

MATH 612

Computational methods for equation solving and function minimization – Week # 9

F.J.S.

Spring 2014 – University of Delaware

Plan for this week

- Discuss any problems you couldn't solve from previous lectures
- We will cover Lectures 23, 32, and 33
- Homework is due next Monday
- There might be a surprise quiz. Apply the prisoner's paradox at will

Remember that...

... I'll keep on updating, correcting, and modifying the slides until the end of each week.

CHOLESKY

Characterization of PD matrices

A Hermitian positive matrix is a matrix $A \in \mathbb{C}^{m \times m}$ for which

$$A^* = A, \quad \text{and} \quad x^*Ax > 0 \quad \forall 0 \neq x \in \mathbb{C}^m.$$

(Note that for a Hermitian matrix and any vector x , $x^*Ax \in \mathbb{R}$. Why?)

The spectral theorem. A matrix A is Hermitian PD if and only if

$$A = QDQ^*,$$

where Q is unitary and D is diagonal with $d_{ii} > 0$ for all i . In other words, a Hermitian matrix is PD if and only if all its eigenvalues are positive.

Characterization of PD matrices (2)

Consequence. A matrix A is Hermitian PD if and only if

$$A = P^* P, \quad \text{with } P \text{ invertible.}$$

Look at this $A = QDQ^* = (QD^{1/2})(QD^{1/2})^*$. Then look at this
 $x^* P^* P x = (P x)^* (P x) = \|P x\|_2^2$.

Today's goal

We will show that we can take P to be upper triangular with positive diagonal. (This is called a Cholesky factorization.) We will show an algorithm to compute this factorization.

Some questions. How do you call a Hermitian matrix such that $-A$ is PD? How about if you just demand $x^* A x \geq 0$ for all x ? And $x^* A x \leq 0$ for all x ? Have you ever heard of an indefinite matrix?

If A is a Hermitian PD matrix,

- $a_{ii} > 0$ for all i

$$a_{ii} = e_i^* A e_i,$$

- if P is invertible, then PAP^* is Hermitian PD as well

$$x^* PAP^* x = (P^* x)^* A (P^* x) > 0, \quad \text{and} \quad x \neq 0 \implies P^* x \neq 0.$$

- we can apply Gaussian elimination without ever swapping rows and the pivots are always positive (this is a characterization as well) – this third fact is not that simple, we'll need two more slides to prove it.

Hermitian PD matrices and Gaussian elimination

The first step.¹ We know that $a_{11} > 0$. We use the multipliers

$$l_{i1} = \frac{a_{i1}}{a_{11}} = \frac{\overline{a_{1i}}}{a_{11}}.$$

Then

$$\begin{bmatrix} 1 & & & & \\ -l_{21} & 1 & & & \\ \vdots & & \ddots & & \\ -l_{m1} & & & 1 & \end{bmatrix} A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ 0 & b_{22} & \dots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & b_{m2} & \dots & b_{mm} \end{bmatrix}$$

In the next slide we will show (it's really easy though) that the submatrix B is also Hermitian and PD.

¹Don't call this an iteration!

Hermitian PD matrices and Gaussian elimination

This is why!

$$\underbrace{\begin{bmatrix} 1 & & & \\ -l_{21} & 1 & & \\ \vdots & & \ddots & \\ -l_{m1} & & & 1 \end{bmatrix}}_{L_1} AL_1^* = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & b_{22} & \dots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & b_{m2} & \dots & b_{mm} \end{bmatrix}$$

The conjugate multipliers can be used to apply Gaussian elimination by columns. The pivot is the same $a_{11} > 0$ and we need to eliminate $a_{i1} = \overline{a_{1i}}$.

What's left is up to you

Can you see why the $(m-1) \times (m-1)$ matrix is Hermitian and PD?

The search for an algorithm

We have shown that for A Hermitian PD, we can apply Gaussian elimination without ever swapping rows. The pivots are always positive. Therefore

$$A = LU$$

where L is lower triangular with 1s in the diagonal and U is upper triangular with $u_{ij} > 0$.

We can do better:

$$L_{m-1} \dots L_1 A L_1^* \dots L_{m-1}^* = \begin{bmatrix} u_{11} & & \\ & \ddots & \\ & & u_{mm} \end{bmatrix} = D$$

This proves that

$$A = LD^{1/2} D^{1/2} L^* = (LD^{1/2}) \underbrace{(LD^{1/2})^*}_{R}.$$

The search for an algorithm (2)

$$A = LU = \underbrace{LD^{1/2}}_{R^*} \underbrace{D^{-1/2}U}_R$$

Here's our first algorithm (not the best one):

- Apply Gaussian elimination, No row swaps are needed or allowed!
- Make a note of the multipliers (as in the LU factorization)
- Multiply the i -th column by $\sqrt{u_{ij}}$. This can be done right after computing the column.

Another way of looking at this is:

- Do not make a note of the multipliers
- After using the j -th row for elimination, divide it by $\sqrt{u_{jj}}$.

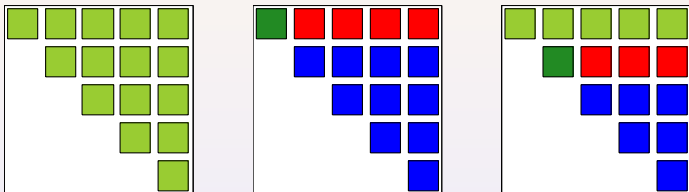
The search for an algorithm (3): bear with me

- In the usual GE process, we need to compute what's under the diagonal, because we need these numbers to compute the multipliers in forthcoming iterations.
- For a Hermitian PD matrix, this is not the case. The smaller blocks to which we apply GE are always Hermitian PD, so everything is above the diagonal
- What to do: forget that the lower part of the matrix (below the diagonal) even exists. Everytime you need one number down there pick it up from the upper part of the matrix. For multipliers, use

$$\frac{\overline{a_{jj}}}{a_{jj}} \quad \text{instead} \quad \frac{a_{jj}}{a_{jj}}.$$

- Save time not dealing with anything below the diagonal.

A picture



Only the upper triangular part of A is ever used. In the first step, the positive **green** pivot and the **red** elements are used to compute the multipliers. All **blue** elements are modified and nothing else. We then move one row down...

Here's the algorithm

```
m=size(A,2);  
R=A;  
for k=1:m  
    for j=k+1:m  
        R(j,j:m)=R(j,j:m)-R(k,j:m)*conj(R(k,j))/R(k,k);  
    end  
    R(k,k:m)=R(k,k:m)/sqrt(R(k,k));  
end
```

An finally...

Once you have $A = R^*R$, to solve $Ax = b$, solve

$$R^*y = b, \quad \text{followed by} \quad Rx = y.$$

Complexity and stability

Work for factorization $\sim \frac{1}{3}m^3$ flops.

If we use the above algorithm to compute the Cholesky factorization and the solve $Ax = b$ using the decomposition, then the complete algorithm is backward stable. The computed solution \tilde{x} satisfies

$$(A + \Delta A)\tilde{x} = b, \quad \frac{\|\Delta A\|}{\|A\|} = O(\epsilon_{\text{machine}}).$$

GOING ITERATIVE

Equation solving so far

- Find a full QR decomposition of A and solve $Rx = Q^*b$.
- Use Gaussian elimination, with or without partial pivoting. At the end solve an upper triangular system.
- (The previous method gives a $PA = LU$ factorization for free. This is practical for future uses of A .)
- For a symmetric PD matrix, find a Cholesky factorization $A = R^*R = LL^*$. Then solve two triangular systems.

All of these methods have complexity of order m^3 . They are very intrusive to the matrix. Few of them take advantage of special structural properties.

Banded matrices

If a matrix A satisfies

$$a_{ij} = 0 \quad \text{whenever } |i - j| > p$$

it is a p -banded matrices. When $p = 0$ this is just a diagonal matrix. When $p = 1$ it is a tridiagonal matrices. The bandwidth is $2p + 1$.

If A is p -banded and **we do not swap rows during Gaussian elimination**², then the resulting upper triangular matrix is banded as well. In fact $A = LU$, where L and U are banded (with the same width).

For the exactly this reason, if A is Hermitian PD and banded, in the Cholesky factorization, R is p -banded as well.

²because we can, and we have decided not to as well


Large sparse matrices

There's no precise definition of sparse matrices³, but here's something that resembles a definition for the case I care about:

A matrix $m \times m$ is sparse when the number of non-zero entries in each row of the matrix is $\mathcal{O}(m^\alpha)$ with $\alpha < 1$.

A typical value in numerical PDEs are $\alpha = 0$, so that the total number of non-zero entries in a matrix is $\mathcal{O}(m)$. The cost of a matrix-vector multiplication is $\mathcal{O}(m^2)$. (Exercise. Compute the exact number of flops.) The cost of a matrix-vector multiplication for a matrix like in the definition is $\mathcal{O}(m^{1+\alpha})$.

And here comes a digression on discretization of the Laplacian... (pay attention!)

³at least no precise definition I know and believe in 

Redefining matrices and changing goals

Instead of the matrix A giving with its entries, we might have been given:

- a sparse representation of A (like in MATLAB's `sparse` construction)
- an operator performing $x \mapsto Ax$.

It is common to assume that we have **an algorithm to multiply by A or by some simple specific parts of A** , like the diagonal of A , A except its diagonal, the upper triangular part of A , etc. Instead of looking for 'exact' solutions (meaning exact in exact arithmetic was possible, approximate because of floating point errors, and stability+conditioning issues), we might be happy enough with \tilde{x} such that

$$\|\tilde{x} - x\| \leq \text{tolerance}.$$

What is an iterative method

In an iterative method for the system $Ax = b$ we build a sequence x_n such that $x_n \rightarrow x = A^{-1}b$.

- Because we do not know the solution, we have to be sure it converges to the right solution (consistency of the method)
- We want to be sure that the method converges.
- We have to figure out when to stop. Typically

$$\frac{\|x_{n+1} - x_n\|}{\|x_{n+1}\|} \leq \text{tolerance}$$

is taken as a stopping criterion. (Do not use the residual for convergence $\|b - Ax_{n+1}\| \leq \text{tolerance}$. The system might be badly conditioned.)

What we want:

- If possible, the method should only use multiplication by (parts of) A and simple non-intrusive manipulation of A .
- If possible, the method should be fast.

CLASSICAL ITERATIVE METHODS

Warning

Today's methods are very old ones. They work for a quite restricted class of matrices and are almost never used to solve the systems, but as parts of larger algorithms.

All the methods today will share a fixed-point structure: to solve $Ax = b$, we iterate

$$Mx_{n+1} = Nx_n + f,$$

where M has to be invertible, and the solution of $Mx = d$ should be easy (M is triangular or diagonal).

The three methods we'll now see need $a_{ii} \neq 0$ for all i . That's a guarantee that we can apply the method, but it's far from being a guarantee for convergence.

Matrix iterations

$$x_{n+1} = Bx_n + c, \quad x_0 \text{ arbitrary.}$$

- If the spectral radius of B (maximum absolute value of its eigenvalues) is less than one, then x_n converges, always to the same limit, independently of the choice of x_0 . (It converges to the unique solution of $(I - B)x = c$. Why?)
- If the spectral radius of B is larger than one (there's an eigenvalue with absolute value larger than one), then x_n DOES NOT converge with probability one (in the choice of x_0).
- If the spectral radius of B is one, there are different situations, and we will not try to characterize them.

Jacobi's discovery of parallelization

Write $Ax = b$ as m separate equations

$$\sum_{j=1}^m a_{ij}x_j = b_i, \quad i = 1, \dots, m$$

isolate the diagonal

$$a_{ii}x_i = b_i - \sum_{j \neq i} a_{ij}x_j, \quad i = 1, \dots, m$$

and iterate in parallel

$$a_{ii}x_i^{(n+1)} = b_i - \sum_{j \neq i} a_{ij}x_j^{(n)}, \quad i = 1, \dots, m$$

Warning

We will use n for the iteration counter and m for the size of the system.

Gauss-Seidel's improved communication skills

In Jacobi's method, the m scalar equations are solved in parallel, and we have to wait for the next iteration to use the computed values. In the GS method we proceed sequentially. Instead of

$$a_{ij}x_i^{(n+1)} = b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(n)} - \sum_{j=i+1}^m a_{ij}x_j^{(n)}, \quad i = 1, \dots, m$$

we work sequentially and update automatically

$$a_{ij}x_i^{(n+1)} = b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(n+1)} - \sum_{j=i+1}^m a_{ij}x_j^{(n)}, \quad i = 1, \dots, m$$

Jacobi vs Gauss-Seidel

Once again, we do not solve systems with these methods. (Okay, there's some very particular cases where these methods are used.) They are used for other purposes.

- Gauss-Seidel communicates updates faster.
- Jacobi is parallel.
- Gauss-Seidel gives a direction to the equations, from first to last. (There's a loss of symmetry. This is solved in the next slide.)
- There are counter examples where each of them is faster than the other.

Symmetric Gauss-Seidel

Sweep down:

$$a_{ij}x_i^{(n+1/2)} = b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(n+1/2)} - \sum_{j=i+1}^m a_{ij}x_j^{(n)}, \quad i = 1, \dots, m$$

Sweep up:

$$a_{ij}x_i^{(n+1)} = b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(n+1/2)} - \sum_{j=i+1}^m a_{ij}x_j^{(n+1)}, \quad i = m, \dots, 1$$

(The $2m$ equations can be reduced to $2m - 2$, because two of them repeat the same computation. Can you see which ones?)

A joint presentation

Write

$$A = D - L - U$$

where $D = \text{diag}(a_{ii})$, L is strictly lower triangular, and U is strictly upper triangular.

Jacobi

$$Dx^{(n+1)} = (L + U)x^{(n)} + b$$

Gauss-Seidel

$$(D - L)x^{(n+1)} = Ux^{(n)} + b$$

Symmetric Gauss-Seidel

$$(D - L)x^{(n+1/2)} = Ux^{(n)} + b$$

$$(D - U)x^{(n+1)} = Lx^{(n+1/2)} + b$$

The symmetric GS method again

$$\begin{aligned}(D - L)x^{(n+1/2)} &= Ux^{(n)} + b, \\ (D - U)x^{(n+1)} &= Lx^{(n+1/2)} + b\end{aligned}$$

$$\begin{aligned}x^{(n+1/2)} &= (D - L)^{-1}Ux^{(n)} + (D - L)^{-1}b, \\ x^{(n+1)} &= (D - U)^{-1}Lx^{(n+1/2)} + (D - U)^{-1}b \\ &= (D - U)^{-1}L(D - L)^{-1}Ux^{(n)} \\ &\quad + (D - U)^{-1}L(D - L)^{-1}b + (D - U)^{-1}b \\ &= Bx^{(n)} + c\end{aligned}$$

ARNOLDI'S METHOD

Krylov spaces: a definition and three properties

Let $b \in \mathbb{C}^m$ and $A \in \mathbb{C}^{m \times m}$. We consider the subspaces

$$\mathcal{K}_n := \langle b, Ab, A^2b, \dots, A^{n-1}b \rangle \quad n \geq 1.$$

Clearly

$$\mathcal{K}_n \subseteq \mathcal{K}_{n+1}, \quad A\mathcal{K}_n \subseteq \mathcal{K}_{n+1}.$$

- 1 If $\mathcal{K}_n = \mathcal{K}_{n+1}$, then $\mathcal{K}_n = \mathcal{K}_{n+1} = \mathcal{K}_{n+2} = \dots$
- 2 If $\mathcal{K}_n = \mathcal{K}_{n+1}$, then $A\mathcal{K}_n \subseteq \mathcal{K}_n$
- 3 If $\mathcal{K}_n = \mathcal{K}_{n+1}$, then $A^{-1}b \in \mathcal{K}_n$.

Arnoldi's method: what it does

Arnoldi's method is the modified Gram-Schmidt method applied to find an orthonormal basis of the Krylov space \mathcal{K}_n .

$$\langle b, Ab, A^2b, \dots, A^{j-1}b \rangle = \langle q_1, \dots, q_j \rangle$$

There are two differences with GS. Because of the particular structure of the Krylov space, the incoming vector in the j -th step is $v_j = A^{j-1}b = Av_{j-1}$. With the Krylov spaces the matrix to which we apply GS does not exist. We build the vectors as we go. However, there's an even more relevant detail in the implementation. It's based in this equality (prove it!)

$$\begin{aligned} \langle b, Ab, A^2b, \dots, A^{j-1}b, A^j b \rangle &= \langle q_1, \dots, q_j, A^j b \rangle \\ &= \langle q_1, \dots, q_j, Aq_j \rangle \end{aligned}$$

Towards Arnoldi

(This was modified GS, in case you have forgotten.)

```
for  $j = 1 : n$   
   $v = a_j$   
  for  $i = 1 : j - 1$   
     $r_{ij} = q_i^* v$   
     $v = v - r_{ij} q_i$   
  end  
   $r_{jj} = \|v\|_2$   
   $q_j = \frac{1}{r_{jj}} v$   
end
```

```
 $v_1 = b$   
for  $j = 1 : n - 1$   
   $v_{j+1} = Av_j$   
end  
for  $j = 1 : n$   
   $v = v_j$   
  for  $i = 1 : j - 1$   
     $r_{ij} = q_i^* v$   
     $v = v - r_{ij} q_i$   
  end  
   $r_{jj} = \|v\|_2$   
   $q_j = \frac{1}{r_{jj}} v$   
end
```

Towards Arnoldi (2) – before and after

```
 $v_1 = b$   
for  $j = 1 : n - 1$   
     $v_{j+1} = Av_j$   
end  
for  $j = 1 : n$   
     $v = v_j$   
    for  $i = 1 : j - 1$   
         $r_{ij} = q_i^* v$   
         $v = v - r_{ij} q_i$   
    end  
     $r_{jj} = \|v\|_2$   
     $q_j = \frac{1}{r_{jj}} v$   
end
```

```
 $v_1 = b$   
for  $j = 1 : n - 1$   
     $v_{j+1} = Av_j$   
end  
 $q_1 = \frac{1}{\|v_1\|_2} v_1$   
for  $j = 1 : n$   
     $v = v_{j+1}$   
    for  $i = 1 : j$   
         $h_{ij} = q_i^* v$   
         $v = v - h_{ij} q_i$   
    end  
     $h_{j+1,j} = \|v\|_2$   
     $q_{j+1} = \frac{1}{h_{j+1,j}} v$   
end
```

Towards Arnoldi (3) – Arnoldi's iteration (!)

```
q1 =  $\frac{1}{\|b\|_2} b$   
for j ≥ 1      % count on the next  
    v = Aqj    % the newcomer  
    for i = 1 : j  
        hij = qi*v    % not R anymore  
        v = v - hijqi  
    end  
    hj+1,j = ||v||2    % not R anymore; stop if zero  
    qj+1 =  $\frac{1}{h_{j+1,j}} v$   
end
```

$$Aq_j = \underbrace{h_{1j}q_1 + h_{2j}q_2 + \dots + h_{jj}q_j}_{\text{orth. proj. onto } \mathcal{K}_j} + h_{j+1,j}q_{j+1}$$

A funny factorization – with a Hessenberg matrix

$$Aq_1 = h_{11}q_1 + h_{21}q_2$$

$$Aq_2 = h_{12}q_1 + h_{22}q_2 + h_{32}q_3$$

$$Aq_3 = h_{13}q_1 + h_{23}q_2 + h_{33}q_3 + h_{43}q_4$$

\vdots

$$A \left[\begin{array}{c|c|c|c} q_1 & q_2 & \dots & q_j \end{array} \right]$$

$$= \left[\begin{array}{c|c|c|c|c} q_1 & q_2 & \dots & q_j & q_{j+1} \end{array} \right]$$

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} & \dots & h_{1j} \\ h_{21} & h_{22} & h_{23} & \dots & h_{2j} \\ & h_{32} & h_{33} & \ddots & h_{3j} \\ & & \ddots & \ddots & \vdots \\ & & & h_{j,j-1} & h_{jj} \\ & & & & h_{j+1,j} \end{bmatrix}$$