計算機実験 (L3) — ライブラリの利用・反復法・対角化

藤堂眞治
wistaria@phys.s.u-tokyo.ac.jp

2017/05/31

一次元配列

- (静的) 一次元配列 (ハンドブック 3.3.1 節)

```
double v[10];
v[0] = 1.0;
v[1] = 2.0;
...
```

要素数はコンパイル時にすでに決まっている定数でなければならない

- (動的) 一次元配列 (ハンドブック 3.11 節)

```
double *v; /* ポインタ */
v = (double*)malloc((size_t)(10 * sizeof(double));
...
free(v); /* 確保した領域を開放 */
```

実行時に要素数を指定可能

ポインタと一次元配列

- 一次元配列を表す変数は、(実は) 最初の要素を指すポインタ (ハンドブック 3.5.3 節)
  - `v` と `&v[0]` は等価
  - `(v+2)` と `&v[2]` は等価
  - `*v` と `v[0]` は等価
  - `*(v+2)` と `v[2]` は等価
  - `(v+2)[3]` は?
- C 言語では配列の添字は 0 から始まることに注意
- `double v[10];` と宣言した場合、`v[0]` 〜 `v[9]` の 10 個の要素を持つ配列が作られる。`v[10]` は存在しない。値を代入したり参照しようとするとエラーとなる

二次元配列

- C 言語では、二次元配列は一次元配列の先頭をさす (ポインタ) の配列として表される (と理解しておけば良い)
- `m[i]` は、要素 `m[i][0]` を指すポインタ
  - `m` と `&m[0]` は等価 (`&m[0][0]` ではない)
  - `m[0]` と `&m[0][0]` は等価
  - `m[2]` と `&m[2][0]` は等価
  - `(m+2)` と `&m[2]` は等価
  - `(*(m+2))[3]` と `*(*(m+2)+3)` と `m[2][3]` は等価
  - `*(m+2)[3]` と `*((m+2)[3])` と `*(m[5])` と `m[5][0]` は等価
  - `[]` は`*`よりも強い

動的二次元配列の確保

- 各行を表す配列とそれぞれの先頭アドレスを保持する配列の二種類が必要

```
double **a;
m = 10;
n = 10;
a = (double**)malloc((size_t)(m * sizeof(double*));
for (int i = 0; i < m; ++i)
  a[i] = (double*)malloc((size_t)(n * sizeof(double));
```

- 各行を保持する配列が、メモリ上で連続に確保される保証はない
- LAPACK を使うときに問題となる

動的二次元配列の確保

- 二次元配列の要素を格納する長い配列を用意する

```
double **a;
m = 10;
n = 10;
a = (double**)malloc((size_t)(m * sizeof(double*));
a[0] = (double*)malloc((size_t)(m*n * sizeof(double));
for (int i = 1; i < m; ++i)
  a[i] = a[i-1] + n;
```

- 開放は逆の順序で行う

```
free(a[0]);
free(a);
```

動的二次元配列の確保

- 二次元配列の確保 (alloc_dmatrix)、開放 (free_dmatrix)、出力 (print_dmatrix)、読み込み (read_dmatrix) を行うためのユーティリティ関数を準備
- ソースコード: exercise/linear_system/matrix_util.h
- 使用例

```
#include "matrix_util.h"
...
double **mat;
mat = alloc_dmatrix(m, n);
...
free_dmatrix(mat);
```

LAPACK (Linear Algebra PACKage)

- 線形計算のための高品質な数値計算ライブラリ
  - http://www.netlib.org/lapack
  - 線形方程式、固有値問題、特異値問題、線形最小二乗問題など
  - (FFT 高速フーリエ変換は入っていない)
  - LAPACK 自体は Fortran で書かれている
- ほぼ全ての PC、ワークステーション、スーパーコンピュータで利用可 (インストール済)
- Netlib でソースが公開されているリファレンス実装は遅いが、それぞれのベンダー (Intel、Fujitsu、etc) による最適化された LAPACK が用意されている場合が多い (MKL、SSL2、etc)
- LAPACK を使うことにより、高速で信頼性が高く、ポータブルなコードを書くことが可能になる

LAPACK による連立一次方程式の求解

- LU 分解を行うサブルーチン dgetrf
  http://www.netlib.org/lapack/explore-html/d3/d6a/dgetrf_8f.html
- Fortran による関数宣言

```
subroutine dgetrf(integer M, integer N,
          double precision, dimension(lda, *) A,
          integer LDA, integer, dimension(*) IPIV,
          integer INFO)
```

- A: 左辺の行列、N,M: 次元、IPIV: 選択されたピボット行のリスト、lda: M と同じで良い

LAPACK による連立一次方程式の求解

- C 言語から呼び出すための関数宣言を作成 (ハンドブック 3.6.4 節)

```
void dgetrf_(int *M, int *N, double *A,
             int *LDA, int*IPIV, int *INFO);
```

関数名は全て小文字。関数名の最後に _ (下線) を付ける

- LU 分解の例

```
M = 10;
N = 10;
LDA = 10;
dgetrf_(&M, &N, &A[0][0], &LDA, &IPIV[0], &INFO);
```

完全なソースコード: exercise/linear_system/lu_decomp.c

C から Fortran のライブラリを呼び出す際の注意事項

- スカラーも配列も全てポインタ渡しとする
- C 言語は添字が 0 から始まる。Fortran は 1 から始まる
- C と Fortran で、二次元配列のメモリ上での並びが違う
  C は row-major: `a[0][0]`, `a[0][1]`, `a[0][2]`, $\cdots$
  Fortran は column-major: `a(1,1)`, `a(2,1)`, `a(3,1)`, $\cdots$
- C で作成した行列を Fortran に渡すと転置されてしまう
- コンパイル時には`-llapack` オプションを指定し、LAPACK ライブラリをリンクする (ハンドブック 3.1.6 節)

直接法と反復法

- 直接法: 連立方程式を有限回数 ($\sim n^3$) の手間で直接解く
- 反復法: $A\mathbf{x} = \mathbf{b}$ を、等価な $\mathbf{x} = \phi(\mathbf{x}) = M\mathbf{x} + \mathbf{c}$ の形に変形し、適当な初期値 $\mathbf{x}_0$ から出発して、$\mathbf{x}^{(k+1)} = \phi(\mathbf{x}^{(k)})$ を繰り返して解を求める
  - 欠点: 有限回数では終わらない (あらかじめ定めた収束条件が満たされるまで反復)
  - 利点: 行列ベクトル積 $M\mathbf{x}$ が計算できさえすればよい。特に M が疎行列の場合には、$M\mathbf{x}$ は非常に高速に計算できる可能性がある。メモリの点でも有利
  - 利点: 直接法に比べて、プログラムも比較的単純になる場合が多い

反復法

- 行列 A を対角行列 D、左下三角行列 E、右上三角行列 F の和に分解

$$A\mathbf{x} = (D + E + F)\mathbf{x} = \mathbf{b}$$

- ヤコビ法: 対角成分以外を右辺に移す

$$\mathbf{x}^{(k+1)} = D^{-1}(\mathbf{b} - (E + F)\mathbf{x}^{(k)}) = -D^{-1}(E + F)\mathbf{x}^{(k)} + D^{-1}\mathbf{b}$$

- ガウスザイデル法: ヤコビ法で右辺の $\mathbf{x}$ の値として、各段階ですでに得られている最新のものを使う

$$\mathbf{x}^{(k+1)} = D^{-1}(\mathbf{b} - E\mathbf{x}^{(k+1)} - F\mathbf{x}^{(k)})$$
$$\mathbf{x}^{(k+1)} = -(D + E)^{-1}F\mathbf{x}^{(k)} + (D + E)^{-1}\mathbf{b}$$

反復法

- SOR (Successive Over-Relaxation) 法: ガウスザイデル法における修正量に 1 より大きな値 (ω) を掛け、補正を加速

$$\xi^{(k+1)} = D^{-1}(\mathbf{b} - E\mathbf{x}^{(k+1)} - F\mathbf{x}^{(k)})$$
$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \omega(\xi^{(k+1)} - \mathbf{x}^{(k)})$$

  $\xi^{(k+1)}$ を消去すると

$$\begin{aligned}\mathbf{x}^{(k+1)} =& (I + \omega D^{-1}E)^{-1}\{(1-\omega)I - \omega D^{-1}F\}\mathbf{x}^{(k)} \\ &+ \omega(D + \omega E)^{-1}\mathbf{b}\end{aligned}$$

- 反復法は常に収束するとは限らない
- 行列 A が対角優位、あるいは正定値対称行列の場合には収束が保証される

時間依存しないシュレディンガー方程式

- 井戸型ポテンシャル中の一粒子問題

$$\Big[-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}+V(x)\Big]\psi(x)=E\psi(x)$$

$$V(x)=\begin{cases}0 & a\le x\le b\\ \infty & \text{otherwise}\end{cases}$$

- $\hbar^2/2m=1$、$a=0$、$b=1$ となるように変数変換して

$$\Big(\frac{d^2}{dx^2}+E\Big)\psi(x)=0 \qquad 0\le x\le 1$$

を境界条件 $\psi(0)=\psi(1)=0$ のもとで解けば良い

常微分方程式の解法の利用

- $x_i = h \times i$ $(h = 1/n)$、$x_0 = 0$、$x_n = 1$ とする
- $\psi(x_0) = 0$、$\psi(x_1) = 1$ を仮定 $(\psi'(x_0) = 1/h$ と与えたことに相当)
- $E = 0$ とおく
- 初期条件のもとで、$x = x_n$ まで積分
- $\psi(x_n)$ の符号がかわるまで、E を少しずつ増やす
- 符号が変わったら、E の区間を半分ずつに狭めていき、$\psi(x_n) = 0$ となる E (固有エネルギー) と $\psi(x)$ (波動関数) を得る

シュレディンガー方程式の行列表示

- シュレディンガー方程式

$$-\frac{d^2}{dx^2}\psi(x) = E\psi(x)$$

- 連立差分方程式を行列の形で表す $(\psi(x_0) = \psi(x_n) = 0)$

$$\begin{pmatrix} \frac{2}{h^2} & -\frac{1}{h^2} & & & \\ -\frac{1}{h^2} & \frac{2}{h^2} & -\frac{1}{h^2} & & \\ & -\frac{1}{h^2} & \frac{2}{h^2} & -\frac{1}{h^2} & \\ & & \ddots & \ddots & \\ & & & -\frac{1}{h^2} & \frac{2}{h^2} \end{pmatrix} \begin{pmatrix} \psi(x_1) \\ \psi(x_2) \\ \psi(x_3) \\ \vdots \\ \psi(x_{n-1}) \end{pmatrix} = E \begin{pmatrix} \psi(x_1) \\ \psi(x_2) \\ \psi(x_3) \\ \vdots \\ \psi(x_{n-1}) \end{pmatrix}$$

- $(n-1) \times (n-1)$ の疎行列の固有値問題
  - 固有値: 固有エネルギー
  - 固有ベクトル: 波動関数

固体物理・量子統計物理に現れる行列

- 強束縛近似 (tight-binding approx.) のもとでの第二量子化表示

$$H = -t \sum_{\langle i,j \rangle \sigma} (c^{\dagger}_{i,\sigma} c_{j,\sigma} + h.c.) + (\text{相互作用})$$

- 局所スピン模型 (ハイゼンベルグ模型)

$$H = \sum_{\langle i,j \rangle} S_i \cdot S_j$$

- 格子点の数を n とすると、ハミルトニアンはそれぞれ $4^n \times 4^n$、$2^n \times 2^n$ の (疎) 行列で表される。
- n が大きくなると、行列の次元は指数関数的に増加
- 量子多体系に共通する困難

実対称行列 (エルミート行列) の性質

- $N \times N$ 実対称行列 A $(= A^T)$ の固有値問題

$$Ax = \lambda x$$

- N 個の固有値 $(\lambda_1, \lambda_2, \cdots, \lambda_N)$ は全て実。固有ベクトル $(\xi_1, \xi_2, \cdots, \xi_N)$ は互いに正規直交するようにとることができる。行列 U を

$$U = \begin{pmatrix} \xi_1 \, \xi_2 \, \cdots \, \xi_N \end{pmatrix}$$

と定義すると、U は直交 (ユニタリ) 行列 $(U^T U = U^{-1} U = I)$

- A の固有分解 (固有値分解)

$$A = U \Lambda U^T \qquad \Lambda = \mathrm{diag}(\lambda_1, \cdots, \lambda_N)$$

行列のべき乗・指数関数

- 行列のべき乗

$$\begin{aligned} A^p &= (U\Lambda U^T)(U\Lambda U^T)\cdots(U\Lambda U^T) \\ &= U\Lambda^p U^T \qquad \Lambda^p = \mathrm{diag}(\lambda_1^p, \cdots, \lambda_N^p) \end{aligned}$$

- 行列の指数関数

$$\begin{aligned} e^{xA} &= \sum_{n=0}^{\infty} \frac{1}{n!}(xA)^n = U\Big[\sum_{n=0}^{\infty} \frac{1}{n!}(x\Lambda)^n\Big]U^T \\ &= Ue^{x\Lambda}U^T \qquad e^{x\Lambda} = \mathrm{diag}(e^{x\lambda_1}, \cdots, e^{x\lambda_N}) \end{aligned}$$

- 逆行列 $A^{-1} = U\Lambda^{-1}U^T$
- 行列式 $|A| = \prod_i \lambda_i$、対角和 (トレース) $\mathrm{tr}A = \sum_i \lambda_i$

行列の数値対角化

- 一般的に次元が 5 以上の行列の固有値は、あらかじめ定まる有限回の手続きでは求まらない
- 必ず何らかの反復法 (+収束判定) が必要となる
- 密行列向きの方法
  - Jacobi 法
  - Givens 変換・Householder 法 (三重対角化) + QR 法など
- 疎行列向きの方法
  - べき乗法
  - Lanczos 法 (三重対角化) + QR 法など
- 固有ベクトル
  - 逆反復法

基本方針

- やってはいけない方法: 特性方程式

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

  の係数を求めて、代数方程式として解く
  - 数値的に不安定 (代数方程式の解は係数の誤差に対して敏感)
  - 計算コスト大 [$\sim O(N!)$]
- スタンダードな方法: 行列を次々に直交変換して、対角行列 (あるいは三重対角行列) に近づけていく

$$A \to U_1^T A U_1 \to U_2^T (U_1^T A U_1) U_2 \to U_3^T (U_2^T (U_1^T A U_1) U_2) U_3 \to \cdots$$

- 固有値は変換された行列の固有値、固有ベクトルは変換後の行列の固有ベクトルに左から $U_1 U_2 U_3 \cdots$ を掛けたもの

Jacobi 法

- 直交行列 U_{pq} を以下のように選ぶ ($(p,p),(p,q),(q,p),(q,q)$ 成分を除くと単位行列)

$$U_{pq} = \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & \cos\theta & & & \sin\theta & & \\ & & & & 1 & & & & \\ & & & & & 1 & & & \\ & & & -\sin\theta & & & \cos\theta & & \\ & & & & & & & 1 & \\ & & & & & & & & \ddots \\ & & & & & & & & & 1 \end{pmatrix}$$

Jacobi 法による相似変換

- $B = U_{pq}^{-1} A U_{pq}$ により、A の p 行、q 行、p 列、q 列のみが変更を受ける

$$
\begin{aligned}
b_{pk} &= b_{kp} = a_{pk}\cos\theta - a_{qk}\sin\theta \qquad k \neq p, q \\
b_{qk} &= b_{kq} = a_{pk}\sin\theta + a_{qk}\cos\theta \qquad k \neq p, q \\
b_{pp} &= \frac{a_{pp}+a_{qq}}{2} + \frac{a_{pp}-a_{qq}}{2}\cos 2\theta - a_{pq}\sin 2\theta \\
b_{qq} &= \frac{a_{pp}+a_{qq}}{2} - \frac{a_{pp}-a_{qq}}{2}\cos 2\theta + a_{pq}\sin 2\theta \\
b_{pq} &= b_{qp} = \frac{a_{pp}-a_{qq}}{2}\sin 2\theta + a_{pq}\cos 2\theta
\end{aligned}
$$

- $b_{pq} = b_{qp} = 0$ とするには、θ を次のように選べば良い

$$
\tan 2\theta = -\frac{2a_{pq}}{a_{pp} - a_{qq}}
$$

Jacobi 法の収束

- 相似変換により対角和は不変に保たれるので

$$\mathrm{tr}\, A^T A = \mathrm{tr}\, B^T B \quad \Rightarrow \quad \sum_{i,j} a_{ij}^2 = \sum_{i,j} b_{ij}^2$$

- 一方、この変換で

$$b_{pp}^2 + b_{qq}^2 = b_{pp}^2 + 2b_{pq}^2 + b_{qq}^2 = a_{pp}^2 + 2a_{pq}^2 + a_{qq}^2$$

すなわち、変換により、対角成分の二乗和は増加する $\Rightarrow$ 非対角成分の二乗和は単調減少

- 全ての非対角成分が十分小さくなるまで繰り返す
- 固有値=対角成分、固有ベクトル $= U_1 U_2 U_3 \cdots$

3 重対角化

- 対角化は有限回の手続きでは行えない
- 3 重対角化であれば、$O(N^3)$ の有限回の計算で決定論的に行える
- Givens 変換: Jacobi 変換と同じ相似変換を利用
  - U_{32} で (3,1) と (1,3) を消去 $\Rightarrow$ U_{42} で (4,1) と (1,4) を消去 $\Rightarrow$ U_{52} で (5,1) と (1,5) を消去 $\Rightarrow$ $U_{62}, \cdots, U_{N,2} \Rightarrow$ $U_{43}, U_{53}, \cdots, U_{N,3} \Rightarrow \cdots \Rightarrow U_{n,n-1}$ で $(n, n-2)$ と $(n-2, n)$ を消去
  - $(4/3)N^3$ 回の乗算と $(2/3)N^3$ 回の加減算で 3 重対角化される
- Householder 変換: $U = I - 2ww^T/|w|^2$
  - $(2/3)N^3$ 回の乗算と加減算で 3 重対角化される
  - Givens 変換に比べ少し効率的なので、こちらが広く使われている

3 重対角行列の対角化

- 二分法、QR 分解、分割統治法、MRRR など様々な方法が知られている
- 固有ベクトル
  - QR 分解では 3 重対角行列の固有ベクトルも同時に求まる
  - あるいは、固有値を求めた後、逆反復法を用いて固有ベクトルを求める
- 逆反復法
  - 近似固有値を μ とするとき、行列 $(A-\mu I)^{-1}$ を考えると、固有ベクトルは A と同じ、固有値は $(\lambda-\mu)^{-1}$。
  - μ が十分に正確であれば、$(\lambda-\mu)^{-1}$ は絶対値最大の固有値。行列 $(A-\mu I)^{-1}$ を適当な初期ベクトルにかけ続けると λ に対応する固有ベクトルに収束 (c.f. べき乗法)
  - 実際には $(A-\mu I)x' = x$ という連立方程式を繰り返し解く

LAPACK の対角化ルーチン

- 様々な対角化ルーチンが準備されている
  - 倍精度実対称行列の対角化 dsyev http://www.netlib.org/lapack/explore-html/dd/d4c/dsyev_8f.html
  - Fortran による関数宣言

```
subroutine dsyev(character JOBZ, character UPLO,
  integer N, double precision, dimension(lda, *) A,
  integer LDA, double precision, dimension(*) W,
  double precision, dimension(*) WORK,
  integer LWORK, integer INFO)
```

- 他にも dsyevd、dsyevr、dsyevx などがある
  3 重対角化までは同じ。3 重対角行列の対角化が異なる
- 単精度版の ssyev、複素 (エルミート行列) 版の zheev など

複素エルミート行列の固有分解

- 固有値は実数
- これまでの方法がそのまま使える (ただし、転置 $\rightarrow$ 複素転置)
- 実対称行列用のサブルーチンを使っての対角化も可能
  - エルミート行列を実部と虚部に分ける: $A = R + iW$
  - エルミート行列の固有値問題 $(R + iW)(u + iv) = \lambda(u + iv)$ を $2N \times 2N$ の実対称行列の問題に書き換える

$$\begin{pmatrix} R & -W \\ W & R \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \lambda \begin{pmatrix} u \\ v \end{pmatrix}$$

  - 固有値は同じ固有値が 2 度づつ現れる
  - 対応する複素行列の固有ベクトルは、$u + iv$ と $-v + iu$

反復法

- 疎行列の場合、行列ベクトル積は高速に行える
- Givens 変換、Householder 変換などを行うと疎行列性が失われる
- 行列ベクトル積のみを用いる反復法が効果的
  - べき乗法
  - Lanczos 法

べき乗法 (Power Method)

- 適当なベクトル v_1 から出発する
- v_1 が最大固有ベクトル ξ_1 と直交していないとすると

$$v_1 = c_1\xi_1 + c_2\xi_2 + c_3\xi_3 + \cdots + c_N\xi_N$$

と展開できる ($c_1 \neq 0$)。この両辺に A を次々掛けて行くと

$$\begin{aligned}
v_2 = Av_1 &= c_1\lambda_1\xi_1 + c_2\lambda_2\xi_2 + c_3\lambda_3\xi_3 + \cdots + c_N\lambda_N\xi_N \\
v_3 = A^2v_1 &= c_1\lambda_1^2\xi_1 + c_2\lambda_2^2\xi_2 + c_3\lambda_3^2\xi_3 + \cdots + c_N\lambda_N^2\xi_N \\
&\vdots \\
v_{n+1} = A^nv_1 &= c_1\lambda_1^n\xi_1 + c_2\lambda_2^n\xi_2 + c_3\lambda_3^n\xi_3 + \cdots + c_N\lambda_N^n\xi_N \\
&= c_1\lambda_1^n\left[\xi_1 + \sum_{k=2}^{N}\frac{c_k}{c_1}\left(\frac{\lambda_k}{\lambda_1}\right)^n\xi_k\right] \approx c_1\lambda_1^n\xi_1
\end{aligned}$$

べき乗法の収束

- べき乗法による固有値

$$\frac{v_{n+1}^T v_{n+1}}{v_{n+1}^T v_n} = \lambda_1 + O\left(\left(\frac{\lambda_2}{\lambda_1}\right)^{2n}\right)$$

- 誤差の収束

$$\frac{v_{n+1}^T v_{n+1}}{v_{n+1}^T v_n} \approx \lambda_1 + e^{-2n \ln(\lambda_1/\lambda_2)}$$

- $1/\ln(\lambda_1/\lambda_2)$ 程度の反復が必要
- λ_1 と λ_2 が近い場合には、反復回数が非常に多くなる

第 2 固有値・第 3 固有値 · · ·

- 第 1 固有ベクトル ξ_1 の成分を行列から差し引く (減次)

$$A_1 = A - \lambda_1 \xi_1 \xi_1^T$$

この行列は、固有値 $0, \lambda_2, \lambda_3, \cdots, \lambda_N$ を持つ

- 行列 A_1 に対してべき乗法を使うと、固有値 λ_2 と固有ベクトル ξ_2 が得られる
- 第 k 固有値まで求まっている場合

$$A_k = A - \sum_{i=1}^{k} \lambda_i \xi_i \xi_i^T$$

- 実際には数値誤差のため、ベクトルの直交性は厳密ではない
- 大きい方から数個程度を求めるのが限界

Rayleigh-Ritz の方法

- $N \times N$ 行列 A について、互いに正規直交するベクトル $v_1, v_2, \cdots, v_M$ $(M < N)$ が張る部分空間の中で「最良の」固有ベクトルを求めたい

- $N \times M$ 行列
$$V = (v_1 v_2 \cdots v_M)$$
を定義すると、$V^T V = I$ が成り立つ (ただし $VV^T \neq I$)

- 部分空間内のベクトルを $w = \sum_i a_i v_i$ と表すと、$\frac{w^T A w}{w^T w}$ が極大値を取る (本当の固有ベクトルにできるだけ平行になる) 条件は、
$$\frac{\partial}{\partial a_i} \frac{w^T A w}{w^T w} \sim \sum_j H_{ij} a_j - \lambda a_i = 0$$

Rayleigh-Ritz の方法

- 行列
$$H_{ij} = v_i^T A_{ij} v_j$$
に対する固有値問題: $Ha = \lambda a$
- λ: もとの行列の近似固有値 (Ritz 値)
- Va: もとの行列の近似固有ベクトル (Ritz ベクトル)
- 最大固有値に対する良い近似固有値が欲しい場合 $\Rightarrow$
$v_1, v_2, \cdots, v_M$ を最大固有ベクトルになるべく近い (しかし、互いに直交する) ベクトルに選べば良い

Lanczos 法

- 初期 (ランダム) ベクトル v_1 に加えて

$$Av_1, Av_2, \cdots A^{M-1}v_1$$

を正規直交化して $v_1, v_2, \cdots, v_M$ を作る (Krylov 部分空間)

- 部分空間での Ritz 値を固有値の近似値とする
- $A^n v_1$ はどんどん最大固有ベクトルに近づいていくので、
$M \ll N$ でも良い近似固有値が得られると期待される
- Lanczos 法 (Arnordi 法)

Lanczos 法

- 正規化された初期 (ランダム) ベクトル v_1 から出発する
- $v_2, v_3, \cdots$ を生成する

$$
\begin{aligned}
v_2 &= (Av_1 - \alpha_1 v_1)/\beta_1 \\
v_3 &= (Av_2 - \alpha_2 v_2)/\beta_2 \\
&\vdots
\end{aligned}
$$

ここで

$$
\begin{aligned}
\alpha_i &= v_i^T A v_i \\
\beta_i &= |Av_i - \beta_{i-1} v_{i-1} - \alpha_i v_i|
\end{aligned}
$$

と選ぶ

Lanczos 法

- $v_1, v_2, v_3, \cdots, v_{M+1}$ は正規直交
- 漸化式を書き換えると

$$
\begin{aligned}
Av_1 &= \alpha_1 v_1 + \beta_1 v_2 \\
Av_2 &= \beta_1 v_1 + \alpha_2 v_2 + \beta_2 v_3 \\
Av_3 &= \beta_2 v_2 + \alpha_3 v_3 + \beta_3 v_4 \\
&\vdots \\
Av_M &= \beta_{M-1} v_{M-1} + \alpha_M v_M + \beta_M v_{M+1}
\end{aligned}
$$

Lanczos 法

- 行列で表現すると

$$A(v_1 v_2 \cdots v_M) = (v_1 v_2 \cdots v_M v_{M+1}) \begin{pmatrix} \alpha_1 & \beta_1 & & & & \\ \beta_1 & \alpha_2 & \beta_2 & & & \\ & \beta_2 & \alpha_3 & \beta_3 & & \\ & & \beta_3 & \alpha_4 & \beta_4 & \\ & & & \ddots & \ddots & \ddots \\ & & & & \beta_{M-1} & \alpha_M \\ & & & & & \beta_M \end{pmatrix}$$

両辺に左から $(v_1 v_2 \cdots v_M)^T$ をかけると

$$(v_1 v_2 \cdots v_M)^T A (v_1 v_2 \cdots v_M)$$

は 3 重対角行列となることがわかる

Lanczos 法

- 原理的には、N ステップ目で $\beta_N = 0$ となり、3 重対角化が完了する
- 実際には、数値誤差のため $v_1, v_2, v_3 \cdots$ の直交性が崩れる
  - M を大きくしすぎると、おかしな固有値が出てくる
  - 全ての固有値が欲しい場合には Householder 法を使う
- Lanczos 法では、大きな固有値に対応する固有ベクトルにできるだけ近いものから部分空間を作っていく
  - 100 万次元以上の行列の場合でも $M = 100 \sim 200$ 程度で最初の数個の固有値は精度良く求まる
- 必要な操作は、行列とベクトルの積、ベクトルの内積・スケーリング・和のみ
  - 疎行列の場合、非常に効率が良い