Lanczos Algorithm

Using Householder Reflection

Outline

- Introduction
- Review: Householder’s QR decomposition
- Similarity transformation using Householder matrices
- Eigenvalues of tri-diagonal matrices
- The algorithm
- Convergence analysis
- Time complexity analysis

Introduction (1)

- Lanczos algorithm
  - A numerical algorithm for computing eigenvalues of symmetric matrices.
- For computing eigenvalues
  1. Using Lanczos iteration to perform tri-diagonalization $T = V^T AV$,

  $\boldsymbol{T}$ is a symmetric tri-diagonal matrix and $\boldsymbol{V}$ is an orthonormal column matrix.

  2. Applying numerical methods to compute the eigenvalues and eigenvector of $\boldsymbol{T}$.
  3. If λ and $\boldsymbol{x}$ are an eigenvalue and the corresponding eigenvector of $\boldsymbol{T}$, then λ is also an eigenvalue of $\boldsymbol{A}$ and $\boldsymbol{y} = \boldsymbol{Vx}$ is the corresponding eigenvector. ($\boldsymbol{V}^{-1} = \boldsymbol{V}^{T} = \boldsymbol{V}$)

Introduction (2)

- Matrix ***T*** is organized as follows

$$T = \begin{bmatrix} a_0 & b_0 & & 0 & & 0 \\ b_0 & a_1 & & & & \\ & 0 & \ddots & \ddots & \ddots & b_{n-2} \\ & & \ddots & \ddots & & \\ & 0 & & b_{n-2} & & a_{n-1} \end{bmatrix}$$

Main diagonal = $\{a_0, a_1, \ldots, a_{n-1}\}$.
The two non-zero off-diagonals = $\{b_0, b_1, \ldots, b_{n-2}\}$

- The i-th row of ***T*** is

$T_0 = [a_0 \; b_0 \; 0 \; \ldots 0]$,
$T_{i.} = [0 \quad \ldots \quad b_{i-1} \quad a_i \quad b_i \quad 0 \quad \ldots]$, $1 \leq i \leq n-2$,
$T_{n-1} = [0 \; \ldots 0 \; b_{n-2} \; a_{n-1}]$.

Introduction (3)

- The original Lanczos algorithm for the tri-diagonalization is numerical unstable.
- We use Householder transformation to perform the job.
- Then, we can use any numerical methods to compute the eigenvalues of $\boldsymbol{T}$.
- In this lecture, we use
  - Bi-section method and
  - Power method

Householder Transformation

- Basic terms:
  - Let $v \in R^n, a\ unit\ vector$ and $\boldsymbol{P}: n \times n\ matrix$.
- Householder reflection matrix is defined as:

  $P = I - \frac{2vv^T}{v^Tv},$
  - If $\|v\| = 1, P = I - 2vv^T$.
- Basic properties

  $\boldsymbol{P}$ is symmetric ,

  $\boldsymbol{P^{-1}} = \boldsymbol{P^T} = \boldsymbol{P}$,

  $\boldsymbol{P}$ is orthogonal. (orthonormal)

  $\boldsymbol{P}$ can be used in a similarity transformation

  $P^{-1}AP = P^TAP = PAP$
  - Similarity transformations are used to simplify $\boldsymbol{A}$.

Householder Matrix

- We are interested in a special ***P***, which projects vector ***x*** onto e_0:

  $Px = \alpha e_0$

- The matrix P

  $v = x \pm \|x\| e_0$, // sign = sign of *x[0]*.

  $P = I - \frac{2vv^T}{v^T v}$,

- Example

  $x = [3 \quad 1 \quad 5]^T$, $\|x\| = \sqrt{35}$,
  $v = [3 + \sqrt{35} \quad 1 \quad 5]^T$,
  $P\vec{x} = [\sqrt{35} \quad 0 \quad 0]^T$.

Question

- Can we use Householder's matrices to eliminate ***A*** into a diagonal matrix?
- Answer:
  - We cannot!
  - Similarity transformation (*PAP*) ≠ QR-decomposition transformation (*PA*).
    - 2-side operation vs. 1-side operation.
  - For example, if we eliminate the 0-th column below A_{00} then the entries A_{0k} of the 0-th row will be modified too.
  - If we eliminate the 0-th row by using ***P***, the entries after A_{00} will not be eliminated.

Tri-diagonalization

- QR-decomposition ≈ forward elimination using $\boldsymbol{H_i}$

$$\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \rightarrow {H_0}^{-1} A H_0 \rightarrow$$

$$\begin{bmatrix} \times & \times^* & 0 & 0 \\ \times^* & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \end{bmatrix} \rightarrow {H_1}^{-1} {H_0}^{-1} A H_0 H_1 \rightarrow \cdots T \text{。}$$

$$\mathrm{T} = \begin{bmatrix} \times & \times^* & 0 & 0 \\ \times^* & \times & \times^* & 0 \\ 0 & \times^* & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} \rightarrow \begin{bmatrix} a_0 & b_0 & 0 & 0 \\ b_0 & a_1 & b_1 & 0 \\ 0 & b_1 & a_2 & b_2 \\ 0 & 0 & b_2 & a_3 \end{bmatrix}.$$

Eliminating the j-th Column

- Let $A_{.j} = \left[a_{0j}, a_{1j}, \ldots, a_{jj}, \ldots, a_{n-1,j}\right]^T$ be the j-th column.
- Construct H_j which eliminates the entry below $a_{j+1,j}$.

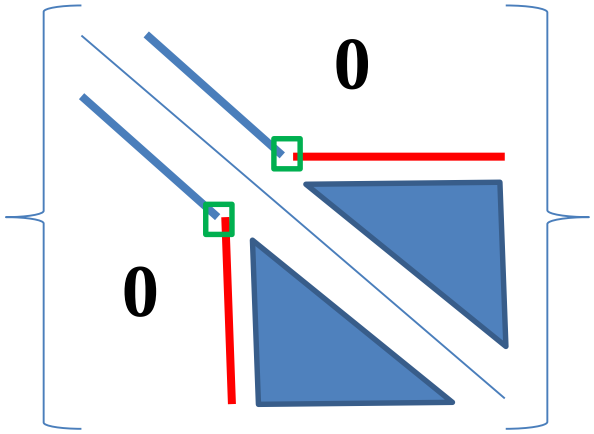
  $H_j A_{.j} = \left[a_{0j}, a_{1j}, \ldots, a_{j+1,j}{}^*, 0, \ldots, 0\right]^T.$
- The algorithm of creating H_j

  $t = [0, \ldots, 0, a_{j+1,j}, \ldots, a_{n-1,j}]^T,$

  $v_j = t \pm \|t\| e_{j+1}$,

  $H_j = I - 2\left(\frac{v_j v_j^T}{v_j^T v_j}\right),$

  $A = H_j A.$



Tri-diagonalization Algorithm

```
for(i=0;i<=n-2;i++){//eliminations
   //create vector v[].
   create_vector_v(A, v, n, i);
   vTv = <v, v>;
   //modify columns and rows.
   for(j=i;j<=n-2;j++){
     //Retrieve the jth row (or column).
     //Vector t = the current row (or col.).
     create_vector_t( A, t, n, i, j);
     vTt = <v, t>;
     //H * t = t - 2 (v^T t / v^T v) v.
     modify_col(A, i, j); //the jth col.
     modify_row(A, i, j); //the jth row
   }
```

```
void create_vector_v(A, v, n, i) //Creating v[].
{
   t[] = {0};
   //retrieve the ith column below A[i][i]
   for(j=i+1;j≤n-1;j++) t[j] = A[j][i];
   //Compute the norm of the vector.
   tTt = inner_product(t, t);
   a = sqrt(tTt);
   //t = t + sign(t[j])*ej;
   if(t[i+1]>=0) t[i+1] = t[i+1] + a;
   else t[i+1] = t[i+1] – a;
  }

void create_vector_t(A, t, n, i, j)
{
   t[] = {0};
   //retrieve the jth column
   for(j=i+1;j≤n-1;j++) t[j] = A[j][i];
}
```

Eigenvalue Computation for Tri-diagonal Matrices

- After the tri-diagonalization process, ***A*** becomes a symmetric tri-diagonal matrix ***T***,
  - Having the same eigenvalues.
- ***T*** is simple, computing its eigenvalues is not trivial.
- But eigenvalue computing algorithms can be sped-up.

Tri-diagonal Matrix

$$
T = \left[\begin{array}{cccc|c}
a_0 & b_0 & & 0 & 0 \\
b_0 & a_1 & & & \\
 & & \ddots & \ddots \quad \ddots & \\
 & 0 & \ddots & \ddots & b_{n-2} \\
\hline
 & 0 & & b_{n-2} & a_{n-1}
\end{array}\right]
$$

The Characteristic Polynomial

The characteristic polynomial can be expressed as,

$$p_n(\lambda) = det(T - \lambda I).$$

$$P_n(\lambda) = (a_{n-1} - \lambda)p_{n-1}(\lambda) - b_{n-2}{}^2 p_{n-2(\lambda)}.$$

Where,

$$p_1(\lambda) = a_0 - \lambda, p_0(\lambda) = 1.$$

$$\begin{bmatrix} a_0 - \lambda & b_0 & 0 & 0 \\ b_0 & a_1 - \lambda & & \\ 0 & & \ddots \quad \ddots \quad \ddots & b_{n-2} \\ 0 & & b_{n-2} & a_{n-1} - \lambda \end{bmatrix}$$

Verify the recurrence equation by

$$B_5 = \begin{bmatrix} s_0 & t_0 & 0 & 0 & 0 \\ t_0 & s_1 & t_1 & 0 & 0 \\ 0 & t_1 & s_2 & t_2 & 0 \\ 0 & 0 & t_2 & s_3 & t_3 \\ 0 & 0 & 0 & t_3 & s_4 \end{bmatrix},$$

$$\det(B_5) = s_4 \det(B_4) - t_3 \det(C),$$

$$C = \begin{bmatrix} s_0 & t_0 & 0 & 0 \\ t_0 & s_1 & t_1 & 0 \\ 0 & t_1 & s_2 & 0 \\ 0 & 0 & t_2 & t_3 \end{bmatrix} = \begin{bmatrix} & B_3 & & \begin{matrix} 0 \\ 0 \\ 0 \end{matrix} \\ 0 & 0 & t_2 & t_3 \end{bmatrix}.$$

$$\det(C) = t_3 \det(B3) - t_2 0 = t_3 \det(B_3) .$$
$$\det(B_5) = s_4 \det(B_4) - t_3 t_3 \det(B_3).$$

Characteristic Polynomial Evaluation

```
double a[], b[]; //keep T in 2 arrays
double p(x, i)
{
    if(i==0) return (1.0);
    if(i==1) return(a[0]-x);
    t1 = p(x, i-1);
    t2 = p(x, i-2);
    return (a[i-1]-x)*t1 –
              b[i-2]*b[i-2]*t2);
}
```

$p_n(\lambda) = (a_{n-1} - \lambda)p_{n-1}(\lambda) - b_{n-2}b_{n-2}p_{n-2(\lambda)}.$

$p_1(\lambda) = a_0 - \lambda,$

$p_0(\lambda) = 1.$

Time complexity = O(n).

Bisection Method for Computing λ

- Given $y < z$ and $p_n(y) * p_n(z) < 0$, we can use bisection method to compute the eigenvalue in [y, z].

```
while(|z-y|>ε){
    λ = (y+z)/2.0;
    if(p(λ,n)*p(y,n)<0.0) z = λ;
    else y = λ;
}
return (λ);
```

Power Method for Computing λ

- Given a vector x, T*x can be simplified.

  $y = T * x$ ::

  $y_0 = a_0 x_0 + b_0 x_1,$
  $y_{n-1} = b_{n-2} x_{n-2} + a_{n-1} x_{n-1},$
  $y_i = b_{i-1} x_{i-1} + a_i x_i + b_i x_{i+1},$
  $1 \le i \le n-2.$

The i-th row of T is

$$T_{i.} = [0 \quad \ldots \quad b_{i-1} \quad a_i \quad b_i \quad 0 \quad \ldots]$$

$$\begin{bmatrix} a_0 & b_0 & & 0 & 0 \\ b_0 & a_1 & & & \\ & 0 & \ddots & \ddots \quad \ddots & b_{n-2} \\ & & \ddots & \ddots & \\ & 0 & & b_{n-2} & a_{n-1} \end{bmatrix}$$

- Thus each iteration of power method can be completed in O(n) steps.

Power Method for Computing λ

x=Ty is computed by::

$x_0 = a_0 y_0 + b_0 y_1,$
$x_{n-1} = b_{n-2} y_{n-2} + a_{n-1} y_{n-1},$
$x_i = b_{i-1} y_{i-1} + a_i y_i + b_i y_{i+1},$
$1 \le i \le n-2.$

$select\ y \neq 0;$
$x = T * y;$
$repeat\{$
$y = \frac{x}{\|x\|};$ // $y^{(k)}$
$x = T * y;$ // $y^{(k+1)}$
$\lambda = \frac{y^T x}{y^T y};$ //eigenvalue.
$r = \lambda * y - x;$
$//r^{(k)} = \lambda^{(k)} y^{(k)} - A y^{(k)}$
$\}until(\|r\| \le \varepsilon);$

Discussion

Time complexity

- The tridiagonalization requires $O(n^3)$ time steps.
- In the bisection method, the time complexity for evaluating $p_n(\lambda)$ is //in each iteration

$T(n) = T(n-1) + T(n-2) + 1$, with

$T(1) = T(0) = 1. \mathrm{T(n)} = \mathrm{O}(n^2)$.

- In total, $O(\log_2 \frac{1}{\varepsilon})$ iterations are required.

Discussion

Using the power method:

- Time complexity for each iteration is O(n).
- We need $O(\log_{\left|\frac{\lambda_0}{\lambda_1}\right|} \frac{1}{\varepsilon})$ iterations.
- If the max eigenvalue is much larger than the 2nd eigenvalue, power method will be a better method.
  - It also gives us the eigenvector.

Discussion

Using the Jacobi method:

- If we apply Jacobi method after the tridiagonalization, how fast can we solve all the eigenvalues and eigenvectors?
- We can optimize the similarity transformation, since ***T*** is tridiagonal.
  - For each similarity transformation, the time complexity is O(1), constant time.
  - Can we limit the number of similarity transformations within O(n) times?
  - After a similarity transformation, ***T*** may be non-tridiagonal.

Inverse Power Method

- Since T is tri-diagonal, shift inverse power method can be sped-up.
- For solving

  $x(k+1) = T^{-1} * y(k);$

  $T * x(k+1) = y(k).$
- The linear system can be solved in O(n) step.
  - Using tridiagonal solver.
- In inverse and shift inverse power method: $O(n^2)$.
- In Rayleigh quotient iteration $O(n^3)$.