Applied Numerical Linear Algebra. Lecture 14

Bisection and Inverse Iteration

- The Bisection algorithm exploits Sylvester's inertia theorem to find only those k eigenvalues that one wants, at cost $O(nk)$. Recall that $\mathrm{Inertia}(A) = (\nu, \zeta, \pi)$, where ν, ζ and π are the number of negative, zero, and positive eigenvalues of A, respectively. Suppose that X is nonsingular; Sylvester's inertia theorem asserts that $\mathrm{Inertia}(A) = \mathrm{Inertia}(X^T A X)$.
- Now suppose that one uses Gaussian elimination to factorize $A - zI = LDL^T$, where L is nonsingular and D diagonal. Then $\mathrm{Inertia}(A - zI) = \mathrm{Inertia}(D)$. Since D is diagonal, its inertia is trivial to compute. (In what follows, we use notation such as "$\#d_{ii} < 0$" to mean "the number of values of d_{ii} that are less than zero.")

We can write $\mathrm{Inertia}(A - zI) = \mathrm{Inertia}(D)$ as:

$$\begin{aligned}
\mathrm{Inertia}(A - zI) &= (\#d_{ii} < 0, \#d_{ii} = 0, \#d_{ii} > 0) \\
&= (\#\,\text{negative eigenvalues of } A - zI, \\
&\quad \#\,\text{zero eigenvalues of } A - zI, \\
&\quad \#\,\text{positive eigenvalues of } A - zI) \\
&= (\#\,\text{eigenvalues of } A < z, \\
&\quad \#\,\text{eigenvalues of } A = z, \\
&\quad \#\,\text{eigenvalues of } A > z).
\end{aligned}$$

The number of eigenvalues in the interval $[z_1, z_2)$

- Suppose $z_1 < z_2$ and we compute $\mathrm{Inertia}(A - z_1 I)$ and $\mathrm{Inertia}(A - z_2 I)$.
- Then the number of eigenvalues in the interval $[z_1, z_2)$ equals (# eigenvalues of $A < z_2$) – (# eigenvalues of $A < z_1$).
- To make this observation into an algorithm, define

$$\mathrm{Negcount}(A, z) = \#\,\text{eigenvalues of } A < z.$$

Bisection algorithm

ALGORITHM. *Bisection: Find all eigenvalues of* A *inside* $[a, b)$ *to a given error tolerance* tol:
$n_a = Negcount(A, a)$
$n_b = Negcount(A, b)$
if $n_a = n_b$, *quit ... because there are no eigenvalues in* $[a, b)$
put $[a, n_a, b, n_b]$ *onto Worklist*
  /* *Worklist contains a list of intervals* $[a, b)$ *containing*
    eigenvalues $n - n_a$ *through* $n - n_b + 1$, *which the algorithm*
    will repeatedly bisect until they are narrower than tol. */
while Worklist is not empty do
  remove $[low, n_{low}, up, n_{up}]$ *from Worklist*
  if $(up - low < \mathrm{tol})$ *then*
    print "there are nup nlow eigenvalues in [low, up)"
  else
    $mid = (low + up)/2$
    $n_{mid} = Negcount(A, mid)$
    if $n_{mid} > n_{low}$ *then ... there are eigenvalues in* $[low, mid)$
      put $[low, n_{low}, mid, n_{mid}]$ *onto Worklist*
    end if
    if $n_{up} > n_{mid}$ *then ... there are eigenvalues in* $[mid, up)$
      put $[mid, n_{mid}, up, n_{up}]$ *onto Worklist*
    end if
  end if
end while

From Negcount(A,z) it is easy to compute Gaussian elimination since

$$A - zI = \begin{bmatrix} a_1 - z & b_1 & \dots & \dots \\ b_1 & a_2 - z & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \dots & b_{n-2} & a_{n-1} - z & b_{n-1} \\ \dots & \dots & b_{n-1} & a_n - z \end{bmatrix} = LDL^T$$

$$= \begin{bmatrix} 1 & \dots & \dots \\ l_1 & 1 & \dots \\ \dots & \dots & \dots \\ \dots & l_{n-1} & 1 \end{bmatrix} \cdot \begin{bmatrix} d_1 & \dots & \dots \\ \dots & d_2 & \dots \\ \dots & \dots & \dots \\ \dots & .. & d_n \end{bmatrix} \cdot \begin{bmatrix} 1 & l_1 \dots & \dots \\ .. & 1 & \dots \\ \dots & \dots & l_{n-1} \\ \dots & \dots & 1 \end{bmatrix}$$

and

$$a_1 - z = d_1, \tag{1}$$

$$d_1 l_1 = b_1, \tag{2}$$

$$l_{i-1}^2 d_{i-1} + d_i = a_i - z, \tag{3}$$

$$d_i l_i = b_i. \tag{4}$$

Substitute $l_i = b_i/d_i$ into $l_{i-1}^2 d_{i-1} + d_i = a_i - z$ to get:

$$d_i = (a_i - z) - \frac{b_{i-1}^2}{d_{i-1}},$$

This is the stable procedure since $A - zI$ is tridiagonal matrix.

Implementation of Negcount(A,z) in Matlab

```
function [ neg ] = Negcount( A,z )
d=zeros(length(A),1);
d(1)=A(1,1)-z;
for i = 2:length(A)
d(i)=(A(i,i)-z)-(A(i,i-1)^2)/d(i-1);
end
%compute number of negative eigenvalues of A
neg=0;
for i = 1:length(A)
if d(i)<0
neg = neg+1;
end
end
end
```

Algorithms for the Symmetric Eigenproblem: Jacobi's Method

Given a symmetric matrix $A = A_0$, Jacobi's method produces a sequence A_1, A_2, ... of orthogonally similar matrices, which eventually converge to a diagonal matrix with the eigenvalues on the diagonal. A_{i+1} is obtained from A_i by the formula $A_{i+1} = J_i^T A_i J_i$, where J_i is an orthogonal matrix called a *Jacobi rotation*. Thus

$$\begin{aligned} A_m &= J_{m-1}^T A_{m-1} J_{m-1} \\ &= J_{m-1}^T J_{m-2}^T A_{m-2} J_{m-2} J_{m-1} = \cdots \\ &= J_{m-1}^T \cdots J_0^T A_0 J_0 \cdots J_{m-1} \\ &= J^T A J. \end{aligned}$$

If we choose each J_i appropriately, A_m approaches a diagonal matrix Λ for large m. Thus we can write $\Lambda \approx J^T A J$ or $J \Lambda J^T \approx A$. Therefore, the columns of J are approximate eigenvectors.
We will make $J^T A J$ nearly diagonal by iteratively choosing J_i to make *one* pair of offdiagonal entries of $A_{i+1} = J_i^T A_i J_i$ zero at a time. We will do this by choosing J_i to be a Givens rotation,

$$J_i = R(j,k,\theta) \equiv \begin{array}{c} \\ j \\ \\ k \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccccc} & & & j & & k & & & \end{array} \\ \begin{bmatrix} 1 & & & & & & & & \\ & 1 & & & & & & & \\ & & \ddots & & & & & & \\ & & & \cos\theta & & -\sin\theta & & & \\ & & & & \ddots & & & & \\ & & & \sin\theta & & \cos\theta & & & \\ & & & & & & \ddots & & \\ & & & & & & & 1 & \\ & & & & & & & & 1 \end{bmatrix} \end{array},$$

where θ is chosen to zero out the j,k and k,j entries of A_{i+1}. To determine θ (or actually $\cos\theta$ and $\sin\theta$),

write

$$\begin{bmatrix} a_{jj}^{(i+1)} & a_{jk}^{(i+1)} \\ a_{kj}^{(i+1)} & a_{kk}^{(i+1)} \end{bmatrix} = \begin{bmatrix} \underbrace{\cos\theta}_{c} & -\sin\theta \\ \underbrace{\sin\theta}_{s} & \cos\theta \end{bmatrix}^T \begin{bmatrix} a_{jj}^{(i)} & a_{jk}^{(i)} \\ a_{kj}^{(i)} & a_{kk}^{(i)} \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$= \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix},$$

where λ_1 and λ_2 are the eigenvalues of

$$\begin{bmatrix} a_{jj}^{(i)} & a_{jk}^{(i)} \\ a_{kj}^{(i)} & a_{kk}^{(i)} \end{bmatrix}.$$

It is easy to compute $\cos\theta$ and $\sin\theta$: Multiplying out the last expression, using symmetry, abbreviating $c \equiv \cos\theta$ and $s \equiv \sin\theta$, and dropping the superscript (i) for simplicity yield

$$\begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} = \begin{bmatrix} a_{jj}c^2 + a_{kk}s^2 + 2sca_{jk} & \underbrace{sc(a_{kk} - a_{jj}) + a_{jk}(c^2 - s^2)}_{=0} \\ \underbrace{sc(a_{kk} - a_{jj}) + a_{jk}(c^2 - s^2)}_{=0} & a_{jj}s^2 + a_{kk}c^2 - 2sca_{jk} \end{bmatrix}.$$

Setting the offdiagonals to 0 and solving for θ we get $0 = sc(a_{kk} - a_{jj}) + a_{jk}(c^2 - s^2)$, or

$$\frac{a_{jj} - a_{kk}}{2a_{jk}} = \frac{c^2 - s^2}{2sc} = \frac{\cos 2\theta}{\sin 2\theta} = \cot 2\theta \equiv \tau.$$

We now let $t = \frac{s}{c} = \tan\theta$ and note that

$$c^2 - s^2 = 2sc\tau, \tag{5}$$

$$c^2 - s^2 - 2sc\tau = 0. \tag{6}$$

Dividing both sides by c^2 and noting that $t = \frac{s}{c}$ we get:

$$1 - t^2 = 2t\tau, \tag{7}$$

$$-(t^2 + 2\tau t - 1) = 0. \tag{8}$$

Thus, we solve $t^2 + 2\tau t - 1 = 0$ to get (via the quadratic formula) $t = \frac{\text{sign}(\tau)}{|\tau| + \sqrt{1+\tau^2}}$, $c = \frac{1}{\sqrt{1+\tau^2}}$ and $s = t \cdot c$. We summarize this derivation in the following algorithm.

ALGORITHM. *Compute and apply a Jacobi rotation to A in coordinates j, k:*

proc Jacobi-Rotation (A, j, k)

if $|a_{jk}|$ *is not too small*

$\tau = (a_{jj} - a_{kk})/(2 \cdot a_{jk})$

$t = \mathrm{sign}(\tau)/(|\tau| + \sqrt{1 + \tau^2})$

$c = 1/\sqrt{1 + \tau^2}$

$s = t \cdot c$

$A = R^T(j, k, \theta) \cdot A \cdot R(j, k, \theta)$ *... where* $c = \cos\theta$ *and* $s = \sin\theta$

if eigenvectors are desired

$J = J \cdot R(j, k, \theta)$

end if

end if

Jacobi's method to find the eigenvalues of a symmetric matrix

The cost of applying $R(j, k, \theta)$ to A (or J) is only $O(n)$ flops, because only rows and columns j and k of A (and columns j and k of J) are modified. The overall Jacobi algorithm is then as follows.

ALGORITHM. *Jacobi's method to find the eigenvalues of a symmetric matrix*:
repeat
    choose $a_{j,k}$ *pair*
    call Jacobi-Rotation(A, j, k)
until A is sufficiently diagonal

We still need to decide how to pick j, k pairs. There are several possibilities. To measure progress to convergence and describe these possibilities, we define

$$\mathrm{off}(A) \equiv \sqrt{\sum_{1 \le j < k \le n} a_{jk}^2}.$$

Thus $\mathrm{off}(A)$ is the root-sum-of-squares of the (upper) offdiagonal entries of A, so A is diagonal if and only if $\mathrm{off}(A) = 0$. Our goal is to make $\mathrm{off}(A)$ approach 0 quickly. The next lemma tells us that $\mathrm{off}(A)$ decreases monotonically with every Jacobi rotation.

LEMMA. Let A' be the matrix after calling Jacobi-Rotation(A, j, k) for any $j \neq k$. Then $\mathrm{off}^2(A') = \mathrm{off}^2(A) - a_{jk}^2$.

The next algorithm was the original version of the algorithm (from Jacobi in 1846), and it has an attractive analysis although it is too slow to use.

ALGORITHM. *Classical Jacobi's algorithm:*

while $\mathrm{off}(A) > \mathrm{tol}$ *(where* tol *is the stopping criterion set by user)*
    choose j *and* k *so* a_{jk} *is the largest off-diagonal entry in magnitude*
    call Jacobi-Rotation(A, j, k)
end while

THEOREM. After one Jacobi rotation in the classical Jacobi's algorithm, we have $\mathrm{off}(A') \leq \sqrt{1 - \frac{1}{N}}\,\mathrm{off}(A)$, where $N = \frac{n(n-1)}{2}$ = the number of superdiagonal entries of A. After k Jacobi-Rotations $\mathrm{off}(\cdot)$ is no more than $(1 - \frac{1}{N})^{k/2}\,\mathrm{off}(A)$.

So the classical Jacobi's algorithm converges at least linearly with the error (measured by $\mathrm{off}(A)$) decreasing by a factor of at least $\sqrt{1 - \frac{1}{N}}$ at a time. In fact, it eventually converges quadratically.

THEOREM. *Jacobi's method is locally quadratically convergent after N steps (i.e., enough steps to choose each a_{jk} once). This means that for i large enough*

$$\mathrm{off}(A_{i+N}) = O(\mathrm{off}^2(A_i)).$$

In practice, we do not use the classical Jacobi's algorithm because searching for the largest entry is too slow: We would need to search $\frac{n^2-n}{2}$ entries for every Jacobi rotation, which costs only $O(n)$ flops to perform, and so for large n the search time would dominate. Instead, we use the following simple method to choose j and k.

ALGORITHM. Cyclic-by-row-Jacobi: Sweep through the off diagonals of A rowwise.

repeat
  for $j = 1$ *to* $n - 1$
   for $k = j + 1$ *to* n
     call Jacobi-Rotation (A, j, k)
   end for
  end for
until A *is sufficiently diagonal*

A no longer changes when Jacobi-Rotation(A, j, k) chooses only $c = 1$ and $s = 0$ for an entire pass through the inner loop. The cyclic Jacobi's algorithm is also asymptotically quadratically convergent like the classical Jacobi's algorithm [J. H. Wilkinson. The Algebraic Eigenvalue Problem. Oxford University Press, Oxford, UK, 1965; p. 270].
The cost of one Jacobi "sweep" (where each j, k pair is selected once) is approximately half the cost of reduction to tridiagonal form and the computation of eigenvalues and eigenvectors using QR iteration, and more than the cost using divide-and-conquer. Since Jacobi's method often takes 5-10 sweeps to converge, it is much slower than the competition.

Algorithms for the Singular Value Decomposition (SVD)

All the algorithms for the eigendecomposition of a symmetric matrix A, except Jacobi's method, have the following structure:

1. Reduce A to tridiagonal form T with an orthogonal matrix Q_1: $A = Q_1 T Q_1^T$.
2. Find the eigendecomposition of T: $T = Q_2 \Lambda Q_2^T$, where Λ is the diagonal matrix of eigenvalues and Q_2 is the orthogonal matrix whose columns are eigenvectors.
3. Combine these decompositions to get $A = (Q_1 Q_2)\Lambda(Q_1 Q_2)^T$. The columns of $Q = Q_1 Q_2$ are the eigenvectors of A.

All the algorithms for the SVD of a general matrix G, except Jacobi's method, have an analogous structure:

1. Reduce G to bidiagonal form B with orthogonal matrices U_1 and V_1: $G = U_1 B V_1^T$. This means B is nonzero only on the main diagonal and first superdiagonal.
2. Find the SVD of B: $B = U_2 \Sigma V_2^T$, where Σ is the diagonal matrix of singular values, and U_2 and V_2 are orthogonal matrices whose columns are the left and right singular vectors, respectively.
3. Combine these decompositions to get $G = (U_1 U_2)\Sigma(V_1 V_2)^T$. The columns of $U = U_1 U_2$ and $V = V_1 V_2$ are the left and right singular vectors of G, respectively.

Practical algorithms for computing the SVD

1. QR iteration and its variations (LR iteration). This is the fastest algorithm for small matrices up to size $n = 25$ to find all the singular values of a bidiagonal matrix.
2. Divide-and-conquer. This is the fastest method to find all singular values and singular vectors for matrices larger than $n = 25$. However, it does not guarantee computation of tiny singular values.
3. Bisection and inverse iteration.
4. Jacobi's method. SVD of a dense matrix G is computed implicitly applying of Jacobi's method to GG^T or G^TG.

QR Iteration and Its Variations : LR iteration

ALGORITHM. *LR iteration: Let T_0 be any symmetric positive definite matrix. The following algorithm produces a sequence of similar symmetric positive definite matrices T_i:*

$i = 0$

repeat

Choose a shift τ_i^2 smaller than the smallest eigenvalue of T_i.

Compute the Cholesky factorization $T_i - \tau_i^2 I = B_i^T B_i$

(B_i is an upper triangular matrix with positive diagonal.)

$T_{i+1} = B_i B_i^T + \tau_i^2 I$

$i = i + 1$

until convergence

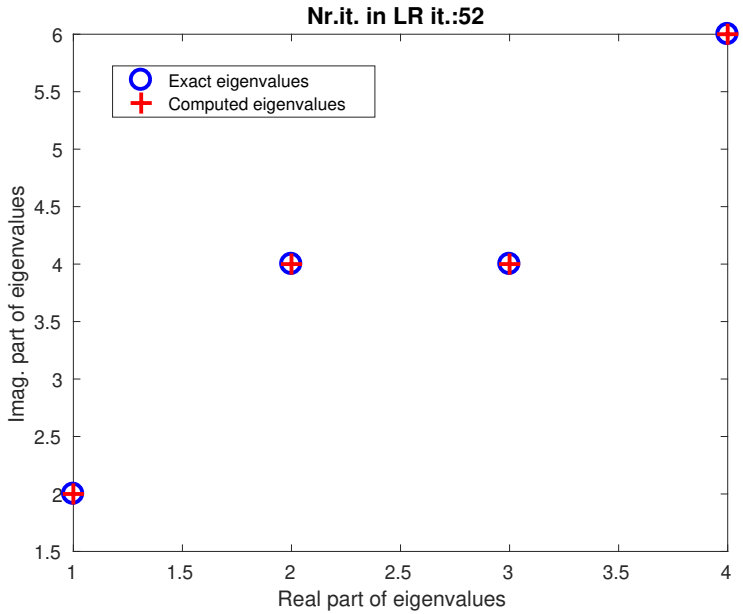
LR iteration versus QR iteration

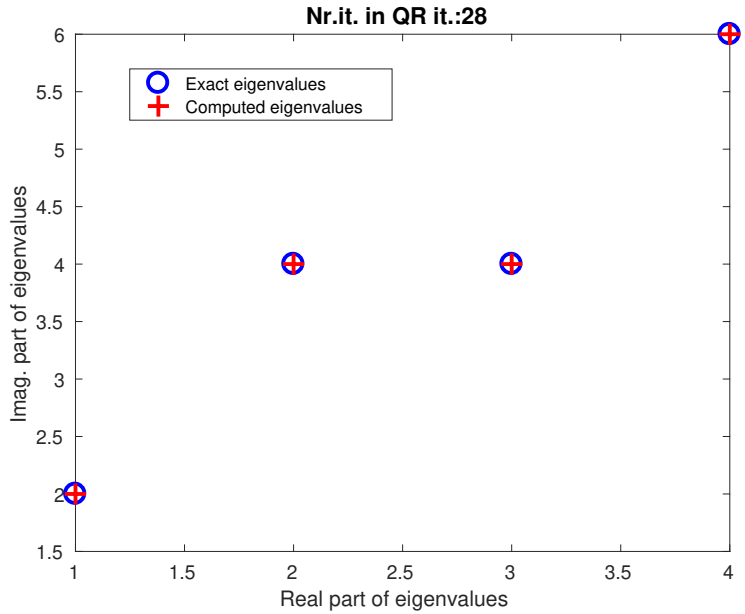
LR iteration is very similar in structure to QR iteration: We compute a factorization, and multiply the factors in reverse order to get the next iterate T_{i+1}. It is easy to see that T_{i+1} and T_i are similar:

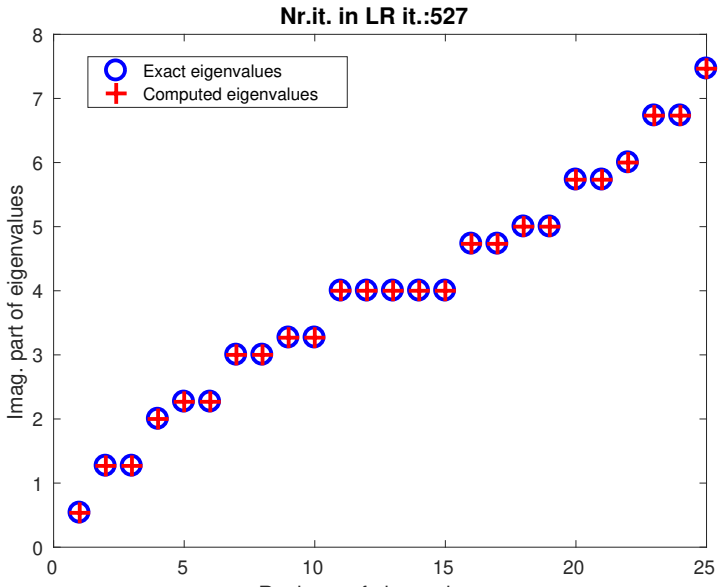
$$\begin{aligned} T_{i+1} &= B_i B_i^T + \tau_i^2 I = \underbrace{B_i^{-T} B_i^T}_{I} B_i B_i^T + \tau_i^2 \underbrace{B_i^{-T} B_i^T}_{I} \\ &= B_i^{-T} (\underbrace{B_i^T B_i + \tau_i^2}_{T_i}) B_i^T = B_i^{-T} T_i B_i^T . \end{aligned} \tag{9}$$

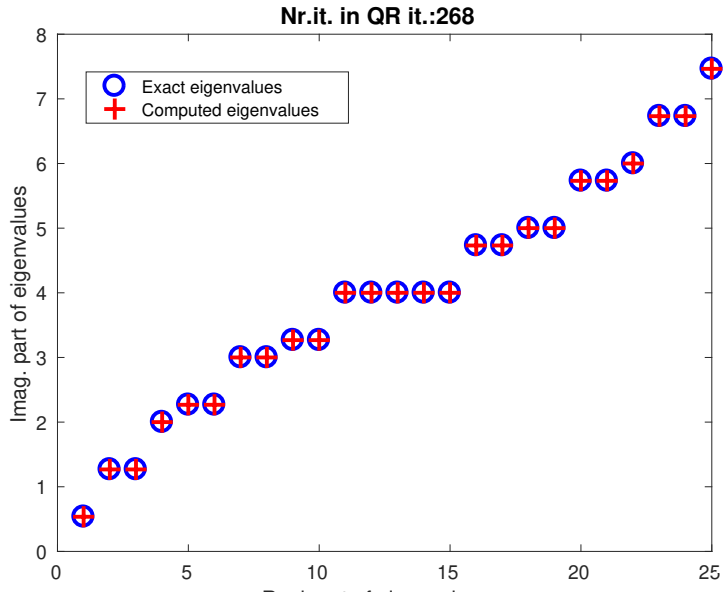
In fact, when the shift $\tau_i^2 = 0$, we can show that two steps of LR iteration produce the same T_2 as one step of QR iteration. We show that on the next slide numerically.

LR iteration versus QR iteration









Jacobi's Method for the SVD

ALGORITHM. *Compute and apply a one-sided Jacobi rotation to G in coordinates j, k:*

proc One-Sided-Jacobi-Rotation (G, j, k)

Compute $a_{jj} = (G^T G)_{jj}$, $a_{jk} = (G^T G)_{jk}$, *and* $a_{kk} = (G^T G)_{kk}$

if $|a_{jk}|$ *is not too small*

$\tau = (a_{jj} - a_{kk})/(2 \cdot a_{jk})$

$t = sign(\tau)/(|\tau| + \sqrt{1 + \tau^2})$

$c = 1/\sqrt{1 + t^2}$

$s = c \cdot t$

$G = G \cdot R(j, k, \theta)$ *... where* $c = \cos\theta$ *and* $s = \sin\theta$

if right singular vectors are desired

$J = J \cdot R(j, k, \theta)$

end if

end if

Note that the *jj*, *jk*, and *kk* entries of $A = G^T G$ are computed by procedure One-Sided-Jacobi-Rotation, after which it computes the Jacobi rotation $R(j, k, \theta)$ in the same way as procedure Jacobi-Rotation.

ALGORITHM. *One-sided Jacobi: Assume that G is n-by-n. The outputs are the singular values σ_i, the left singular vector matrix U, and the right singular vector matrix V so that $G = U\Sigma V^T$, where $\Sigma = \mathrm{diag}(\sigma_i)$.*

repeat
  for $j = 1$ to $n-1$
    for $k = j+1$ to n
      call One-Sided-Jacobi-Rotation (G, j, k)
    end for
  end for
until $G^T G$ is diagonal enough
Let $\sigma_i = ||G(:,i)||_2$ (the 2-norm of column i of G)
Let $U = [u_1, \ldots, u_n]$, where $u_i = G(:,i)/\sigma_i$
let $V = J$, the accumulated product of Jacobi rotations

The following theorem shows that one-sided Jacobi can compute the SVD to high relative accuracy, despite roundoff, provided that we can write $G = DX$, where D is diagonal and X is well-conditioned.
THEOREM. Let $G = DX$ be an n-by-n matrix, where D is diagonal and nonsingular, and X is nonsingular. Let $\hat{G}$ be the matrix after calling One-Sided-Jacobi-Rotation (G, j, k) m times in floating point arithmetic. Let $\sigma_1 \geq \ldots \geq \sigma_n$ be the singular values of G, and let $\hat{\sigma}_1 \geq \ldots \geq \hat{\sigma}_n$ be the singular values of $\hat{G}$. Then

$$\frac{|\sigma_i - \hat{\sigma}_i|}{\sigma_i} \leq O(m\varepsilon)\kappa(X),$$

where $\kappa(X) = ||X|| \cdot ||X^{-1}||$ is the condition number of X. In other words, the relative error in the singular values is small if the condition number of X is small.

Numerical Solution of Poisson's Equation (see Lecture 5)

The model problem is the following Dirichlet problem for Poisson's equation:

$$\begin{aligned} -\triangle u(x) &= f(x) \quad \text{in} \quad \Omega, \\ u &= 0 \text{ on } \partial\Omega. \end{aligned} \tag{10}$$

Here, $f(x)$ is a given function, $u(x)$ is the unknown function, and the domain Ω is the unit square $\Omega = \{(x_1, x_2) \in (0,1) \times (0,1)\}$. To solve numerically (10) we first discretize the domain Ω with $x_{1\,i} = ih_1$ and $x_{2\,j} = jh_2$, where $h_1 = 1/(n_i - 1)$ and $h_2 = 1/(n_j - 1)$ are the mesh sizes in the directions x_1, x_2, respectively, n_i and n_j are the numbers of discretization points in the directions x_1, x_2, respectively. Usually, in computations we have the same mesh size $h = h_1 = h_2$. In this example we choose $n_i = n_j = n$ with $n = N + 2$, where N is the number of inner nodes in the directions x_1, x_2, respectively.
Indexes (i, j) are such that $0 < i, j \leq n$ and are associated with every global node n_{glob} of the finite difference mesh. Global nodes numbers n_{glob} in two-dimensional case can be computed using the following formula:

$$n_{glob} = i + n_j \cdot (j - 1). \tag{11}$$

We use the standard finite difference discretization of the Laplace operator Δu in two dimensions and obtain discrete laplacian $\Delta u_{i,j}$:

$$\Delta u_{i,j} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}, \tag{12}$$

where $u_{i,j}$ is the solution at the discrete point (i, j). Using (12), we obtain the following scheme for solving problem (10):

$$-\left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}\right) = f_{i,j}, \tag{13}$$

where $f_{i,j}$ are the value of the function f at the discrete point (i, j). Then (13) can be rewritten as

$$-\left(u_{i+1,j} - 2u_{i,j} + u_{i-1,j} + u_{i,j+1} - 2u_{i,j} + u_{i,j-1}\right) = h^2 f_{i,j}, \tag{14}$$

or in the more convenient form as

$$-u_{i+1,j} + 4u_{i,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} = h^2 f_{i,j}. \tag{15}$$

System (15) can be written in the form $Au = b$. The vector b has the components $b_{i,j} = h^2 f_{i,j}$. The explicit elements of the matrix A are given by the following block matrix

$$A = \left(\begin{array}{c|c|c} A_N & -I_N & \\ \hline -I_N & \ddots & \ddots \\ & \ddots & \ddots & -I_N \\ \hline & & -I_N & A_N \end{array}\right)$$

with blocks A_N of order N given by

$$A_N = \begin{pmatrix} 4 & -1 & 0 & 0 & \cdots & 0 \\ -1 & 4 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 4 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & 0 & -1 & 4 \end{pmatrix},$$

which are located on the diagonal of the matrix A, and blocks with the identity matrices $-I_N$ of order N on its off-diagonals. The matrix A is symmetric and positive definite. Therefore, we can use the LU factorization algorithm without pivoting.

Introduction to Iterative Methods for Solution of Linear Systems

We will discuss following basic iterative methods:

- 1. *Jacobi*.
- 2. *Gauss-Seidel*.
- 3. *Successive overrelaxation (SOR)*.

as well as Krylov subspace methods using conjugate gradient (CG) algorithm and preconditioning.

Basic Iterative Methods

The basic iterative methods for the solution of system of linear equations $Ax = b$
are:

- 1. *Jacobi*.
- 2. *Gauss-Seidel*.
- 3. *Successive overrelaxation (SOR)*.

These methods produce a sequence of iterative solutions x_m which converge to the solution $x = A^{-1}b$ provided that there exist initial guess x_0. To use iterative methods we will introduce a splitting: $A = M - K$, where $\det M \neq 0$. Applying this splitting to $Ax = b$ we get:

$$Ax = Mx - Kx = b.$$

From the equation above we can get

$$Mx = b + Kx$$

and thus

$$x = M^{-1}(b + Kx) = M^{-1}b + M^{-1}Kx.$$

Let us define

$$Rx = M^{-1}Kx, c = M^{-1}b.$$

The the iterative update for x_m can be written as:

$$x_{m+1} = Rx_m + c, \tag{16}$$

where m is the number of iteration.

Lemma

Let $\|R\| = \max_{x \neq 0} \frac{\|Rx\|}{\|x\|}$. *If* $\|R\| < 1$ *then iterations (16) will converge for all initial guesses* x_0.

Theorem

Let $\|R\| = \max_{x \neq 0} \frac{\|Rx\|}{\|x\|}$. *If* $\rho(R) < 1$ *then iterations (16) will converge for all initial guesses* x_0.

Proof.

Subtracting equation for exact solution $x = Rx + c$ from (16) we have:

$$\begin{aligned} \|x_{m+1} - x\| = \|R(x_m - x)\| &\leq \|R\| \cdot \|x_m - x\| \\ &\leq \|R^{m+1}\| \cdot \|x_0 - x\| = \lambda_{max}^{m+1} \|x_0 - x\|. \end{aligned} \tag{17}$$

Since $\rho(R) < 1$ then using lemma above we choose a such operator norm that $\|R\|_{(R,\varepsilon)} < 1$. Then by lemma 6 iterations (16) will converge for all initial guesses x_0.

□

In the iterative methods considered below we want to have efficient splitting $A = M - K$ as possible. Let us introduce following notations: If A has no zeros on its diagonal we will write the splitting as

$$A = D - \tilde{L} - \tilde{U} = D(I - L - U), \tag{18}$$

where D is a diagonal matrix, $-\tilde{L}$ is the strictly lower triangular part of A such that $DL = \tilde{L}$, and $-\tilde{U}$ is the strictly upper triangular part of A such that $DU = \tilde{U}$.

Jacobi Method

The splitting for Jacobi method is:

$$A = D - (\tilde{L} + \tilde{U}). \tag{19}$$

Applying it to the solution of $Ax = b$ we have:

$$Ax = Dx - (\tilde{L}x + \tilde{U}x) = b.$$

From the equation above we can get

$$Dx = b + \tilde{L}x + \tilde{U}x$$

and thus

$$x = D^{-1}(b + \tilde{L}x + \tilde{U}x) = D^{-1}b + D^{-1}\tilde{L}x + D^{-1}\tilde{U}x.$$

Let us define

$$\begin{aligned} R_J &\equiv D^{-1}(\tilde{L} + \tilde{U}) = L + U, \\ c_J &\equiv D^{-1} b. \end{aligned} \tag{20}$$

Then iterative update in the Jacobi method can be written as:

$$x_{m+1} = R_J x_m + c_J. \tag{21}$$

After multiplication by D both sides of (21) and using (20) we get

$$Dx_{m+1} = D(R_J x_m + c_J) = D(D^{-1}(\tilde{L} + \tilde{U}))x_m + DD^{-1}b = b + \tilde{L}x_m + \tilde{U}x_m. \tag{22}$$

Using the definition of D on the element level the formula (22) can be written as

$$a_{j,j} x_{m+1,j} = b_j - \sum_{k \neq j} a_{j,k} x_{m,k} \tag{23}$$

Algorithms: one step in the Jacobi method

for $j = 1$ to n

$$x_{m+1,j} = \frac{b_j - \sum_{k \neq j} a_{j,k} x_{m,k}}{a_{j,j}}$$

end

In the case of the model problem for the Poisson's equation on a square we will have the following Jacobi's method:

for $i = 1$ to N

for $j = 1$ to N

$$u_{m+1,i,j} = \frac{u_{m,i-1,j} + u_{m,i+1,j} + u_{m,i,j-1} + u_{m,i,j+1} + h^2 f_{i,j}}{4}$$

end

end

Gauss-Seidel Method

To get the Gauss-Seidel method we use the same splitting (19) as for the Jacobi method. Applying it to the solution of $Ax = b$ we have:

$$Ax = Dx - (\tilde{L}x + \tilde{U}x) = b,$$

we rearrange terms in the right hand side now like that:

$$Dx - \tilde{L}x = b + \tilde{U}x \tag{24}$$

and thus now the solution is computed as

$$x = (D - \tilde{L})^{-1}(b + \tilde{U}x) = (D - \tilde{L})^{-1}b + (D - \tilde{L})^{-1}\tilde{U}x.$$

We can rewrite the above equation using notations $DL = \tilde{L}$ and $DU = \tilde{U}$ as:

$$\begin{aligned} x &= (D - \tilde{L})^{-1} b + (D - \tilde{L})^{-1} \tilde{U} x \\ &= (D - DL)^{-1} b + (D - DL)^{-1} \tilde{U} x \\ &= (I - L)^{-1} D^{-1} b + (I - L)^{-1} D^{-1} \tilde{U} x \\ &= \underbrace{(I - L)^{-1} D^{-1} b}_{c_{GS}} + \underbrace{(I - L)^{-1} U}_{R_{GS}} x. \end{aligned} \tag{25}$$

Let us define

$$\begin{aligned} R_{GS} &\equiv (I - L)^{-1} U, \\ c_{GS} &\equiv (I - L)^{-1} D^{-1} b. \end{aligned} \tag{26}$$

Then iterative update in the Gauss-Seidel method can be written as:

$$x_{m+1} = R_{GS} x_m + c_{GS}. \tag{27}$$

Formula (25) in iterative update after multiplication by $(I - L)D$ can be also written as

$$(I - L)Dx_{m+1} = b + DUx_m \tag{28}$$

or

$$Dx_{m+1} - LDx_{m+1} = b + DUx_m, \tag{29}$$

$$Dx_{m+1} = b + DUx_m + LDx_{m+1} \tag{30}$$

or using the definition of D, L, U on the element level as

$$a_{j,j} x_{m+1,j} = b_j - \sum_{k=1}^{j-1} a_{j,k} x_{m+1,k} - \sum_{k=j+1}^{n} a_{j,k} x_{m,k}. \tag{31}$$

Here, terms $\sum_{k=1}^{j-1} a_{j,k} x_{m+1,k}$ represent already updated terms with x_{m+1} which we can use in computations since we have computed them, and terms $\sum_{k=j+1}^{n} a_{j,k} x_{m,k}$ are with older x_m which we have updated on the iteration m.

Algorithm: one step in the Gauss-Seidel method

for $j = 1$ to n

$$x_{m+1,j} = \frac{b_j - \sum_{k=1}^{j-1} a_{j,k} x_{m+1,k} - \sum_{k=j+1}^{n} a_{j,k} x_{m,k}}{a_{j,j}}$$

end

Gauss-Seidel method for the Poisson's equation

If we want apply the Gauss-Seidel method for the solution of the model problem for the Poisson's equation we need organize ordering for the new $m+1$ variables and old already computed values m. We will use such called red-black ordering based on the chessboard-like coloring. Let **B** nodes correspond to the black squares on a chessboard, and **R** nodes correspond to the weight squares. The the the Gauss-Seidel method for the solution of the two-dimensional Poisson's equation on a square becomes the following.

for all **R** red nodes i, j

$$u_{m+1,i,j} = \frac{u_{m,i-1,j} + u_{m,i+1,j} + u_{m,i,j-1} + u_{m,i,j+1} + h^2 f_{i,j}}{4}$$

end

for all **B** black nodes i, j

$$u_{m+1,i,j} = \frac{u_{m+1,i-1,j} + u_{m+1,i+1,j} + u_{m+1,i,j-1} + u_{m+1,i,j+1} + h^2 f_{i,j}}{4}$$

end

Successive Overrelaxation SOR(ω) Method

The method of successive overrelaxation improves the Gauss-Seidel method in the following way: it takes weighted average of values x_{m+1} and x_m such that:

$$x_{m+1,j} = (1-\omega)x_{m,j} + \omega x_{m+1,j}, \tag{32}$$

where ω is a weight called also relaxation parameter. When $\omega = 1$, then we will get usual Gauss-Seidel method, when $\omega < 1$ we get underrelaxation method, and when $\omega > 1$ - overrelaxation method.
To get SOR(ω) method in a matrix form, we again apply splitting (19) and obtain equation similar to the equation (24) obtained in Gauss-Seidel, but only in the iterative form:

$$(D - \tilde{L})x_{m+1} = b + \tilde{U}x_m. \tag{33}$$

Applying now weighted average (32) to this equation we have:

$$\begin{aligned}
(D-\tilde{L})((1-\omega)x_m+\omega x_{m+1}) &= b+\tilde{U}x_m,\\
(D-\tilde{L})(1-\omega)x_m+(D-\tilde{L})\omega x_{m+1} &= b+\tilde{U}x_m,\\
(D-\tilde{L})\omega x_{m+1} &= b+\tilde{U}x_m-(D-\tilde{L})(1-\omega)x_m,\\
(D-\tilde{L})\omega x_{m+1} &= b+\tilde{U}x_m-D(1-\omega)x_m+\tilde{L}(1-\omega)x_m.
\end{aligned} \tag{34}$$

From (32) we see that $(1-\omega)x_m = x_{m+1}-\omega x_{m+1} = (1-\omega)x_{m+1}$, and thus the last expression we can write as

$$\begin{aligned}
(D-\tilde{L})\omega x_{m+1} &= b+\tilde{U}x_m-D(1-\omega)x_m+\tilde{L}(1-\omega)x_{m+1},\\
(D-\tilde{L})\omega x_{m+1}-\tilde{L}(1-\omega)x_{m+1} &= b+\tilde{U}x_m-D(1-\omega)x_m,\\
(D\omega-\tilde{L})x_{m+1} &= b+\tilde{U}x_m-D(1-\omega)x_m
\end{aligned} \tag{35}$$

Denoting $\omega := \frac{1}{\omega}$ we get SOR-scheme:

$$(D-\omega\tilde{L})x_{m+1} = \omega b+((1-\omega)D+\omega\tilde{U})x_m. \tag{36}$$

Using notations $DL = \tilde{L}$ and $DU = \tilde{U}$ the equation (36) can be rewritten as

$$\begin{aligned} x_{m+1} &= (D - \omega\tilde{L})^{-1}\omega b + (D - \omega\tilde{L})^{-1}((1-\omega)D + \omega\tilde{U})x_m \\ &= (I - \omega L)^{-1}D^{-1}\omega b + (I - \omega L)^{-1}((1-\omega)I + \omega U)x_m. \end{aligned} \tag{37}$$

Now defining

$$\begin{aligned} R_{SOR} &= (I - \omega L)^{-1}((1-\omega)I + \omega U), \\ c_{SOR} &= (I - \omega L)^{-1}D^{-1}\omega b \end{aligned} \tag{38}$$

we can rewrite (37) in the form

$$x_{m+1} = R_{SOR}x_m + c_{SOR}. \tag{39}$$

Algorithm: one step in the SOR(ω) method

To get SOR(ω) for implementation, we take $x_{m+1,j}$ in the right hand side of (32) from the Gauss-Seidel algorithm and obtain the following algorithm:

One step in the SOR(ω) method

for $j = 1$ to n

$$x_{m+1,j} = (1-\omega)x_{m,j} + \omega \left[\frac{b_j - \sum_{k=1}^{j-1} a_{j,k}x_{m+1,k} - \sum_{k=j+1}^{n} a_{j,k}x_{m,k}}{a_{j,j}} \right]$$

end

Algorithm: SOR(ω) for the solution of the Poisson's equation

To apply the SOR(ω) method for the solution of the model problem for the Poisson's equation we will use the red-black ordering as in the Gauss-Seidel method. The SOR(ω) method will be the following.

One step in the SOR(ω) method for two-dimensional Poisson's equation

for all **R** red nodes i, j

$$u_{m+1,i,j} = (1-\omega)u_{m,i,j} + \frac{\omega(u_{m,i-1,j} + u_{m,i+1,j} + u_{m,i,j-1} + u_{m,i,j+1} + h^2 f_{i,j})}{4}$$

end

for all **B** black nodes i, j

$$u_{m+1,i,j} = (1-\omega)u_{m,i,j} + \frac{\omega(u_{m+1,i-1,j} + u_{m+1,i+1,j} + u_{m+1,i,j-1} + u_{m+1,i,j+1} + h^2 f_{i,j})}{4} \tag{40}$$

end

Convergence of SOR(ω)

Theorem

If the matrix A is strictly row diagonally dominant (if $|a_{ii}| > \sum_{i \neq j} |a_{ij}|$), then Jacobi and Gauss-Seidel methods converge such that

$$\|R_{GS}\|_\infty < \|R_J\|_\infty < 1, \tag{41}$$

where R_{GS} and R_J are defined in (26), (20), *respectively.*

Proof follows from following Lemma:

Lemma

Let $\|R\| = \max_{x \neq 0} \frac{\|Rx\|}{\|x\|}$. If $\|R\| < 1$ then iterations (16) will converge for all initial guesses x_0.

Theorem

Let the spectral radius of R_{SOR} is such that $\rho(R_{SOR}) \geq |\omega - 1|$. Then $0 < \omega < 2$ is required for convergence of SOR(ω).

Proof.
We write the characteristic polynomial for R_{SOR} as

$$\begin{aligned} \varphi(\lambda) &= \det(\lambda I - R_{SOR}) = \det(\lambda I - (I - \omega L)^{-1}((1-\omega)I + \omega U)) \\ &= \det((I - \omega L)^{-1}((I - \omega L)\lambda I - ((1-\omega)I + \omega U))) \\ &= \det((\lambda + \omega - 1)I - \omega\lambda L - \omega U). \end{aligned} \tag{42}$$

From the equation above we have (since $\varphi(0)$ is an upper triangular matrix)

$$\varphi(0) = \pm \prod \lambda_i(R_{SOR}) = \pm \det((\omega - 1)I) = \pm(\omega - 1)^n,$$

and thus

$$\max_i |\lambda_i(R_{SOR})| \geq |\omega - 1|,$$

from what follows that $\rho(R_{SOR}) \geq |\omega - 1|$.

For convergence from Lemma

Lemma

Let $\|R\| = \max_{x \neq 0} \frac{\|Rx\|}{\|x\|}$. *If* $\|R\| < 1$ *then iterations (16) will converge for all initial guesses* x_0.

we should have

$$\rho(R_{SOR}) < 1$$

and thus

$$|\omega - 1| < 1$$

or

$$-1 < \omega - 1 < 1$$

and

$$0 < \omega < 2.$$

Theorem

If A is s.p.d matrix then $\rho(R_{SOR}) < 1$ for all $0 < \omega < 2$.

Example.
The matrix A in the model problem for the Poisson's equation is s.p.d..
Thus, SOR(ω) for this problem will converge for all $0 < \omega < 2$.

Krylov Subspace Methods

Krylov subspace methods are used for the solution of large system of linear equations $Ax = b$ and for finding eigenvalues of A avoiding matrix-matrix multiplication. Instead, these methods use multiplication of matrix by the vector.

Definition

The Krylov subspace generated by matrix A of the size $n \times n$ and vector b of the size n is the linear subspace spanned by powers of A and multiplied by b:

$$K_r(A, b) = \{b, Ab, A^2 b, ..., A^{r-1} b\}. \tag{43}$$

For the symmetric matrix A we can write the decomposition $Q^T AQ = H$, where Q is the orthogonal transformation and H is the upper Hessenberg matrix which also will be a lower Hessenberg, and thus, tridiagonal matrix. Writing Q as $Q = \{q_1, ..., q_n\}$ and using $AQ = QH$ we have

$$Aq_j = \sum_{i=1}^{j+1} h_{i,j} q_i. \tag{44}$$

We multiply both sides of the above expression by orthonormal vectors q_m^T and use the fact that q_i are orthonormal to obtain:

$$q_m^T A q_j = \sum_{i=1}^{j+1} h_{i,j} q_m^T q_i = h_{m,j},\ 1 \leq m \leq j. \tag{45}$$

We can rewrite (44) as

$$h_{j+1,j} q_{j+1} = A q_j - \sum_{i=1}^{j} h_{i,j} q_i. \tag{46}$$

The formula above as well as (45) are used in the Arnoldi algorithm for the reduction of matrix A to upper Hessenberg form. Let r will be the number of columns in the matrices Q and H which we need to compute. We now formulate the Arnoldi algorithm which performs partial reduction to Hessenberg form. The vectors q_j computed in this algorithm are called Arnoldi vectors.

Arnoldi algorithm

Initialization: $q_1 = \frac{b}{\|b\|_2}$
for $j = 1$ to r
$z = Aq_j$
for $i = 1$ to j
$h_{i,j} = q_i^T z$
$z = z - h_{i,j} q_i$
end
$h_{j+1,j} = \|z\|_2$
if $h_{j+1,j} = 0$ quit
$q_{i+1} = \frac{z}{h_{j+1,j}}$
end

For the case of a symmetric matrix A the Arnoldi algorithm can be simplified since the matrix H is symmetric and tridiagonal what means that

$$H = \begin{pmatrix} \alpha_1 & \beta_1 & \dots & \dots \\ \beta_1 & \alpha_2 & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \alpha_{n-1} & \beta_{n-1} \\ \dots & \dots & \beta_{n-1} & \alpha_n \end{pmatrix} \tag{47}$$

Rewriting (44) for the case of the symmetric and tridiagonal H given by (47) we have

$$Aq_j = \beta_{j-1} q_{j-1} + \alpha_j q_j + \beta_j q_{j+1}. \tag{48}$$

We note that columns of Q are orthonormal and thus

$$q_j^T A q_j = q_j^T (\beta_{j-1} q_{j-1} + \alpha_j q_j + \beta_j q_{j+1}) = \alpha_j. \tag{49}$$

Combining (49) and (48) we get Lanczos algorithm for partial reduction to symmetric tridiagonal form.

Lanczos algorithm

Initialization: $q_1 = \frac{b}{\|b\|_2}, \beta_0 = 0, q_0 = 0$
for $j = 1$ to r
$z = Aq_j$
$\alpha_j = q_j^T z$
$z = z - \alpha_j q_j - \beta_{j-1} q_{j-1}$
$\beta_j = \|z\|_2$
if $\beta_j = 0$ quit
$q_{i+1} = \frac{z}{\beta_j}$
end

The vectors q_j computed by the algorithm are called Lanczos vectors. The vectors q_r computed in the Lanczos or Arnoldi algorithm create orthonormal basis of the Krylov subspace K_r defined in (43). The matrix $H_r = Q_r^T A Q_r$ in both algorithms is called the projection of A to the Krylov subspace K_r.
Our goal now to use r steps in the Lanczos or Arnoldi algorithms to solve linear system $Ax = b$. To do that we seek best approximation x_r to the exact solution $x = A^{-1}b$ given by

$$x_r = \sum_{j=1}^{r} z_j q_j = Q_r z, \tag{50}$$

where $z = (z_1, ..., z_r)^T$.

Let us define the residual as $R_r = b - Ax_r$. For the case of s.p.d. matrix A we can define the norm $\|R\|_{A^{-1}} := (R^T A^{-1} R)^{1/2}$. We note that $\|R\|_{A^{-1}} = \|x_r - x\|_A$. Thus, the the best computed solution x_r will minimize $\|R\|_{A^{-1}}$. The algorithm which can compute such vector x_r is called the conjugate gradient (CG) algorithm.

Theorem

Let A is a symmetric matrix, $H_r = Q_r^T A Q_r$ and residuals are defined as $R_r = b - Ax_r \forall x_r \in K_r$. When H_r is not singular we can define

$$x_r = Q_r H_r^{-1} e_1 \|b\|_2, \tag{51}$$

where $e_1 = (1, 0, ..., 0)^T$. Then $Q_r^T R_r = 0$.
Let A is also positive definite matrix. Then H_r must be nonsingular and x_r defined as in (51) minimizes $\|R_r\|_{A^{-1}}$ for all $x_r \in K_r$, where $R_r = \pm \|R_r\|_2 q_{r+1}$.

Conjugate gradient algorithm

Now we introduce *conjugate gradients* vectors p_r. The p_r are called *gradients* because in a single step of the CG algorithm we compute the approximated solution as $x_r = x_{r-1} + \nu p_r$ and this solution minimizes the residual norm $\|R_r\|_{A^{-1}} = (R_r^T A^{-1} R_r)^{1/2}$. The vectors p_r are called *conjugate*, or more precisely *A-conjugate*, because $p_r^T A p_j = 0$ if $j \neq r$.

Conjugate gradient algorithm

Initialization: $r = 0; \quad x_0 = 0; \quad R_0 = b; \quad p_1 = b;$

repeat

$r = r + 1$

$z = A \cdot p_r$

$\nu_r = (R_{r-1}^T R_{r-1})/(p_r^T z)$

$x_r = x_{r-1} + \nu_r p_r$

$R_r = R_{r-1} - \nu_r z$

$\mu_{r+1} = (R_r^T R_r)/(R_{r-1}^T R_{r-1})$

$p_{r+1} = R_r + \mu_{r+1} p_r$

until $||R_r||_2$ *is small enough*

Preconditioning for Linear Systems

Preconditioning technique is used for the reduction of the condition number of the considered problem. For the solution of linear system of equations $Ax = b$ the preconditioner matrix P of a matrix A is a matrix $P^{-1}A$ such that $P^{-1}A$ has a smaller condition number then the original matrix A. This means that instead of the solution of a system $Ax = b$ we will consider solution of the system

$$P^{-1}Ax = P^{-1}b. \tag{52}$$

The matrix P should have the following properties:

- P is s.p.d. matrix;
- $P^{-1}A$ is well conditioned;
- The system $Px = b$ should be easy solvable.

The preconditioned conjugate gradient method is derived as follows. First we multiply both sides of (52) by $P^{1/2}$ to get

$$(P^{-1/2}AP^{-1/2})(P^{1/2}x) = P^{-1/2}b. \tag{53}$$

We note that the system (53) is s.p.d. since we have chosen the matrix P such that $P = QQ^T$ which is the eigendecomposition of P. Then the matrix $P^{1/2}$ will be s.p.d. if it is defined as

$$P^{1/2} = Q^{1/2}Q^T.$$

Defining

$$\tilde{A} := P^{-1/2}AP^{-1/2}, \tilde{x} := P^{1/2}x, \tilde{b} = P^{-1/2}b$$

we can rewrite (53) as the system $\tilde{A}\tilde{x} = \tilde{b}$. Matrices $\tilde{A}$ and $P^{-1}A$ are similar since $P^{-1}A = P^{-1/2}\tilde{A}P^{1/2}$. Thus, $\tilde{A}$ and $P^{-1}A$ have the same eigenvalues. Thus, instead of the solution of $P^{-1}Ax = P^{-1}b$ we will present preconditioned conjugate gradient (PCG) algorithm for the solution of $\tilde{A}\tilde{x} = \tilde{b}$.

Preconditioned conjugate gradient algorithm

Initialization: $r = 0; \quad x_0 = 0; \quad R_0 = b; \quad p_1 = P^{-1}b; y_0 = P^{-1}R_0$

repeat

$r = r + 1$

$z = A \cdot p_r$

$\nu_r = (y_{r-1}^T R_{r-1})/(p_r^T z)$

$x_r = x_{r-1} + \nu_r p_r$

$R_r = R_{r-1} - \nu_r z$

$y_r = P^{-1} R_r$

$\mu_{r+1} = (y_r^T R_r)/(y_{r-1}^T R_{r-1})$

$p_{r+1} = y_r + \mu_{r+1} p_r$

until $||R_r||_2$ *is small enough*

Common preconditioners

Common preconditioner matrices P are:

- Jacobi preconditioner $P = (a_{11}, ..., a_{nn})$. Such choice of the preconditioner reduces the condition number of $P^{-1}A$ around factor n of its minimal value.
- block Jacobi preconditioner

$$P = \begin{pmatrix} P_{1,1} & ... & 0 \\ ... & ... & ... \\ 0 & ... & P_{r,r} \end{pmatrix} \tag{54}$$

with $P_{i,i} = A_{i,i}, i = 1, ..., r$, for the block matrix A given by

$$A = \begin{pmatrix} A_{1,1} & ... & A_{1,r} \\ ... & ... & ... \\ A_{r,1} & ... & A_{r,r} \end{pmatrix} \tag{55}$$

with square blocks $A_{i,i}, i = 1, ..., r$. Such choice of preconditioner P minimizes the condition number of $P^{-1/2}AP^{-1/2}$ within a factor of r.

- Method of SSOR can be used as a block preconditioner as well. If the original matrix A can be split into diagonal, lower and upper triangular as $A = D + L + L^T$ then the SSOR preconditioner matrix is defined as

$$P = (D + L)D^{-1}(D + L)^T$$

  It can also be parametrised by ω as follows:

$$P(\omega) = \frac{\omega}{2 - \omega} \left(\frac{1}{\omega} D + L \right) D^{-1} \left(\frac{1}{\omega} D + L \right)^T$$

- Incomplete Cholesky factorization with $A = LL^T$ is often used for PCG algorithm. In this case a sparse lower triangular matrix $\tilde{L}$ is chosen to be close to L. Then the preconditioner is defined as $P = \tilde{L}\tilde{L}^T$.
- Incomplete LU preconditioner.