Lecture 9

Banded, LU, Cholesky

David Semeraro

University of Illinois at Urbana-Champaign

September 24, 2013

More Algorithms for Special Systems

- tridiagonal systems
- banded systems
- LU decomposition
- Cholesky factorization

Tridiagonal

A tridiagonal matrix A

$$\begin{bmatrix} d_1 & c_1 & & & & & \\ a_1 & d_2 & c_2 & & & & \\ & a_2 & d_3 & c_3 & & & \\ & & \cdots & \cdots & \cdots & & \\ & & & a_{i-1} & d_i & c_i & \\ & & & & \cdots & \cdots & \cdots \\ & & & & \cdots & \cdots & \cdots \\ & & & & & a_{n-1} & d_n \end{bmatrix}$$

- storage is saved by not saving zeros
- only $n + 2(n-1) = 3n - 2$ places are needed to store the matrix versus n^2 for the whole system
- can operations be saved? yes!

Tridiagonal

$$\begin{bmatrix} d_1 & c_1 & & & & & \\ a_1 & d_2 & c_2 & & & & \\ & a_2 & d_3 & c_3 & & & \\ & & \cdots & \cdots & \cdots & & \\ & & & a_{i-1} & d_i & c_i & \\ & & & & \cdots & \cdots & \cdots \\ & & & & \cdots & \cdots & \cdots \\ & & & & & a_{n-1} & d_n \end{bmatrix}$$

Start forward elimination (without any special pivoting)

1. subtract a_1/d_1 times row 1 from row 2
2. this eliminates a_1, changes d_2 and does not touch c_2
3. continuing:

$$\tilde{d}_i = d_i - \left(\frac{a_{i-1}}{\tilde{d}_{i-1}} c_{i-1}\right) \quad \tilde{b}_i = b_i - \left(\frac{a_{i-1}}{\tilde{d}_{i-1}} \tilde{b}_{i-1}\right)$$

for $i = 2 \ldots n$

Tridiagonal

$$\begin{bmatrix} \tilde{d}_1 & c_1 & & & & & \\ & \tilde{d}_2 & c_2 & & & & \\ & & \tilde{d}_3 & c_3 & & & \\ & & & \cdots & \cdots & & \\ & & & & \tilde{d}_i & c_i & \\ & & & & & \cdots & \cdots \\ & & & & & \cdots & \cdots \\ & & & & & & \tilde{d}_n \end{bmatrix}$$

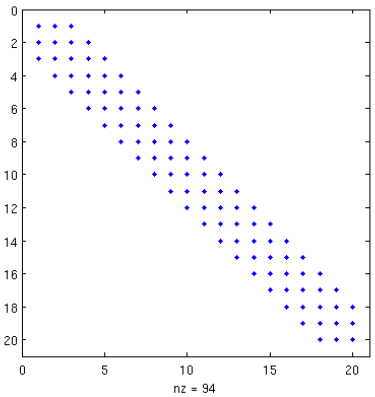
This leaves an upper triangular (2-band). With back substitution:

1. $x_n = \tilde{b}_n / \tilde{d}_n$
2. $x_{n-1} = (1/\tilde{d}_{n-1})(\tilde{b}_{n-1} - c_{n-1}x_n)$
3. $x_i = (1/\tilde{d}_i)(\tilde{b}_i - c_i x_{i+1})$

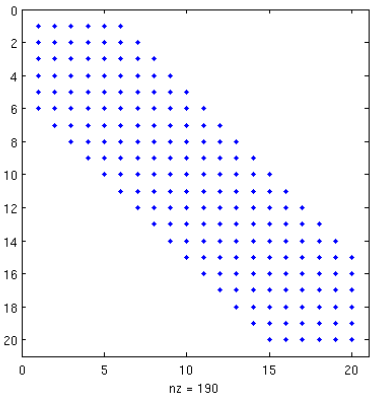
Tridiagonal Algorithm

```
1   input: n, a, d, c, b
2   for i = 2 to n
3       xmult = a_{i-1} / d_{i-1}
4       d_i = d_i - xmult · c_{i-1}
5       b_i = b_i - xmult · b_{i-1}
6   end
7   x_n = b_n / d_n
8   for i = n - 1 down to 1
9       x_i = (b_i - c_i x_{i+1}) / d_i
10  end
```

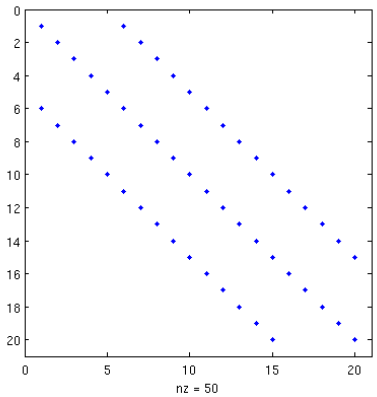
m-band



$m = 5$



$m = 11$

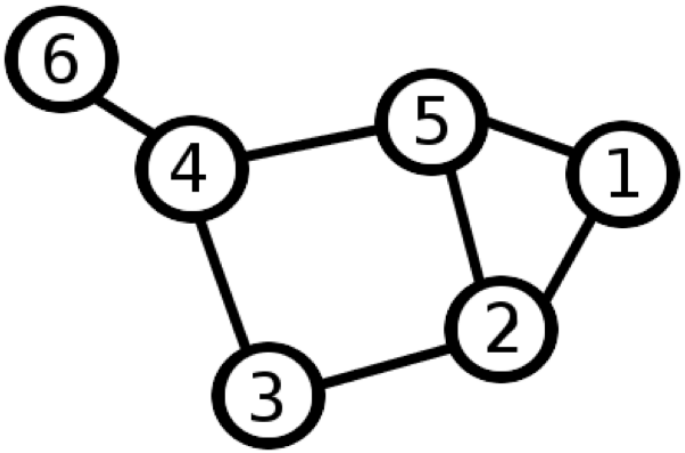


$m = 11$

- the m correspond to the total width of the non-zeros
- after a few passes of GE *fill-in* with occur within the band
- so an empty band costs (about) the same as a non-empty band
- one fix: reordering (e.g. Cuthill-McKee)
- generally GE will cost $\mathcal{O}(m^2 n)$ for m-band systems

Motivation: Graph Theory

- Given a graph, construct associated matrix, called the graph Laplacian
- Row i of matrix corresponds to node i of graph
  - Diagonal entry is valence (total # of edges) for node i
  - Place a negative one in column j if node j is connected to i



$$\Rightarrow \begin{bmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$

Motivation: Graph Theory

Graph is Laplacian useful for

- Calculating spanning trees
- Partitioning a graph evenly
- and many more....

To use the graph Laplacian, you need to solve $Ax = b$ for many different vectors, b.

Motivation: Graph Theory (Multiple Right Hand Sides)

- Solve $Ax = b$ for many different b vectors
- For k different b vectors, Gaussian Elimination costs $\mathcal{O}(kn^3)$
- We can do better: LU factorization

Motivation: Symmetric Matrix

- A is symmetric, if $A = A^T$
- If $A = LU$ and A is symmetric, then could $L = U^T$?
- If so, this could save 50% of the computation of LU by only calculating L
- Save 50% of the FLOPS!
- This is achievable: LDL^T and Cholesky (L^TL) factorization

Factorization Methods

Factorizations are the common approach to solving $Ax = b$:
simply organized Gaussian elimination.

Goals for today:

- LU factorization
- Cholesky factorization
- Python-Numpy functions

LU Factorization

Find L and U such that

$$A = LU$$

and L is lower triangular, and U is upper triangular.

$$L = \begin{bmatrix} 1 & 0 & \cdots & & 0 \\ \ell_{2,1} & 1 & 0 & & 0 \\ \ell_{3,1} & \ell_{3,2} & 1 & & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ \ell_{n,1} & \ell_{n,2} & \cdots & \ell_{n-1,n} & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ 0 & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ 0 & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & & & u_{n-1,n} \\ 0 & 0 & & & u_{n,n} \end{bmatrix}$$

Since L and U are triangular, it is easy to apply their inverses.

Why?

- Since L and U are triangular, it is easy, $\mathcal{O}(n^2)$, to apply their inverses
- Decompose once, solve k right-hand sides quickly:
  - $\mathcal{O}(kn^3)$ with GE
  - $\mathcal{O}(n^3 + kn^2)$ with LU
- Given $A = LU$ you can compute A^{-1}, $det(A)$, $rank(A)$, $ker(A)$, etc...

LU Factorization

Since L and U are triangular, it is easy to apply their inverses.
Consider the solution to $Ax = b$.

$$A = LU \Longrightarrow (LU)x = b$$

Regroup since matrix multiplication is associative

$$L(Ux) = b$$

Let $Ux = y$, then

$$Ly = b$$

Since L is triangular it is easy (without Gaussian elimination) to compute

$$y = L^{-1}b$$

This expression should be interpreted as "Solve $Ly = b$ with forward substitution."

LU Factorization

Now, since y is known, solve for x

$$x = U^{-1}y$$

which is interpreted as "Solve $Ux = y$ with backward substitution."

LU Factorization

Listing 1: LU Solve

```
1  Factor A into L and U
2  Solve Ly = b for y          use forward substitution
3  Solve Ux = y for x          use backward substitution
```

LU Factorization

- If we have $Ax = b$ and perform GE we end up with

$$A = \begin{bmatrix} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \end{bmatrix} \Rightarrow \begin{bmatrix} x' & x' & x' & x' \\ 0 & x' & x' & x' \\ 0 & 0 & x' & x' \\ 0 & 0 & 0 & x' \end{bmatrix}$$

- Remember from Lecture 6, that naive Gaussian Elimination can be done by matrix multiplication

$$MAx = Mb$$

$$Ux = Mb$$

- MA is upper triangular and called U
- M is the elimination matrix

LU Factorization

As an example take one column step of GE, A becomes

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} \rightarrow \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & -12 & 8 & 1 \\ 0 & 2 & 3 & -14 \end{bmatrix}$$

using the elimination matrix

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ -\frac{1}{2} & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

So we have performed

$$M_1 A x = M_1 b$$

LU Factorization

From Lecture 6

- Inverting M_i is easy: just flip the sign of the lower triangular entries

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ -\frac{1}{2} & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \Rightarrow M_1^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}$$

- M_i^{-1} is just the multipliers used in Gaussian Elimination!
- $M_i^{-1} M_j^{-1}$ is still lower triangular, for $i < j$,
  and is the union of the columns
- $M_1^{-1} M_2^{-1} \ldots M_j^{-1}$ is lower triangular, with the lower triangle the multipliers from Gaussian Elimination

LU Factorization

- Zeroing each column yields another elimination matrix operation:

$$M_3 M_2 M_1 A x = M_3 M_2 M_1 b$$

- $M = M_3 M_2 M_1$. Thus
- $L = M_1^{-1} M_2^{-1} M_3^{-1}$ is lower triangular

$$\begin{aligned} MA &= U \\ M_3 M_2 M_1 A &= U \\ A &= M_1^{-1} M_2^{-1} M_3^{-1} U \\ A &= LU \end{aligned}$$

LU (forward elimination) Algorithm

Listing 2: LU

```
given A

for k = 1...n-1
   for i = k+1...n
      xmult = a_ik / a_kk
      a_ik = xmult
      for j = k+1...n
         a_ij = a_ij - (xmult) a_kj
      end
   end
end
```

- U is stored in the upper triangular portion of A
- L (without the diagonal) is stored in the lower triangular

Doolittle Factorization (LU)

Listing 3: Doolittle

```
1  given A
2  output L, U
3
4  for k = 1...n
5     ℓ_kk = 1
6     for j = k...n
7           u_kj = a_kj − Σ_{i=1}^{k−1} ℓ_ki u_ij
8     end
9     for j = k+1...n
10          ℓ_jk = (a_jk − Σ_{i=1}^{k−1} ℓ_ji u_ik) / u_kk
11    end
12 end
```

- Mathematically the same as previous LU
- Difference is we now explicitly form L and U

What About Pivoting?

- Pivoting (that is row exchanges) can be expressed in terms of matrix multiplication
- Do pivoting during elimination, but track row exchanges in order to express pivoting with matrix P
- Let P be all zeros
  - Place a 1 in column j of row 1 to exchange row 1 and row j
  - If no row exchanged needed, place a 1 in column 1 of row 1
  - *Repeat for all rows of* P
- P is a permutation matrix
- Now using pivoting,

$$LU = PA$$

NUMPY LU

Like GE, LU needs pivoting. With pivoting the LU factorization always exists, even if A is singular. With pivoting, we get

$$LU = PA$$

```
1  import pprint
2  import scipy
3  import scipy.linalg   # SciPy Linear Algebra Library
4
5  A = scipy.array([ [5, 4, 6, 9], [4, 4, 1, 4], [1, 7, 1, 10],
       [9, 8, 9, 3] ])
6  P, L, U = scipy.linalg.lu(A)
7
8  print "A:"
9  pprint.pprint(A)
10
11 print "P:"
12 pprint.pprint(P)
13
14 print "L:"
15 pprint.pprint(L)
16
17 print "U:"
18 pprint.pprint(U)
```

LU Tutorial Module

http://www.cse.illinois.edu/iem/linear_equations/gaussian_elimination/

Use SYMMETRY ! YRTEMMYS esU

- Suppose
$$A = LU, \text{ and } A = A^T$$
- Then
$$LU = A = A^T = (LU)^T = U^T L^T$$
- Thus
$$U = L^{-1} U^T L^T$$
and
$$U(L^T)^{-1} = L^{-1} U^T = D$$
- We can conclude that
$$U = DL^T$$
and
$$A = LU = LDL^T$$

Symmetric Doolittle Factorization (LDL^T)

Listing 4: Symm Doolittle

```
given A
output L, D

for k = 1...n
   ℓ_kk = 1

   d_k = a_kk − Σ_{ν=1}^{k−1} d_ν ℓ_kν^2

   for j = k+1...n
        ℓ_kj = 0
        ℓ_jk = ( a_jk − Σ_{ν=1}^{k−1} ℓ_jν d_ν ℓ_kν ) / d_k
   end
end
```

- Special form of the Doolittle factorization

LL^T: Cholesky Factorization

- A must be symmetric and positive definite (SPD)
- A is Positive Definite (PD) if for all $x \neq 0$ the following holds

$$x^T A x > 0$$

- Positive definite gives us an all positive D in $A = LDL^T$
  - Let $x = L^{-1}e_i$, where e_i is the i-th column of I
- L becomes $LD^{1/2}$
- $A = LL^T$, i.e. $L = U^T$
  - Half as many flops as LU!
  - Only calculate L not U

Cholesky Factorization

Listing 5: Cholesky

```
1  given A
2  output L
3
4  for k = 1...n
5     ℓ_kk = ( a_kk − Σ_{i=1}^{k−1} ℓ_ki^2 )^{1/2}
6
7     for j = k+1...n
8        ℓ_jk = ( a_jk − Σ_{i=1}^{k−1} ℓ_ji ℓ_ki ) / ℓ_kk
9     end
10 end
```

Why SPD?

In general, SPD gives us

- non singular
  - If $x^T Ax > 0$, for all nonzero x
  - Then $Ax \neq 0$ for all nonzero x
  - Hence, the columns of A are linearly independent
- No pivoting
  - From algorithm, can derive that $|l_{kj}| \leqslant \sqrt{a_{kk}}$
  - Elements of L do not grow with respect to A
  - *For short proof see book*
- Cholesky faster than LU
  - No pivoting
  - Only calculate L, not U

Why SPD?

A matrix is Positive Definite (PD) if for all $x \neq 0$ the following holds

$$x^T Ax > 0$$

- For SPD matrices, use the Cholesky factorization, $A = LL^T$
- Cholesky Factorization
  - Requires no pivoting
  - Requires one half as many flops as LU factorization, that is only calculate L not L and U.
  - Cholesky will be more than *twice* as fast as LU because no pivoting means no data movement
- Use SCIPY's built-in `scipy.linalg.cholesky()` function for routine work

Motivation Revisited

Multiple right hand sides

- Solve $Ax = b$ for k different b vectors
- Using LU factorization, the cost is $\mathcal{O}(n^3) + \mathcal{O}(kn^2)$
- Using Gaussian Elimination, the cost is $\mathcal{O}(kn^3)$

If A is symmetric

- Save 50% of the flops with LDL^T factorization
- Save 50% of the flops and many memory operations with Cholesky (L^TL) factorization