPcを使った演習**
  - OBCXでの実演習
    - （連立一次方程式）
    - 固有値計算
  - 応用問題への取り組み

はじめに

表記法について

- **ベクトル：　小文字で表記（'→'記号はつけない、太文字にもしない）。**

$$v \in \mathrm{R}^m$$ m行(縦方向)

- **行列：　大文字イタリックで表記する。**

$$A \in \mathrm{R}^{m \times n}$$ m行(縦方向), n列(横方向)

$$A = [a_1, a_2, \cdots, a_n]$$ の様に, ベクトルを並べた記法使用も

- **数体：　特に指示がなければ実数であり、複素数は以下の記法を使用する。ただし、i　は添字としても使用するので、文脈で判断。**

$$a + ib \in \mathrm{C},\ a, b \in \mathrm{R}, i = \sqrt{-1}, i^2 = -1$$

表記法について

- **ノルム：　縦二本線表記（原則２ノルム）**

$$\|v\| = \left(\sum_{j=1}^{m} |v_j|^2 \right)^{\frac{1}{2}}$$

- **行列のノルム：　上記のノルムから定義、縦二本線表記**

$$\|A\| = \sup_{\|x\| \neq 0} \frac{\|Ax\|}{\|x\|}$$

- **行列式：　縦１本線表記**
- **条件数：**

$$\kappa(A) = \|A\| \|A^{-1}\|$$

表記法について

- **連立一次方程式**

$$Ax = b$$

  - 記号を適時変えることがあるが、基本は上記記号を用いることが多い
  - 残差ベクトルを $r = b - Ax$ として定義する。
  - （未知の真解との近さを評価できないため）、残差ベクトルのノルムで近似解を評価する。
- **固有値固有ベクトル**

$$Ax = \lambda x$$

  - **固有値**をギリシャ記号のラムダ(λ) もしくはシータ(θ)
  - **固有ベクトル**は x もしくは s を用いて表現するが、適時記号を変えるので注意。
  - 固有値と対応する固有ベクトルを**「固有対」**と呼ぶ
  - 固有値の記号に対応した大文字の行列は、固有値を対角に並べてできる対角行列とする
  - また、固有ベクトルを並べてできるベクトルも同様に同じ文字の大文字で表記することがある。

固有値(線形代数)と応用問題

主に線形代数の復習から

固有値とは

- **工学における現象解析、システム解析に利用する**
  - 構造物の耐震振動解析
  - ネットワークなどの定常状態解析

$$Ax = \lambda x$$

- **計算方法：線形代数の理論上は次の特性多項式を解けばよい**

$$|A - \lambda I_n| = 0$$

  - コンピュータ上で行列式の計算は難しい。多項式の求解は？ニュートン法でできるか？

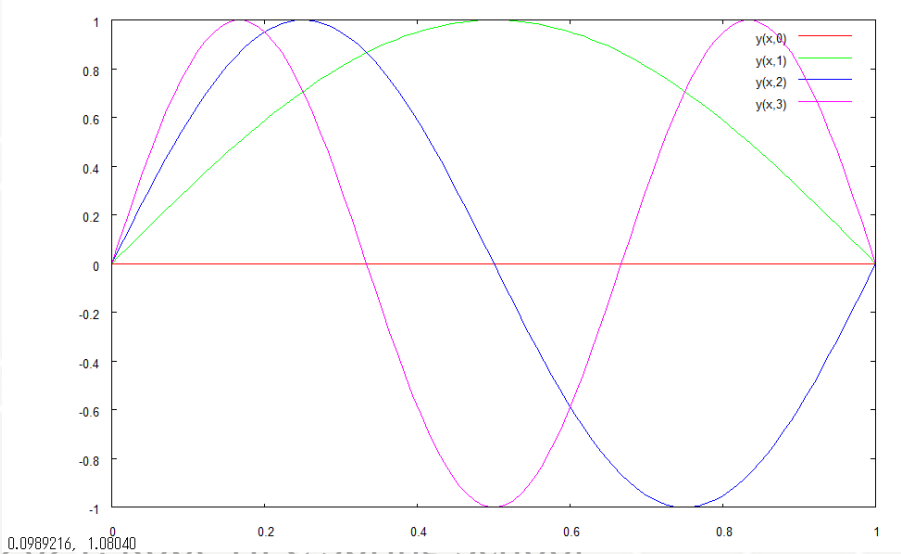
固有値計算の例

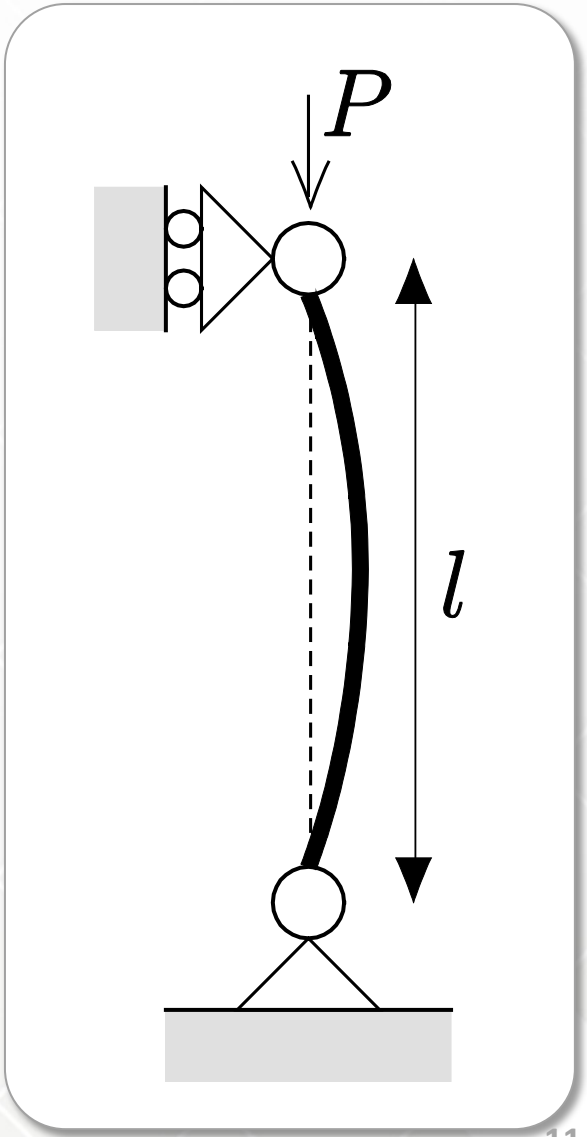
- **柱の変形**

$$\left.\begin{aligned} \frac{d^2y}{dx^2} + \frac{Py}{EI} &= 0 \\ y(0) = y(l) &= 0 \end{aligned}\right\}$$

- 境界条件を考慮して解は

$$y(x) = a \sin \sqrt{\frac{P}{EI}} \qquad P = \frac{n^2\pi^2 EI}{l^2}$$

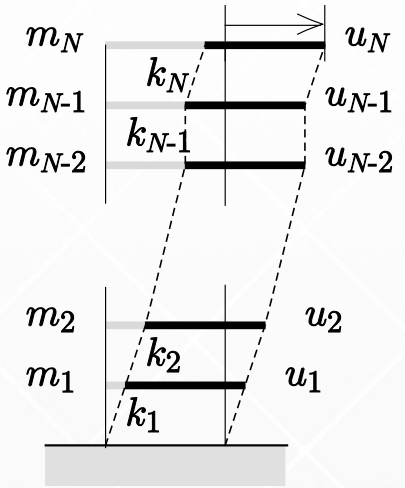




多自由度系の振動

- **支配方程式系**

$$\begin{bmatrix} m_1 & & & \\ & m_2 & & \\ & & \ddots & \\ & & & m_N \end{bmatrix} \begin{bmatrix} \dot{u}_1 \\ \dot{u}_2 \\ \vdots \\ \dot{u}_N \end{bmatrix} + \begin{bmatrix} k_1+k_2 & -k_2 & & \\ -k_2 & k_2+k_3 & -k_3 & \\ & & \ddots & \\ & & -k_N & k_N \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{bmatrix} = 0$$



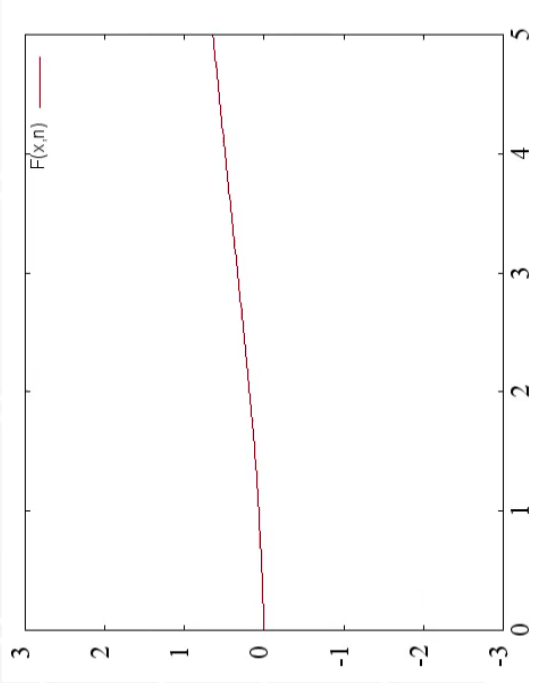
- 変数分離法で解ける

$$u_i = (a\sin\omega t + b\cos\omega t)y_i(x)$$

- 行列・ベクトルで整理すると

$$(M^{-1}K)y = (\omega^2)y$$

$Ay = \lambda y$ の形をしている



机上で計算できる場合

Computer simulations create the future

$$m_1 = m_2 = \cdots = m_N = m,$$

$$k_1 = m_2 = \cdots = k_N = k$$

$k/m = s$ とおけば

$$\lambda \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{pmatrix} = \begin{pmatrix} 2s & -s & & & \\ -s & 2s & -s & & \\ & \ddots & \ddots & \ddots & \\ & & -s & 2s & -s \\ & & & -s & s \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{pmatrix}$$

机上で計算できる場合

$$\lambda \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{pmatrix} = \begin{pmatrix} 2s & -s & & \\ -s & 2s & -s & \\ & & \ddots & \\ & & -s & s \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{pmatrix}$$

$-1+2\omega-\omega^2=\lambda'\omega\ (\lambda'=\lambda/s)$
を満足する2つのω（複素数）に対して,

$u_1=[\omega_1,\omega_1^2,\ldots,\omega_1^N]^T,$
$u_2=[\omega_2,\omega_2^2,\ldots,\omega_2^N]^T$ とおき

$u=\alpha_1u_1+\alpha_2u_2$ なる線形結合に対して、両項に

$\alpha_1\omega_1^0+\alpha_2\omega_2^0=0,\ \alpha_1\omega_1^{N+1}+\alpha_2\omega_2^{N+1}=\alpha_1\omega_1^N+\alpha_2\omega_2^N$ を課せば、

λが固有値, uが対応する固有ベクトルとなる

$-1+2\omega-\omega^2=\lambda'\omega$ より $\omega_1\omega_2=1,\ \omega_1+\omega_2=2-\lambda'$ さらに$\alpha_1=-\alpha_2$

$$\omega_1^{N+1}(\omega_1^{N+1}-\omega_1^N)=\omega_1^{N+1}(\omega_2^{N+1}-\omega_2^N)=1-\omega_1\Rightarrow\omega_1^{2N+1}=-1$$

$$\omega_1=e^{\frac{k\pi i}{2N+1}},\ \omega_2=e^{-\frac{k\pi i}{2N+1}},\ \omega_1^j-\omega_2^j=2\sin\frac{jk\pi}{2N+1},\omega_1+\omega_2=2\cos\frac{k\pi}{2N+1}$$

$$\lambda'=2-(\omega_1+\omega_2)=2(1-\cos\frac{k\pi}{2N+1})=4\sin^2\frac{k\pi}{2(2N+1)},\ (k=1,\ldots,N)$$

固有値とは

- **工学における現象解析、システム解析に利用する**
  - 構造物の耐震振動解析
  - ネットワークなどの定常状態解析

$$Ax = \lambda x$$

- **計算方法：線形代数の理論上は次の特性多項式を解けばよい**

$$|A - \lambda I_n| = 0$$

  - コンピュータ上で行列式の計算は難しい。多項式の求解は？ニュートン法でできるか？

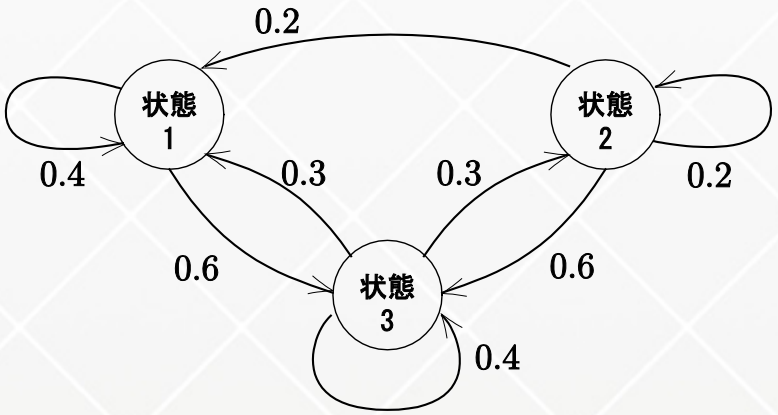
ネットワーク解析

- **ある時刻にサイトをアクセスしている人が、次の時刻に別のサイトに移動する確率が、サイトとサイトの間で決まっているとしよう。**

$q = [q_1, q_2, q_3]^T$ をサイトにいる人の数とする。

遷移確率に従って人が移動すれば

$$q^{(k+1)} = Pq^{(k)}$$



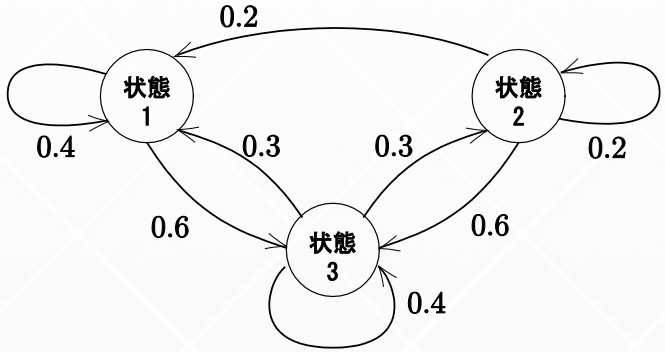
定常状態(k→∞)では

$$q = Pq$$

ネットワーク解析

- **実際に人が移動する状態を確認してみよう**

$$P = \begin{bmatrix} .4 & .2 & .3 \\ 0 & .2 & .3 \\ .6 & .6 & .4 \end{bmatrix}$$



＊[100,0,0]の状態からスタート

[100,0,0]->[40,0,60]->[34,18,48]->[31.6, 18, 50.4]

-> [31.36,18.72,49,92] -> [31.264,18.720,50.016]

-> … -> [31.25, 18.75, 50.0]

定常的には100人のうちの半数の50人がサイト３を見ていることになる。

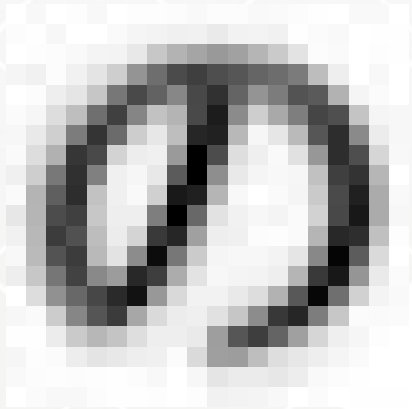
固有「の」：特異値解析

- **特徴量の主成分解析**
  - ベクトルとして表現された複数データの主たる特徴
  - 類似性の計算
  - 平均的な顔の定義

$$\sum \frac{x_i x_i^T}{\|x\|^2}$$

（例）固有「の」

スキャンデータから切り出した１０個の「の」の最も平均的な特徴を示す「の」を計算



密行列向け
固有値計算アルゴリズム

べき乗法

（絶対値）最大固有値の計算方法

- べき乗法
  - 初期ベクトルを選択
  - ベクトルに行列を乗じて正規化する
  - ベクトルが収束したら
    - ノルムが絶対値最大固有値
    - 対応する固有ベクトル

/* べき乗法 */

$x^{(0)}$: 正規化された初期ベクトル

do

$$v = Ax^{(k)}$$

$$\lambda^{(k)} = (x^{(k)}, v)$$

$$x^{(k+1)} = \frac{v}{\|v\|}$$

$$k \leftarrow k + 1$$

while $\left| \|v\| - |\lambda^{(k)}| \right| > \varepsilon$

べき乗法

- 行列 A に x を乗じて、正規化し x と置き直す。これを繰り返す。下に、実際にどのような計算が進行するかを示します。

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 2 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad x = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} .447 \\ .894 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} .745 \\ .596 \\ .298 \end{bmatrix} \Rightarrow \begin{bmatrix} .619 \\ .761 \\ .190 \end{bmatrix}$$

2.23　　3.00　　3.13

$$\Rightarrow \begin{bmatrix} .678 \\ .693 \\ .241 \end{bmatrix} \Rightarrow \begin{bmatrix} .653 \\ .724 \\ .219 \end{bmatrix} \Rightarrow \begin{bmatrix} .664 \\ .711 \\ .229 \end{bmatrix} \Rightarrow \begin{bmatrix} .661 \\ .714 \\ .226 \end{bmatrix} \Rightarrow \begin{bmatrix} .660 \\ .715 \\ .225 \end{bmatrix} \Rightarrow \begin{bmatrix} .661 \\ .715 \\ .226 \end{bmatrix}$$

3.15　　3.163　　3.164　　3.1642　　3.1642　　3.16425

べき乗法

- **反復系統の解法の基本であるので、原理と解けるための条件をまとめる**
  - 条件：A の絶対固有値が全て異なるとする(最大だけで十分)
  - 初期ベクトルは固有ベクトルが一次独立なので線形結合で書ける

$$|\lambda_1| > |\lambda_2| > \cdots |\lambda_n|$$

  - ここで、次のように $x^{(k)}$ を更新していく

$$x^{(0)} = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$$

$$v^{(k+1)} = Ax^{(k)}, x^{(k+1)} = v^{(k+1)} / \|v^{(k+1)}\|$$

べき乗法

- ここで c_1 は０でないと仮定すると、x の更新の方法から

$$x^{(1)} = \frac{1}{r_1} A x^{(0)} = \frac{1}{R_1}(c_1\lambda_1 x_1 + c_2\lambda_2 x_2 + \cdots + c_n\lambda_n x_n)$$
$$x^{(2)} = \frac{1}{r_2} A x^{(1)} = \frac{1}{R_2}(c_1\lambda_1^2 x_1 + c_2\lambda_2^2 x_2 + \cdots + c_n\lambda_n^2 x_n)$$

$R_j = \prod_{k=1}^{j} r_k$ とおく

$$\vdots$$
$$x^{(j)} = \frac{1}{r_j} A x^{(1)} = \frac{1}{R_j}(c_1\lambda_1^j x_1 + c_2\lambda_2^j x_2 + \cdots + c_n\lambda_n^j x_n)$$

$$\lim_{j\to\infty} \frac{R_j}{c_1\lambda_1^j} x^{(j)} = \lim_{j\to\infty} \left(x_1 + \frac{c_2}{c_1}\left(\frac{\lambda_2}{\lambda_1}\right)^j x_2 + \cdots + \frac{c_n}{c_1}\left(\frac{\lambda_n}{\lambda_1}\right)^j x_n \right) = x_1$$

べき乗法

- **ポイント**
  - 絶対値最大の固有値 $|\lambda_1|$ に対する固有ベクトルに収束
  - 絶対値最大固有値に重複があると、一般に収束しない（複数のベクトルが張る空間内で変化し続けます）

  - 絶対値最小固有値計算にも応用可能
    - $A^{-1}x = \frac{1}{\lambda}x$ が成り立つことから、

    逆行列の絶対値最大固有値は元の行列の絶対値最小固有値に対応
  - ある値(σ)に近い、固有値も計算可能

$$(A - \sigma I)^{-1}x = \frac{1}{\lambda - \sigma}x = \frac{1}{\lambda'}x \Rightarrow \lambda = \sigma + \lambda'$$

ヤコビ法

ヤコビ法

- 行列 A を $n \times n$ 対称行列とする。
- “全て”の固有値・固有ベクトルの組を求めたい。

/* ヤコビ法 */

$A^{(0)} \leftarrow A$: 初期行列

do

$\alpha = |a_{ij}| = \max_{l>m} |a_{lm}|$ ($A^{(k)}$ の下三角要素の中で絶対値最大) を選択.

if $a_{ii} = a_{jj}$ **then**

$\theta_k = \frac{\pi}{4}$

else

$\theta_k = \frac{1}{2} \tan^{-1} \frac{2a_{ij}}{a_{ii} - a_{jj}}$

endif

ギブンスの回転行列 $U(\theta_k)$ を計算 ($\ddots$ の部分は 1, その他は 0)

$$U(\theta_k) = \begin{bmatrix} \ddots & & & & \\ & \cos\theta_k & & -\sin\theta_k & \\ & & \ddots & & \\ & \sin\theta_k & & \cos\theta_k & \\ & & & & \ddots \end{bmatrix} \begin{matrix} \\ \leftarrow i \\ \\ \leftarrow j \\ \\ \end{matrix}$$

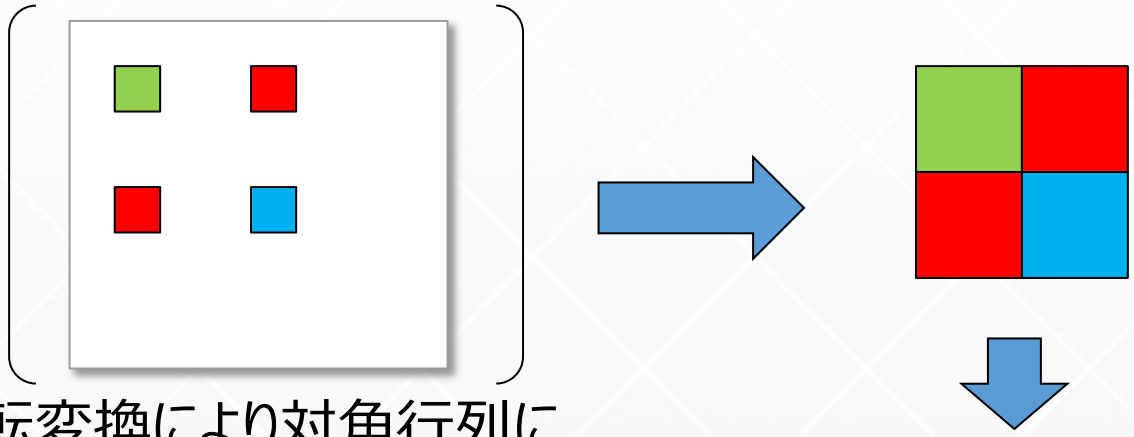
$A^{(k+1)} = U(\theta_k)^T A^{(k)} U(\theta_k)$

$k \leftarrow k + 1$

while $\alpha > \varepsilon$

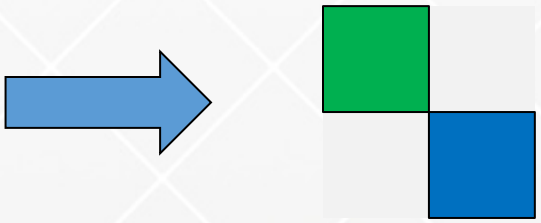
ヤコビ法

- **行列を逐次相似変換して対角行列に収束**
  - 非対角項の絶対値最大の２＊２小行列を取り出し



  - 回転変換により対角行列に

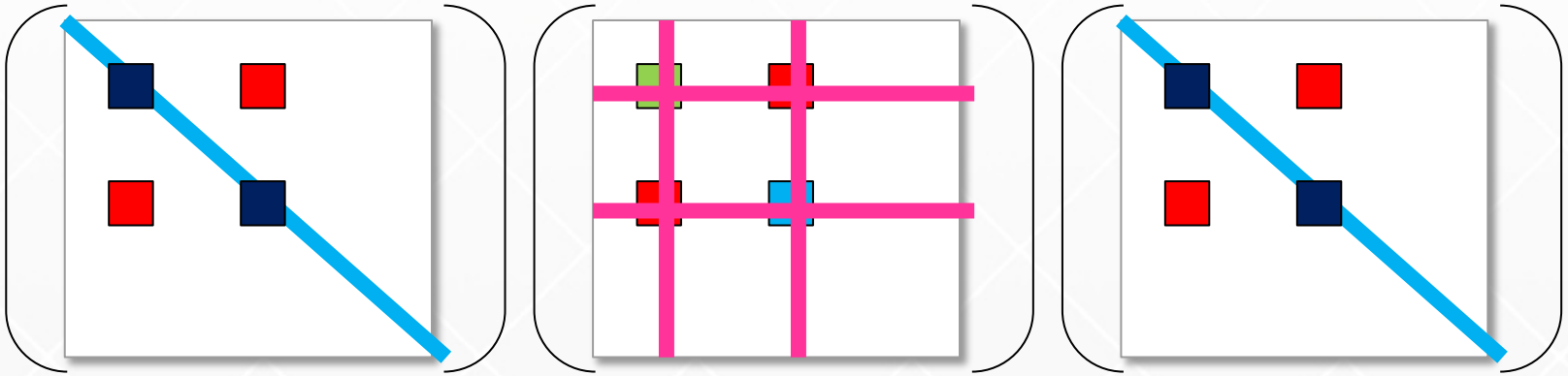
$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$



  - （アルゴリズムから） θ を定め、回転変換を左右より乗じる

ヤコビ法

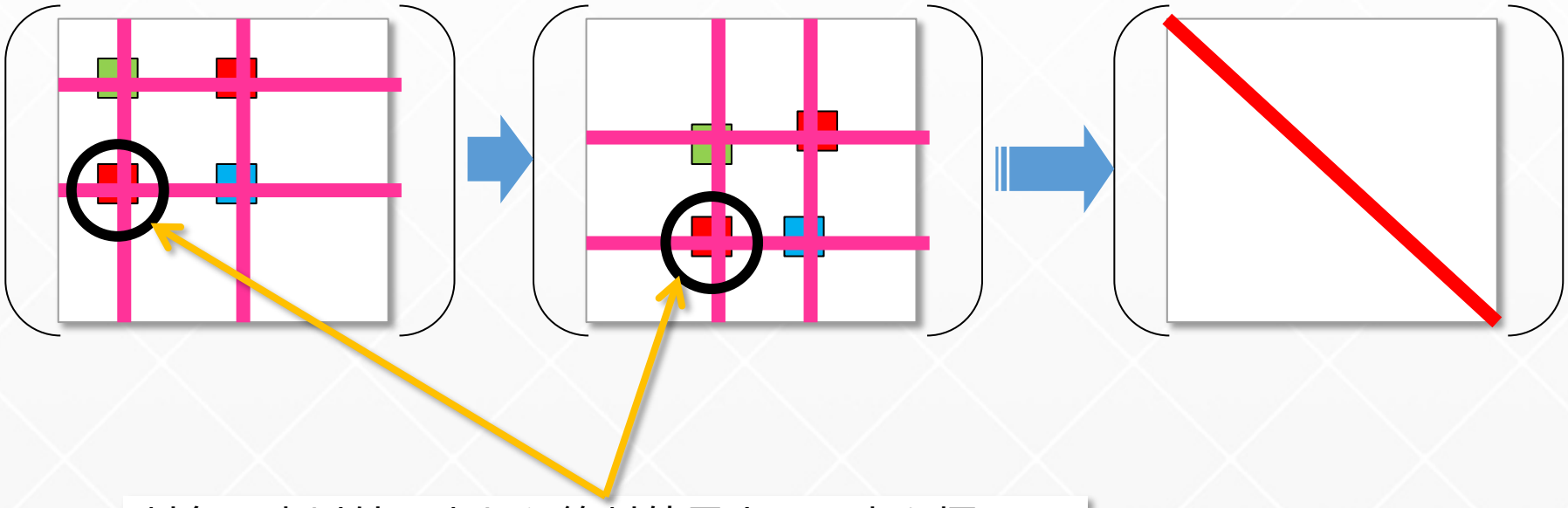
- **行列を逐次相似変換して対角行列に収束**
  - 実際は、下の箇所で計算



- **回転行列**は、$n \times n$　行列に拡大し、対角(青)に1をおく
- **行列** A が変換されるのは、ピンク色線の箇所
- この相似変換により、行列 A は徐々に対角行列に近づく
  - 数学的な収束の証明は後半に

ヤコビ法

- ポイント： 行列を逐次相似変換して対角行列に収束



対角要素以外の中から絶対値最大の要素を探し、
その要素を基準とした２ｘ２行列を基本とした
２ｘ２のGivens回転行列を求め作用させていく。

ヤコビ法

- 相似変換を繰り返すから

$$U_k^T U_{k-1}^T \cdots U_2^T U_1^T A U_1 U_2 \cdots U_{k-1} U_k \to \Lambda$$

- $U = U_1 U_2 \cdots U_{k-1} U_k$ とおけば

$$AU = U\Lambda$$

- U の各列ベクトルは固有ベクトルでもある。
- つまり、ヤコビ法の手続きを繰り返した結果得る対格行列の要素が**固有値**であり、その位置に対応する U の列ベクトルが**対応する固有ベクトル**である。

ヤコビ法の原理

- **はたして対角行列に収束しているのか？**

  a_{ij} **を0に変換するとき、i,jを含む行と列に変更が加わる。**

$$
\begin{cases}
a'_{i,k} = +\cos\theta a_{i,k} + \sin\theta a_{j,k} \\
a'_{j,k} = -\sin\theta a_{i,k} + \cos\theta a_{j,k} \\
a'_{i,i} = \cos^2\theta a_{i,i} + 2\sin\theta\cos\theta a_{i,j} + \sin^2\theta a_{j,j} \\
a'_{j,j} = \sin^2\theta a_{i,i} - 2\sin\theta\cos\theta a_{i,j} + \cos^2\theta a_{j,j}
\end{cases}
\qquad (i \neq j)
$$

$A^{(k+1)} = U(\theta_k)^T A^{(k)} U(\theta_k)$ に対して、対角を除いた成分の２乗和を定めると

$$
\Delta^{(k)} = \sum_{x \neq y} (a_{x,y}^{(k)})^2 \quad \rightarrow \quad \Delta^{(k+1)} = \Delta^{(k)} - (a_{i,j}^{(k)})^2 < \Delta^{(k)}
$$

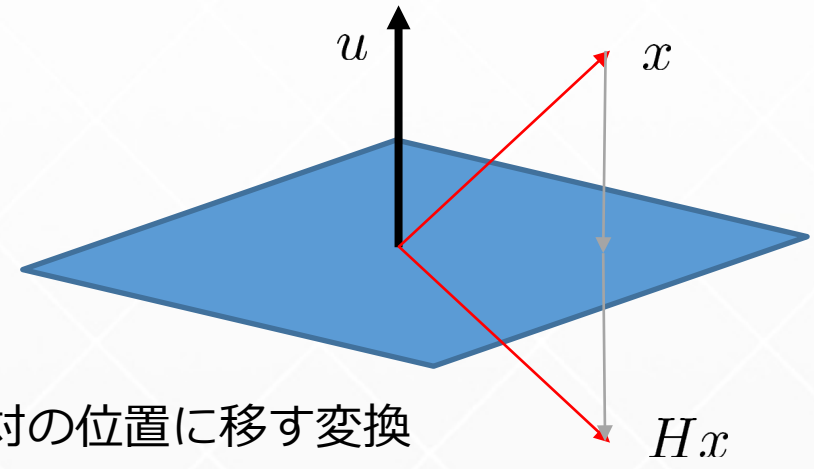
対角以外の部分の２乗和は単調減少 → 対角行列に収束する

ハウスホルダー三重対角法＋α

ハウスホルダー三重対角化

- **ハウスホルダー変換**

$$H = I - 2\frac{uu^T}{\|u\|^2}$$



ベクトル uを鏡線として、鏡面に反対の位置に移す変換

$$A = \begin{bmatrix} \alpha & a^T \\ a & \tilde{A} \end{bmatrix} \quad : \quad \text{対称行列}$$

$Ha = \|a\|e_1$ とするようなハウスホルダー変換を定めることができる。

$e_1 = (1, 0, \cdots, 0)^T$ なるベクトルで、 $u = a + \|a\|e_1$ ととればよい。

$$Ha = a - u\frac{2u^T a}{\|u\|^2} = a - (a + \|a\|e_1) = \|a\|e_1$$

$$\begin{bmatrix} \|u\|^2 = 2\|a\|(\|a\| + a_1) \\ 2u^T a = 2\|a\|(\|a\| + a_1) \end{bmatrix}$$

ハウスホルダー三重対角化

- **ハウスホルダー変換**

$$A = \begin{bmatrix} \alpha & a^T \\ a & \tilde{A} \end{bmatrix} \quad : \quad \text{対称行列}$$

$H_1 a = \|a\| e_1$ とするようなハウスホルダー変換を定めることができる。

$$P_1 = \begin{bmatrix} 1 & 0 \\ 0 & H_1 \end{bmatrix}$$ とブロック拡大表記した相似変換 P を定め、A に左右から作用

$$P_1 A P_1 = \begin{bmatrix} 1 & 0 \\ 0 & H_1 \end{bmatrix} \begin{bmatrix} \alpha & a^T \\ a & \tilde{A} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & H_1 \end{bmatrix} = \begin{bmatrix} \alpha & \|a\| e_1^T \\ \|a\| e_1 & H_1 \tilde{A} H_1 \end{bmatrix}$$

$A' = H_1 \tilde{A} H_1$ に対しても同様の変換を再帰的に実施しすれば, 左下の成分は対角成分以下に非ゼロ要素をもつ行列、すなわち三重対角行列（対称）T に最終的になる。

三重対角行列は要素数も少なく、固有値計算も容易。多くのアルゴリズムが存在する。

三重対角行列の固有値計算

- **ハウスホルダー三重対角化後の行列T**

$$P(A-\lambda I)P^T = T - \lambda I$$

と変形できるので、行列式を計算すると

$$|P||A-\lambda I||P^T| = |PP^T||A-\lambda I| = |A-\lambda I| = |T-\lambda I|$$

つまり、Tの固有値はAの固有値と同じである。

- **固有ベクトルは**

$$Tx = \lambda x \qquad (P^T T P)P^T x = P^T(\lambda x) \leftrightarrow Ay = \lambda y, y = P^T x$$

Tの固有ベクトルに対して P^T を作用（逆変換）すればよい

ＱＲ法

- ＱＲ分解

$A = QR, Q^T Q = QQ^T = I, R$ ： 上三角行列

今, $A^{(0)} = Q^{(0)} R^{(0)}$, $A^{(1)} = R^{(0)} Q^{(0)} = Q^{(1)} R^{(1)}$, …

なる手続きを進めていくと、最終的にある行列に収束する性質を使い固有値・固有ベクトルを計算する。

ただし、 $A = A^{(0)}$ は対称行列とする。

$A^{(1)} = {Q^{(0)}}^T A^{(0)} Q^{(0)}$ $A^{(2)} = {Q^{(1)}}^T A^{(1)} Q^{(1)}$ → $D = Q^T A Q$

これより、A の固有値は D の各成分,

固有ベクトルは $Q = Q^{(0)} Q^{(1)} \cdots$ の対応する列ベクトル

T 行列の更新手順では Q のみ必要で、$Q^T T Q$ は常に三重対角の構造を保持するため計算が容易

２分法

- **三重対角行列の特性多項式の計算**

$$f_n(\lambda) = |T - \lambda I| = \begin{vmatrix} a_1 - \lambda & b_1 & & & \\ b_1 & a_2 - \lambda & b_2 & & \\ & b_2 & a_3 - \lambda & b_3 & \\ & & & b_{n-1} & a_n - \lambda \end{vmatrix}$$

$$f_{n-1}(\lambda) = \begin{vmatrix} a_2 - \lambda & b_2 & & & \\ b_2 & a_3 - \lambda & b_3 & & \\ & & & b_{n-1} & a_n - \lambda \end{vmatrix}, \cdots$$

とおくと、

一般形として $f_k = (a_k - \lambda) f_{k-1} - b_k^2 f_{k-2} \quad f_{-1} = 0, f_0 = 1$

により特性多項式を求められる。ゼロ点を所謂２分法により求めればよい。
ただし、nが大きいときはオーバフローの危険があり、ストゥルムの数え上げが有効。

（２分法）

$f(x) < 0 < f(x')$ なる区間(x,x')に対して中点(x+x')/2の符号により区間縮小を行う

逆反復法

- 何らかの方法でTの固有値近似 $\tilde{\lambda}$ 、さらには固有ベクトルの近似 $\tilde{x}$ が得られたとする。

- べき乗法の箇所でも説明したように，$\tilde{\lambda}$ に近い固有ベクトルは $(T-\tilde{\lambda})^{-1}$ の絶対値最大固有値に対応する固有ベクトルである。
- 従って，

$$\tilde{x} \leftarrow (T - \tilde{\lambda} I)^{-1} \tilde{x}$$

により、固有ベクトルの精度を高める計算を行う。

ただし、本方法で求めた固有ベクトルは直交性の観点からはよくないので、直交化の手続きをしなくてはいけない

分割統治法

- **三重対角行列を適当な摂動により以下のようにする**

$$T = \begin{bmatrix} T_1 & \\ & T_2 \end{bmatrix} + \rho e_{k,k+1} e_{k,k+1}^T, \quad e_{k,k+1} = e_k + e_{k+1}$$

何らかの方法で T_1 と T_2 の固有値計算が為されたとする。それぞれの固有値と固有ベクトルを並べた行列(D_1 Q_1 など) を用いて

$$\begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix}^T T \begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix} = \begin{bmatrix} D_1 & \\ & D_2 \end{bmatrix} + \rho [Q_{1k}; Q_{21}][Q_{1k}; Q_{21}]^T$$

$$= D + \rho u u^T$$

と問題を簡単化できる(相似変換なので固有値はもとのものと同じ、固有ベクトルは逆変換が必要)

分割統治法

- **簡単化された問題，｛対角行列＋1階摂動｝**

$$C = D + \rho u u^T \quad \rightarrow \quad (D + \rho u u^T)x = \lambda x$$

適時変形を行っていけば次式を得る。

$$1 + \rho u^T (D - \lambda)^{-1} u = 1 + \rho \sum \frac{u_k^2}{d_k - \lambda} = 0$$

この有理方程式は、区間$[d_k, d_{k+1}]$ に解が存在することが分かっているので
独立に解くことが容易である。

- **固有ベクトル**

上記で求めた λ を用いて、以下の式より x を計算する。最終的には正規化が必要

$$x = (D - \lambda I)^{-1} u$$

ＱＲ法もう一回

- 先の説明でQR法を（対称）三重対角行列に適用したが、一般的な非対称行列に対しての有効な解法はQR法しかないのが現状である。
- 非対称行列の場合はハウスホルダー変換を実施しても、三重対角化はできず、ヘッセンベルグ形式（下副対角成分以下が０の行列）に変形してから、前述のＱＲ法を行い上三角行列に収束する性質を利用して固有値を計算する。（複素固有対がある場合は2x2の対角ブロックが出現）

$$\begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1.5 & .914 \\ 1.9 & .5 \end{bmatrix} \rightarrow \begin{bmatrix} 2.25 & 1.32 \\ .32 & -.25 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 2.43 & -.93 \\ .062 & -.434 \end{bmatrix} \rightarrow \rightarrow \rightarrow \begin{bmatrix} 2.412 & -1.0 \\ .000 & -.4142 \end{bmatrix}$$

ハウスホルダー逆変換

- **ハウスホルダー変換自身はそれ自身が逆変換でもあるので、作用させた逆順に作用していけばよい。**
- **近代的なアルゴリズムでは複数のハウスホルダー変換をまとめてブロック化する。**

$$(I - \beta_1 u_1 u_1^T)(I - \beta_2 u_2 u_2^T) = I - UCU^T$$

$$U = [u_1, u_2] \quad C = \begin{bmatrix} \beta_1 & -\beta_1 \beta_2 u_1^T u_2 \\ & \beta_2 \end{bmatrix}$$

2つ以上の場合にも適用は可能である。なお、添字の順番などで形式が違うこともあるので注意。

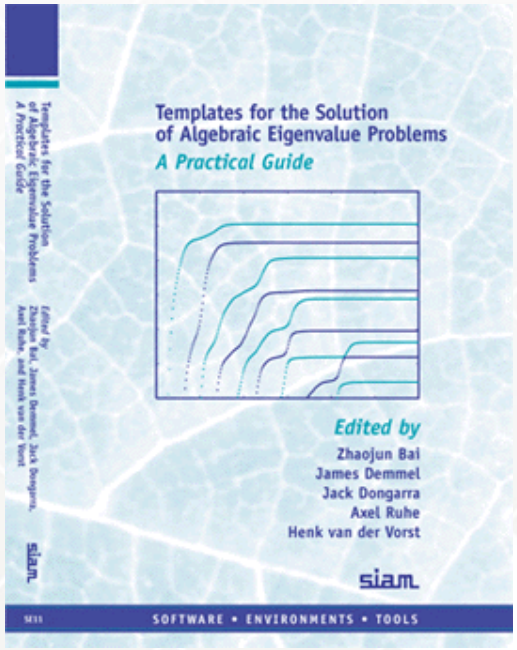
この変形で、殆どの演算が行列と行列の積に置き換えられる。

疎行列向けの反復解法

反復解法の多くは

Templates for the Solution of Algebraic Eigenvalue Problems, A Practical Guide, SIAM

を参考にしています



べき乗法から

べき乗法

- 疎行列では、記憶領域の問題やフォーマット変形操作が煩雑といった理由から、行列の変形操作を行わない。
- 主に、行列とベクトルもしくはベクトル同士の積、近似もしくは写像し縮小した行列での操作が中心となる。

密行列でも扱ったが、べき乗法が計算の基本となる。

/* べき乗法 */

$x^{(0)}$: 正規化された初期ベクトル

do

$$v = Ax^{(k)}$$

$$\lambda^{(k)} = (x^{(k)}, v)$$

$$x^{(k+1)} = \frac{v}{\|v\|}$$

$$k \leftarrow k + 1$$

while $\left| \|v\| - |\lambda^{(k)}| \right| > \varepsilon$

べき乗法

- 疎行列の場合、絶対値最大固有１つという計算はあまり行われず、最大値から数個、最小値から数個という場合が多い。
- 複数ベクトルを用いて、べき乗法の原理から

$$
\begin{aligned}
&[X^{(0)}, R] = \mathrm{qr}(X) \\
&\textbf{do} \\
&\quad V = AX^{(k)} \\
&\quad H = X^{(k)*}V \\
&\quad \textbf{if } \|V - X^{(k)}H\| < \varepsilon \textbf{ stop} \\
&\quad [X^{(k+1)}, R] = \mathrm{qr}(V) \\
&\quad k \leftarrow k + 1 \\
&\textbf{end do}
\end{aligned}
$$

ランチョス法の前に

- 先のべき乗法で　$H = X^{(k)*} A X^{(k)}$　が登場したが、これは直交基底Xによって張られる部分空間の射影問題を扱っているといえる。
- 今Aの近似固有解 $\in K$ に対して、その残差をその元となる部分空間 K に直交するように決める

$$(A\tilde{x} - \tilde{\lambda}\tilde{x}) \perp K$$

これは以下と同値です。

$$v^*(A\tilde{x} - \tilde{\lambda}\tilde{x}) = 0,\ \forall v \in K$$

また，K が直交基底 $V = \{v_1, v_2 \cdots, v_m\}$ で張られるのであれば、

$\tilde{x} = V\tilde{y}$　となっており（ V　の線形結合）

$$V^*(AV\tilde{y} - \tilde{\lambda}V\tilde{y}) = 0 \Rightarrow H_m\tilde{y} = \tilde{\lambda}\tilde{y}$$

Rayleigh/Ritz

- この様な、固有空間に対してガレルキン近似を入れて計算する方法を、Rayleigh/Ritzの手法と呼びます。 H_m に対する射影が本質部分といえます。
- 実際には、空間を張るm本の基底ベクトルから、射影した H_m の所望するk(<m)個の近似固有値を取得する。
- 次に、それに対する H_m の固有ベクトルにVを乗じて得られる近似固有ベクトルを得る。

- このとき、上記の近似固有値をRitz値, 対応する近似固有ベクトルをRitzベクトルと呼ぶ。

ランチョス法

- **先の解法には, 固有ベクトルを近似する空間の張り方に自由度があった。ここで、クリロフ部分空間法によって生成される基底を使用してみる。**

$$K^j(A, v) = \mathrm{span}\{v, Av, A^2v, A^{j-1}v\}$$

$$u_2 = Av_1$$
$$\alpha_1 = v_1^t u_2$$
$$u_2 = u_2 - \alpha_1 u_1$$
$$\beta_1 = \|u_2\|$$
$$v_2 = u_2/\beta_1$$

といった計算により, Vが逐次生成されていく。 Aの対称性を考慮すると

$$AV_j = V_j T_j + re_j^*, \quad V_j^* r = 0$$

Tはαが対角、βが副対角に並ぶ三重対角行列である。rは反復の打ち切りで生じる項

ランチョス法（アルゴリズム）

$$
\begin{aligned}
& r = v : \text{ initial vector} \\
& \beta_0 = \|r\| \\
& \textbf{do} \quad j = 1, 2, \cdots, \\
& \quad v_j = r / \beta_{j-1} \\
& \quad r = A v_j \\
& \quad r = r - v_{j-1} \beta_{j-1} \\
& \quad \alpha_j = v_j^* r \\
& \quad r = r - v_j \alpha_j \\
& \quad \beta_j = \|r\| \\
& \quad \text{Compute } (S, \Theta) = \text{eig}(T) \\
& \textbf{end do} \\
& X = V_j S
\end{aligned}
$$

ランチョス法

- 先の枠組みに当てはめれば，Tの固有値固有ベクトルがRitz値，Ritzベクトルになっている。

$$V_j^* A V_j = T_j + V_j^* r e_j^* = T_j$$

- Ritz対によるAに対する残差を見てみると

$$T_j s_i^{(j)} = s_i^{(j)} \theta_i^{(j)} \Rightarrow x_i^{(j)} = V_j s_i^{(j)}$$

$$r_i^{(j)} = A x_i^{(j)} - \theta_i^{(j)} x_i^{(j)} = A V_j s_i^{(j)} - \theta_i^{(j)} V_j s_i^{(j)} = A V_j s_i^{(j)} - V_j T_j s_i^{(j)}$$

$$= (A V_j - V_j T_j) s_i^{(j)} = r e_j^* s_i^{(j)} = \beta_j v_{j+1} (s_i^{(j)})_j$$

$$\| r_i^{(j)} \| \leq |\beta_j|$$

- つまり，反復を打ち切る際にでる v_{j+1} のノルム係数 β_j が十分に小さいかに、Ritzベクトルの精度はよっていることになる

ランチョス法（アルゴリズム）

$$
\begin{aligned}
&r = v : \text{ initial vector} \\
&\beta_0 = \|r\| \\
&\textbf{do} \quad j = 1, 2, \cdots, \\
&\quad v_j = r/\beta_{j-1} \\
&\quad r = Av_j \\
&\quad r = r - v_{j-1}\beta_{j-1} \\
&\quad \alpha_j = v_j^* r \\
&\quad r = r - v_j \alpha \\
&\quad \beta_j = \|r\| \\
&\quad \textbf{If } |\beta_j| < \varepsilon \textbf{ STOP}. \\
&\textbf{end do} \\
&\text{Compute } (S, \Theta) = \text{eig}(T),\ X = V_j S
\end{aligned}
$$

アーノルディ法

- Aが対称行列（もしくはエルミート）であった条件を緩和して、非対称行列にする。基本的な枠組みは同様であり

$$K^j(A, v) = \mathrm{span}\{v, Av, A^2v, A^{j-1}v\}$$

- の形で固有ベクトルを形成する部分空間を作成していき、適当な精度が出たところで打ち切る。

- ただし、途中計算で生成される射影行列は、ヘッセンベルグ行列となる。

アーノルディ法（アルゴリズム）

$v_1 = v/\|v\|$: initial vector

do $j = 1, 2, \cdots, m$

$w = Av_j$

do $i = 1, 2, \cdots j$

$h_{ij} = w^* v_i$

$w = w - h_{ij} v_i$

enddo

$h_{j+1,j} = \|w\|$

if $h_{j+1,j} < \varepsilon$ **STOP**

$v_{j+1} = w/h_{j+1,j}$

end do

Compute $[S, \Theta] = \mathrm{eig}(H)$, Then $X = V_j S$.

リスタート

- ランチョス法、アーノルディ法ともに 反復により部分空間を張る基底を作成し続けても、所望の固有ベクトルを十分近似できないことがある。

- その様な場合に、適切な部分空間基底を初期対として再度反復を開始するリスタートの技術が存在する。
- Thick Restart Lanzcosなどいくつかの手法があるが、今回は名前だけを紹介するのみとする。

（リスタートアルゴリズム）

Thick Restart, Implicit Restart, Explicit Restartなどなど

Jacobi-Davidoson

- **Krylov部分空間法とは異なる原理で、部分空間生成基底を生成する方法としてJD法が存在する。**

$$AV_m s - \theta V_m s \perp \{v_1, \cdots, v_m\} \Rightarrow V_m^* A V_m s - \theta = 0$$

$A(u_j^{(m)} + t) = \lambda(u_j^{(m)} + t), t \perp u_j^{(m)}$ Ritz対から生じる条件を課して基底を作る

実際には以下の、方程式を解き基底 t を作る

$$(I - u_j^{(m)} u_j^{(m)*})(A - \lambda I)(I - u_j^{(m)} u_j^{(m)*})t = -(A - \theta_j^{(m)} I)u_j^{(m)} = r$$

ここで、$(I - u_j^{(m)} u_j^{(m)*})$ は u_j の成分を排除するフィルタの役割をする。

JD法（アルゴリズム）

$t = v_0$: initial vector

do $m = 1, 2, \cdots,$

  do $i = 1, 2, \cdots m-1$

    $t = t - (v_i^* t) v_i$

  enddo

  $v_m = t/\|t\|, w_m = A v_m$

  do $i = 1, 2, \cdots, m$

    $M_{i,m} = v_i^* w_m$

  enddo

  Compute the largest eigenpair (θ, s) of M.

  $u = Vs, z = Ws, r = z - \theta s$

  If $\|r\| < \varepsilon$ **STOP**

  Solve a $t \perp u$ from $(I - uu^*)(A - \theta I)(I - uu^*)t = -r$

end do

その他の解法

- **対称行列の固有値問題はRayleigh商の極小問題**

$$\frac{x^* A x}{\|x\|^2}$$

を局所最小化することでも計算可能 → 非線形関数の最小化問題

LOBPCGやCGR, MINRESなどの手法もあるが、今回は省略します。

数値計算ライブラリ

数値計算ライブラリとは

- **高度なプログラミングを保証するための構成要素の一つです。**
- **非常に複雑な数学的機能、アルゴリズム、スキーム、データ処理などを完備なインターフェイスで提供します。**
  - 例：連立一次方程式求解、高速フーリエ変換または離散フーリエ変換（FFT or DFT)、固有値計算、特異値計算、最小化、統計処理など
- **参照コードとして役立つことがあります**
  - 良い（悪い）プログラミングの例かもしれません

```
*
*        Form  C := alpha*A*B + beta*C.
*
         DO 90 j = 1,n
            IF (beta.EQ.zero) THEN
               DO 50 i = 1,m
                  c(i,j) = zero
   50          CONTINUE
            ELSE IF (beta.NE.one) THEN
               DO 60 i = 1,m
                  c(i,j) = beta*c(i,j)
   60          CONTINUE
            END IF
            DO 80 l = 1,k
               temp = alpha*b(l,j)
               DO 70 i = 1,m
                  c(i,j) = c(i,j) + temp*a(i,l)
   70          CONTINUE
   80       CONTINUE
   90    CONTINUE
```

http://www.netlib.org/lapack/explore-html/d1/d54/group__double__blas__level3_gaeda3cbd99c8fb834a60a6412878226e1.html

数値計算ライブラリとは

- **高度なプログラミングを保証するための構成要素の一つです。**
- **非常に複雑な数学的機能、アルゴリズム、スキーム、データ処理などを完備なインターフェイスで提供します。**
  - 例：連立一次方程式求解、高速フーリエ変換または離散フーリエ変換（FFT or DFT)、固有値計算、特異値計算、最小化、統計処理など
- **参照コードとして役立つことがあります**
  - 良い（悪い）プログラミングの例かもしれません

- **多くの場合、専門家により作成であり、あなたが作ったものよりも優れた計算速度と計算精度を提供してくれます**
  - 市販のライブラリ：より速く、より正確だが高価
  - オープンソースライブラリ：高速で自由な利用が認められてるライセンス形態（商用ライブラリよりも高速な場合もある)

アプリケーションコードを実行する前にどれが皆さんに適しているかをチェックする必要があります。

たとえば

- **行列積がボトルネックになっているコードを高速化したい場合は、適切なBLAS (Basic Linear Algebra Subprograms) ライブラリのDGEMMを使用してください (またはリンクしてください)。**

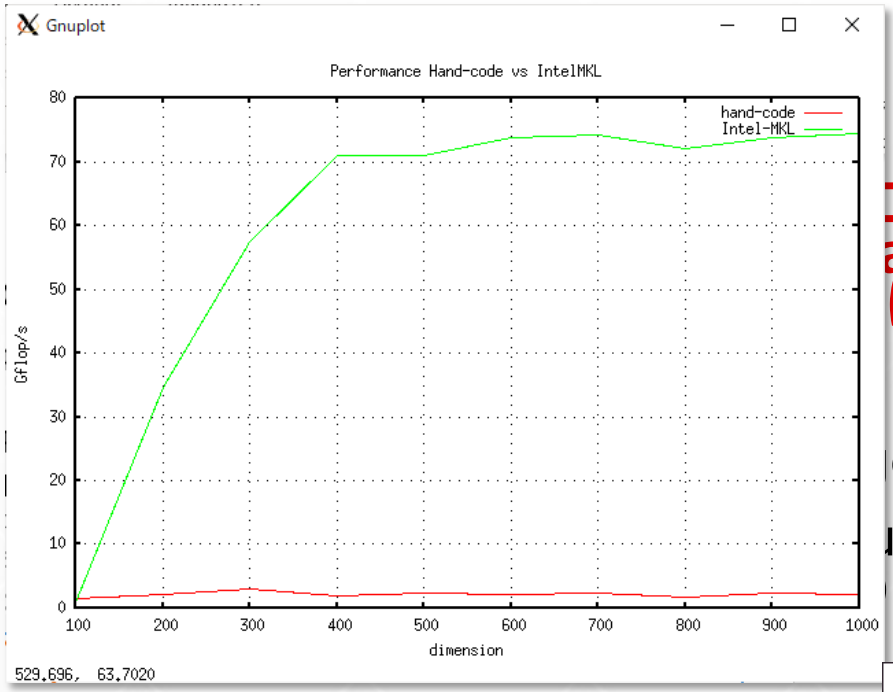
- Standard API for linear algebra kernels (case of (C)BLAS).
  - GEMM：Matrix-matrix multiplication ($C := \alpha AB + \beta C$)
  - AXPY: linear combination of 2 Vectors ( )
  - NRM2: Norm of a vector, etc. ( )

```
for(i=0; i<N; i++)
 for(j=0; j<N; j++) {
  t = 0.0;
  for(k=0; k<N; k++)
     t += a[i][k]*b[k][j];
  c[i][j] = t;
 }
```

→

```
cblas_dgemm( CblasRowMajor,
             CblasNoTrans, CblasNoTrans,
             N, N, N,
             1.0, a, N, b, N, 0.0, c, N);
```

- NVIDIA CUBLAS, AMD clMATH, MAGMABLAS(@UTK), KBLAS(@KAUST), ASPEN.K2(@RIKEN)：for GPGPU



ードを高速化したい場合は、適切な
a Subprograms) ライブラリの
はリンクしてください)。

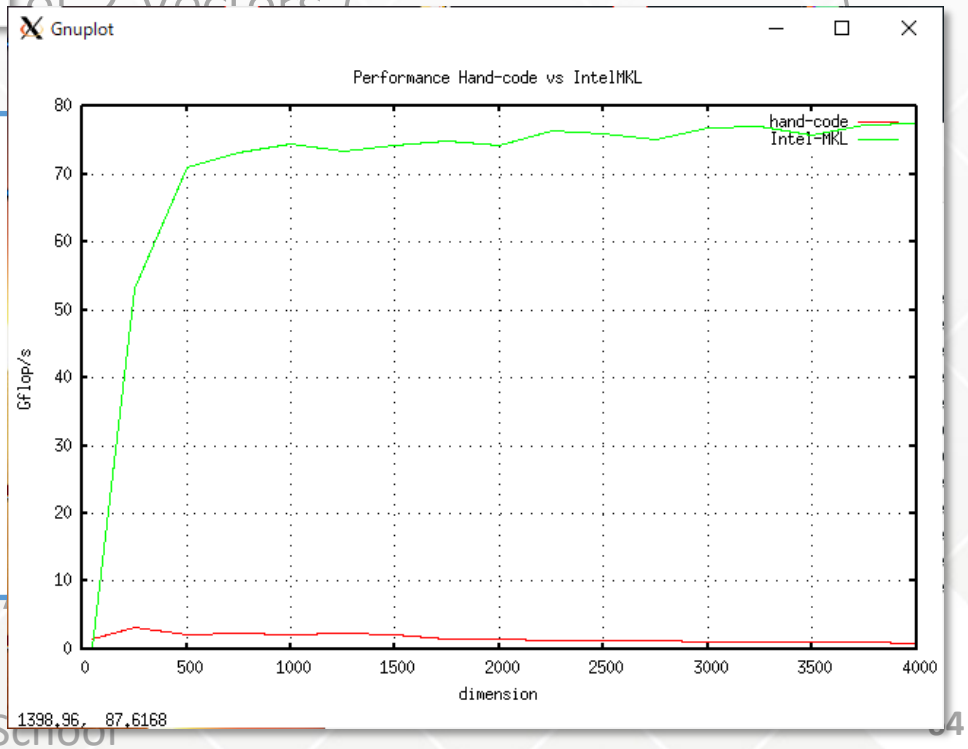
ebra kernels (case of (C)BLAS).

ltiplication

of 2 Vectors ( )

- NRM2: Norm of a vector,

```
for(i=0; i<N; i++)
 for(j=0; j<N; j++) {
  t = 0.0;
  for(k=0; k<N; k++)
     t += a[i][k]*b[k][j];
  c[i][j] = t;
 }
```



- NVIDIA CUBLAS, AMD clAmdBlas, MAGMA, KBLAS(@KAUST), ASPEN.

たとえば

- **行列積がボトルネックになっているコードを高速化したい場合は、適切なBLAS (Basic Linear Algebra Subprograms) ライブラリのDGEMMを使用してください (またはリンクしてください)。**

  - Standard API for linear algebra kernels (case of (C)BLAS).
    - GEMM : Matrix-matrix multiplication ($C := \alpha AB + \beta C$)
    - AXPY: linear combination of 2 Vectors ($y := \alpha x + y$)
    - NRM2: Norm of a vector, etc. ($a = \|x\|_2$)

    Reference codes are available from netlib@UTK.

    http://www.netlib.org/BLAS/

  - 商用: Intel MKL, AMD ACML (free)
  - オープンソース: ATLAS(@UTK), GotoBLAS(@TACC), OpenBLAS for 汎用マルチコアプロセッサ向け
  - nVIDIA CUBLAS, AMD clMATH, MAGMABLAS(@UTK), KBLAS(@KAUST), ASPEN.K2(@RIKEN) : GPGPU向け

もっと紹介します

Computer simulations create the future

提案：より複雑な問題の場合は、以下を使用してください。

- LAPACK (http://www.netlib.org/lapack/)
- ScaLAPACK (http://www.netlib.org/scalapack/) — *Dense, General*
- Elemental (http://libelemental.org/)
- EigenExa (http://www.aics.riken.jp/labs/lpnctrt/EigenExa_e.html) — *Dense Eigenvalue*
- ELPA (http://elpa.rzg.mpg.de/)
- PETSc (http://www.mcs.anl.gov/petsc/) — *Sparse, General*
- Trillions (https://trilinos.org/)
- ARPACK (http://www.caam.rice.edu/software/ARPACK/) — *Sparse, Eigenvalue*
- FFTW (http://www.fftw.org/)
- FFTE (http://www.ffte.jp/) — *FFT*
- 2decomp&FFT (http://www.2decomp.org/)
- MT, MTGP, dSFMT (http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/index.html) — *Random number*
- GMP big number librart(https://gmplib.org/)
- QD pack, MPACK, and so on — *Multi-precision number*

OBCXでの実演習

OBCXでの実行時の注意事項（再掲）

Computer simulations create the future

- 環境変数LANGを適切に設定（抜けていることがあるので）

```
% export LANG=C
```

- 3日目午前は, PETScというライブラリを利用しますので、ログインノード上で以下を実行して、モジュールを有効にしてください。

```
% module load impi petsc phdf5
```

- makeコマンド実行時に ‘depend_c.inc:1: *** multiple target patterns. Stop.’, といったメッセージがあった場合、上記文モジュールが正しく読み込まれていない可能性があります。”module list”とタイプしてモジュールとそのバージョンを確認して下さい。

```
Currently Loaded Modulefiles:
  1) impi/2019.5.281    3) phdf5/1.10.5
  2) intel/2019.5.281   4) petsc/3.11.2
```

（再掲）ほとんどのソースファイルとヒントは、

Computer simulations create the future

/work/gt65/t65021/share/

PETSc/ → PETSc sample code, a job-script, and Makefile
SLEPc/ → SLEPc sample code, a job-script, and Makefile

- **Please ‘cd’ your work directory, then ‘cp -R’ the directory (‘%’ is a prompt, and do not type). You will see**

```
% cd /work/gt65/tXXXXX
% cp -R /work/gt65/t65021/share ./third_day
% cd third_day
% ls
PETSc  SLEPc
```

PETSc(SLEPC)

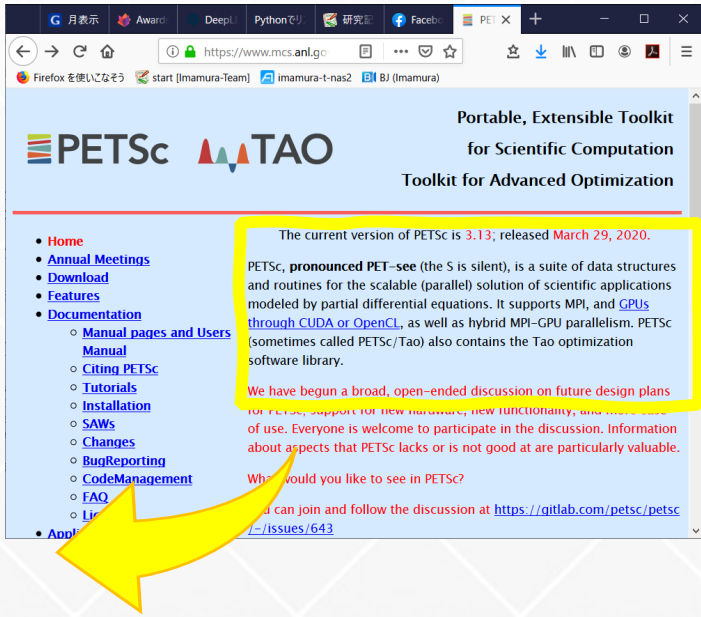
Official site on PETSc and SLEPC

Hands-on exercises

PETSc/TAO

- **Developed by Argonne National Lab. USA**
  - Portable, Extensible Toolkit for Scientific Computation
  - Toolkit for Advanced Optimization

https://www.mcs.anl.gov/petsc/



The current version of PETSc is 3.13; released March 29, 2020.

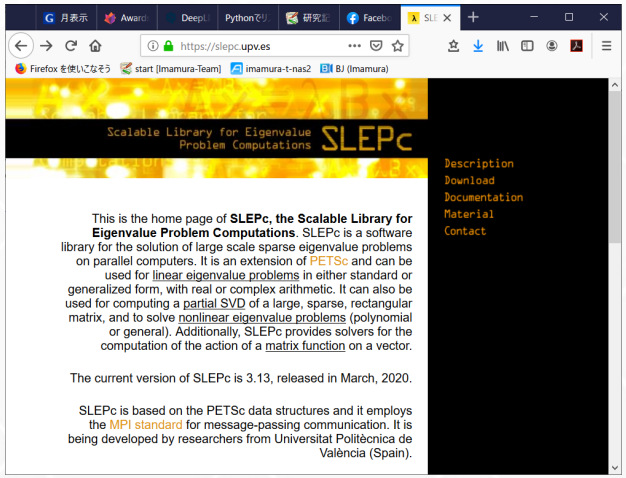
PETSc, pronounced PET-see (the S is silent), is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It supports MPI, and GPUs through CUDA or OpenCL, as well as hybrid MPI-GPU parallelism. PETSc (sometimes called PETSc/Tao) also contains the Tao optimization software library.

(cf. PETSc/TAO homepage)

SLEPc

- **Significant plugin modules of PETSc for EVP**
  - developed by Universitat Politecnica de Valencia, Spain

https://slepc.upv.es/



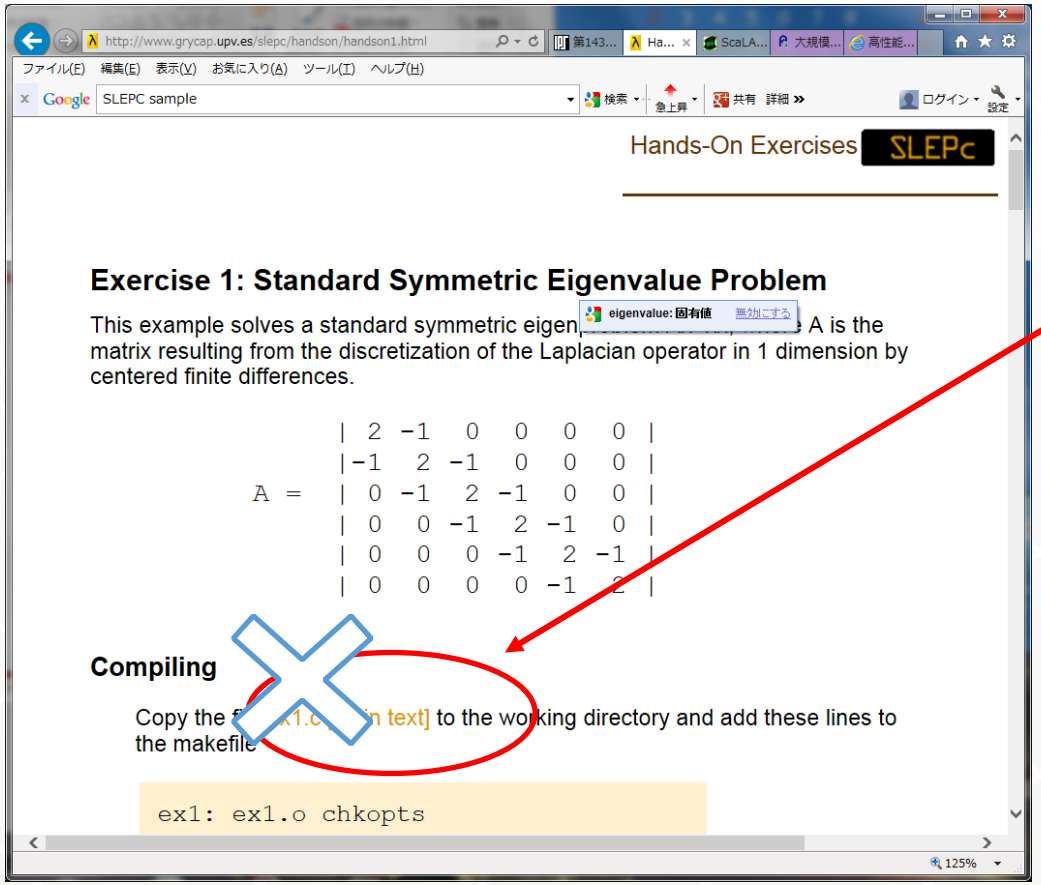
SLEPc is a software library for the solution of large scale sparse eigenvalue problems on parallel computers. It is an extension of PETSc and can be used for linear eigenvalue problems in either standard or generalized form, with real or complex arithmetic. It can also be used for computing a partial SVD of a large, sparse, rectangular matrix, and to solve nonlinear eigenvalue problems (polynomial or general). Additionally, SLEPc provides solvers for the computation of the action of a matrix function on a vector.

The current version of SLEPc is 3.13, released in March, 2020.

サンプルコードへのアクセス

Computer simulations create the future

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



リンク先URL が無効なので.
以下のディレクトリをコピーしてください

Other tutorial examples are on
/work/gt65/t65021/share/SLEPc
/work/gt65/t65021/share/PETSc

コンパイルとリンク

Makefile(一部のみ掲載)

```
ARCH=
SLEPC_DIR=/work/gt65/t65021/slepc/

CC=mpiicc
FC=mpiifort
BLAS=

CCFLAGS_PETSC=-std=c11 -qopenmp -Wall -O3 -g -I$(PETSC_DIR)/include
LDFLAGS_PETSC=-std=c11 -qopenmp -L$(PETSC_DIR)/lib -lpetsc -lhdf5_hl -
L$(MKLROOT)/lib/intel64 -lmkl_rt
CCFLAGS_SLEPC=-std=c11 -qopenmp -Wall -O3 -g -I$(SLEPC_DIR)/include
LDFLAGS_SLEPC=-std=c11 -qopenmp -L$(PETSC_DIR)/lib -lpetsc -L$(SLEPC_DIR)/lib -
lslepc -lhdf5_hl -L$(MKLROOT)/lib/intel64 -lmkl_rt

all: ex1
ex1: ex1.o
        $(CC) $(CCFLAGS_SLEPC) -o $@ $< $(LDFLAGS_SLEPC)
ex1.o: ex1.c
        $(CC) $(CCFLAGS_SLEPC) -c $<
clean:
        ¥rm *.o eps_exec
```

make コマンドを使います！！

→ → →

ex1が生成されます！

プログラム実行（ジョブ投入）

```
#!/bin/bash
#PJM -L rscgrp=lecture7
#PJM -L node=1
#PJM --mpi proc=4
#PJM -L elapse=0:10:00
#PJM -g gt65
#PJM -N slepc
#PJM -j

module load intel impi petsc phdf5

export SLEPC_DIR=/work/gt65/t65021/slepc/
export LD_LIBRARY_PATH=$SLEPC_DIR/lib:$LD_LIBRARY_PATH

mpiexec.hydra -n ${PJM_MPI_PROC} ./ex1 -n 540 -m 180 -eps_type gd -
eps_nev 4 -eps_monitor -eps_view
```

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 version

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 version

PETSc/SLEPCで遊んでみます

- チュートリアルのページを参考にして，

```
$ ./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
```

サンプルコードを学習 (Pseudo code)


```
SlepcInitialize( PETSC_NULL_CHARACTER, ierr )

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


(…. Calculation of matrix elements and others …..)


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 )
```

Modern style

1. Initialization
2. Create a Handle for data
3. Setup parameters

4. Create a Handle for solvers
5. Setup parameters

6. Kick the solver

7. Retrieve the results

8. Destroy handles
9. Finalize

行列データはどうやって設定するの? (C)

- **PETScは内部データフォーマットとインターフェースデータを柔軟に処理します。PETScのデータ管理機構により、ユーザはメモリ上の実際の状態を通常見ることができません。行列Aはハンドラ変数で処理され、行列要素はクエリAPIなどを介してアクセスされます。**

```
// Simple matrix format
Mat         A;
EPS         eps;
EPSType     tname;
PetscReal   tol, error, *values;

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

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

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

matrix ハンドラ作成

行列データはどうやって設定するの? (C)

- **PETScは内部データフォーマットとインターフェースデータを柔軟に処理します。PETScのデータ管理機構により、ユーザはメモリ上の実際の状態を通常見ることができません。行列Aはハンドラ変数で処理され、行列要素はクエリAPIなどを介してアクセスされます。**

```
// Simple matrix format
Mat         A;
EPS         eps;
EPSType     tname;
PetscReal   tol, error, *values;

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

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

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

行列データはどうやって設定するの? (C)

- **PETScは内部データフォーマットとインターフェースデータを柔軟に処理します。PETScのデータ管理機構により、ユーザはメモリ上の実際の状態を通常見ることができません。行列Aはハンドラ変数で処理され、行列要素はクエリAPIなどを介してアクセスされます。**

```
// Simple matrix format
Mat         A;
EPS         eps;
EPSType     tname;
PetscReal   tol, error, *values;

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

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

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

mxn ブロック行列valuesから行列Aの値がセットされます

行列データはどうやって設定するの? (C)

- **PETScは内部データフォーマットとインターフェースデータを柔軟に処理します。PETScのデータ管理機構により、ユーザはメモリ上の実際の状態を通常見ることができません。行列Aはハンドラ変数で処理され、行列要素はクエリAPIなどを介してアクセスされます。**

```
// Simple matrix format
Mat         A;
EPS         eps;
EPSType     tname;
PetscReal   tol, error, *values;

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

MatGetOwnershipRange(A, &Istart, &Iend)
MatSetValues(  A, m, idxm, n, idxn, value, INSERT_VALUES|ADD_VALUES )

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

分散状態のデータをアセンブルして、行列データを適切な状態で分散します

行列データはどうやって設定するの?(in fortran90)

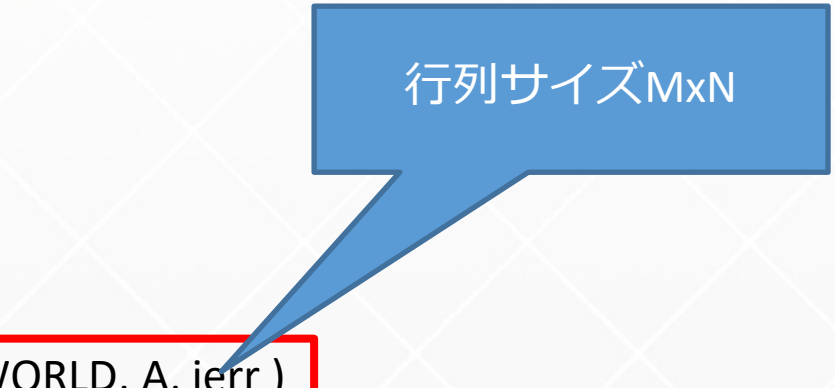
- **PETScは内部データフォーマットとインターフェースデータを柔軟に処理します。PETScのデータ管理機構により、ユーザはメモリ上の実際の状態を通常見ることができません。行列Aはハンドラ変数で処理され、行列要素はクエリAPIなどを介してアクセスされます。**

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

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

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

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



行列データはどうやって設定するの?(in fortran90)

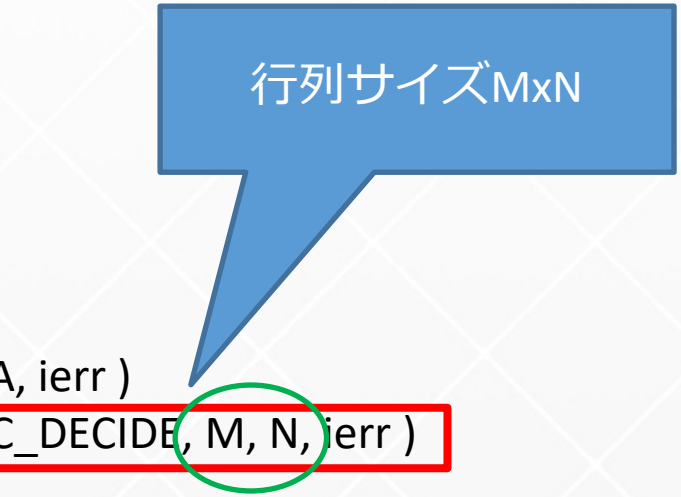
- **PETScは内部データフォーマットとインターフェースデータを柔軟に処理します。PETScのデータ管理機構により、ユーザはメモリ上の実際の状態を通常見ることができません。行列Aはハンドラ変数で処理され、行列要素はクエリAPIなどを介してアクセスされます。**

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

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

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

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



行列データはどうやって設定するの?(in fortran90)

- **PETScは内部データフォーマットとインターフェースデータを柔軟に処理します。PETScのデータ管理機構により、ユーザはメモリ上の実際の状態を通常見ることができません。行列Aはハンドラ変数で処理され、行列要素はクエリAPIなどを介してアクセスされます。**

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

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

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

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

mxn ブロック行列
valuesから行列Aの
値がセットされます

行列データはどうやって設定するの?(in fortran90)

- **PETScは内部データフォーマットとインターフェースデータを柔軟に処理します。PETScのデータ管理機構により、ユーザはメモリ上の実際の状態を通常見ることができません。行列Aはハンドラ変数で処理され、行列要素はクエリAPIなどを介してアクセスされます。**

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

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

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

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

分散状態のデータをアセンブルして、行列データを適切な状態で分散します

Hands-on time

/work/gt65/t65021/share/ から必要なファイルをコピーしてください.

PETSc/ → PETSc sample code, a job-script, and Makefile

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++) {
  PetscInt ib=i/n; PetscInt 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

- **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

結果を可視化してみよう

- **固有ベクトルをファイルに書き出してみる**
  - 下記操作を、ex1.cの130行目の前あたりに追加する

(a.awk は書式をgnuplot方式にするための外部スクリプト：下部枠内)

```
{
    char filename[256];
    sprintf(filename,"eigenvector%03d.txt",i);
    PetscViewer viewer;
    PetscViewerASCIIOpen(PETSC_COMM_WORLD, filename, &viewer);
    PetscViewerPushFormat(viewer, PETSC_VIEWER_ASCII_INDEX);
    VecView(xr,viewer);
    PetscViewerPopFormat(viewer);
    PetscViewerDestroy(&viewer);
    if ( Istart==0 ) {
      char command[1024];
      sprintf(command,"awk -v n=%d -v out=v%03d.data -f a.awk %s",
          n, i, filename);
      system(command);
    }
  }
```

```
BEGIN{ if(n==0)n=10; if(out=="")out="ev000.data"; }
/^[0-9]*:/{
    gsub(/:/,""); N=$1;
    row = N % n; col = N / n;
    printf("%d %d %lf%c¥n", row, col, $2, (row==n-1 ? "¥n" : " ")) > out;
}
```

結果を可視化してみよう

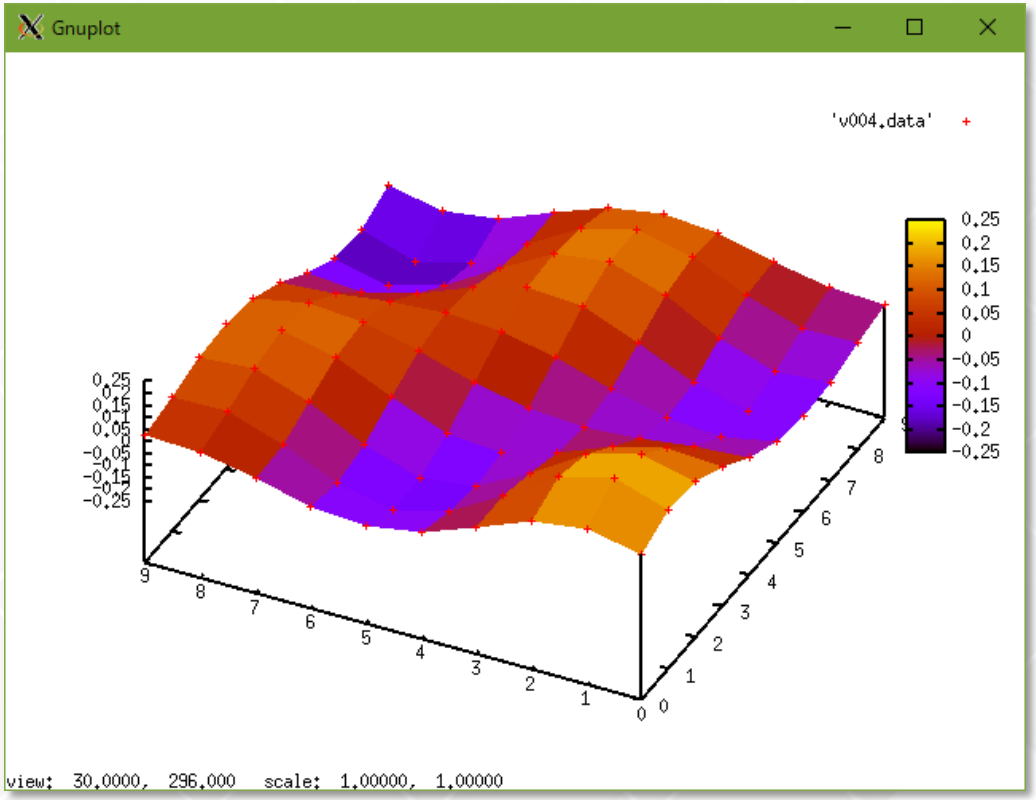
Computer simulations create the future

実行ジョブスクリプト: n=10として、0付近の固有値を5個計算

```
# mpirun dplace ./ex1-2d -n 10 -eps_nev 5 -eps_target 0
```

（右図）Gnuplotでsplot

```
# set pm3d
# splot ‘v004.data’
```



まとめ

- **代表的な並列数値計算ライブラリ**
  - ScaLAPACK
  - EigenExa
  - PETSc

  他にも多数存在。逐次版でLAPACK, BLASなど

- **PETSc and SLEPc on OBCX**
  - 疎行列だけでなくで手軽に並列計算可能
  - アルゴリズムも選択できる
  - 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