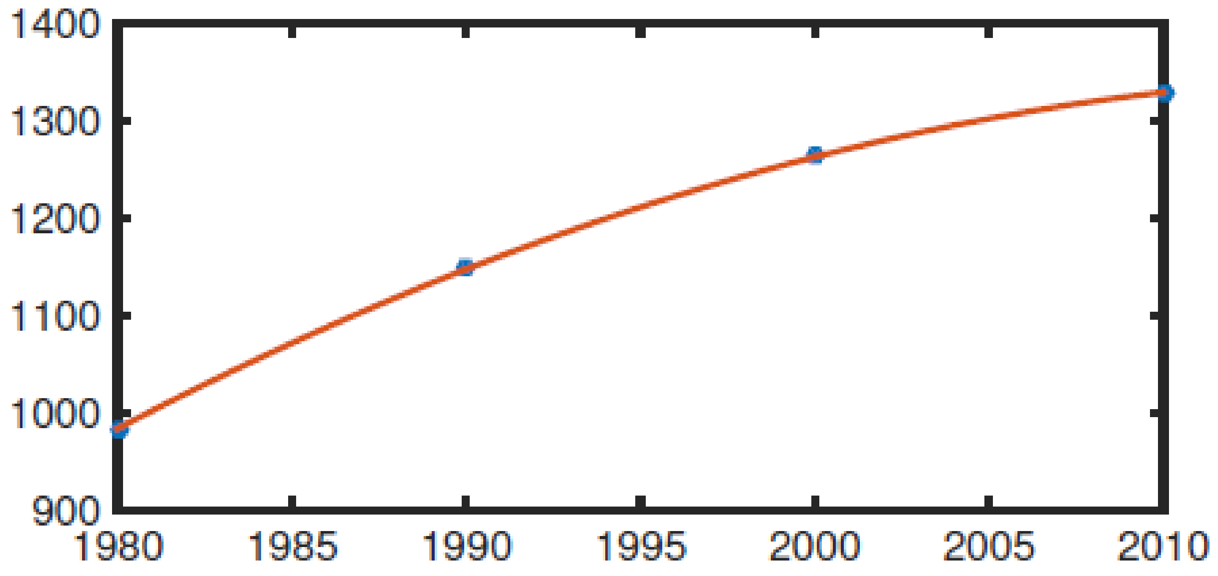
Chapter 2

Square linear systems: $Ax = b$



Comparing function sizes in a limit

- We want to compare how functions behave in limiting cases
- Let $f(n) = \tan(1/n)$, $g(n) = 1/n$ and $h(n) = 1/n^2$
- How do these compare to each other as $n \to \infty$, or "large n"
- The limit is easy: $\lim_{n\to\infty} f(n) = \tan(0) = 0$
- What we want to know is how fast this function gets to the limit. We can do that by comparing f to g and h
- With g, $\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty} \tan\left(\frac{1}{n}\right) / \left(\frac{1}{n}\right) = 1$
- We conclude that f and g are the same size…

Comparing functions: Order and asymptotic

- We say that "f is asymptotic to g" or $f \sim g$
- These two approach the limit at the same rate
- We can also say "f is order g" or $f = O(g)$ when the limit is bounded
- With h, $\lim_{n\to\infty} \frac{f(n)}{h(n)} = \lim_{n\to\infty} [\tan\left(\frac{1}{n}\right)]/\left(\frac{1}{n^2}\right) = \lim_{n\to\infty} n = \infty$
- The limit doesn't exist, because h goes to zero faster than f
- So, f and h are not the same order.
- If we define $k(n) = \tan^2(\frac{1}{n})$, we find $k \sim h$

Comparing functions: Order and asymptotic

- The dominant part of a growing function can be identified using this approach
- Let $f(n) = a_1 n^2 + b_1 n + c_1$ and $g(n) \ = \ n$^2
- Then $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} a_1 + b_1 n^{-1} + c_1 n^{-2} = a_1$
- We say that "f is asymptotic to n^2" or $f \sim n^2$
- We'll use this to evaluate operation counts and performance

Operation Counts: Example

- Matlab example multiplies a matrix and a vector, then adds a vector
- Count *,/,+,- as same operation
- Neglect storage
- Inner loop: Multiplying one row of A with x takes n * and n-1 +; adding vector adds 1 +
- Outer loop: adding that up for n rows gives final result of $2n^2$

```
n = 6;
A = magic(n);
x = ones(n,1);
y = zeros(n,1);
for i = 1:n
    for j = 1:n
        y(i) = y(i) + A(i,j)*x(j);
    end
end
```

$$\sum_{i=1}^{n}\sum_{j=1}^{n} 2 = \sum_{i=1}^{n} 2n = 2n^2.$$

Operation Counts: Matlab

- Let’s test Matlab matrix-vector multiplication
- Time using stopwatch functions tic and toc
- Repeat to get more reliable timing

```
fprintf('    n      time (sec)\n')
fprintf('
fprintf('\n\n')
```

```
t_ = [];
n_ = 400:400:4000;
for n = n_
    A = randn(n,n);   x = randn(n,1);
    tic
    for j = 1:10
        A*x;
    end
    t = toc;
    t_ = [t_,t/10];
end
```

```
   n      time (sec)
  400      1.47e-04
  800      7.49e-04
 1200      1.80e-03
 1600      1.88e-03
 2000      2.69e-03
 2400      4.14e-03
 2800      5.29e-03
 3200      4.81e-03
 3600      7.75e-03
 4000      7.40e-03
```

How fast is the time growing?

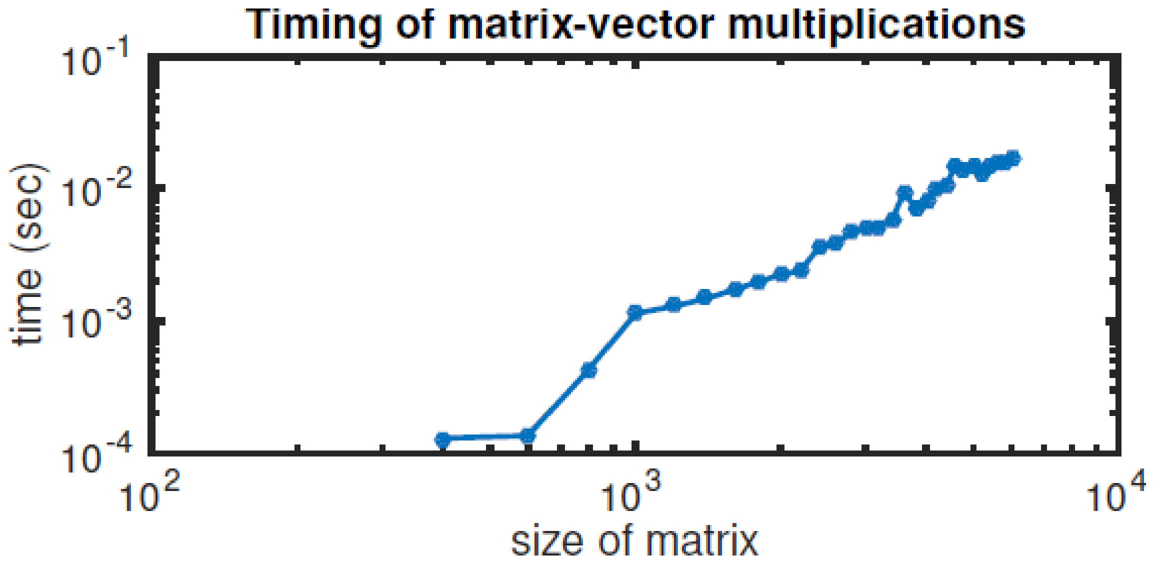
Operation Counts: Matlab

- Matlab matrix-vector multiplication
- How to identify rate? Expecting power law: use log-log plot.

$$t = Cn^p \qquad \Longrightarrow \qquad \log t = p(\log n) + (\log C).$$

- If we get a line, the slope is p

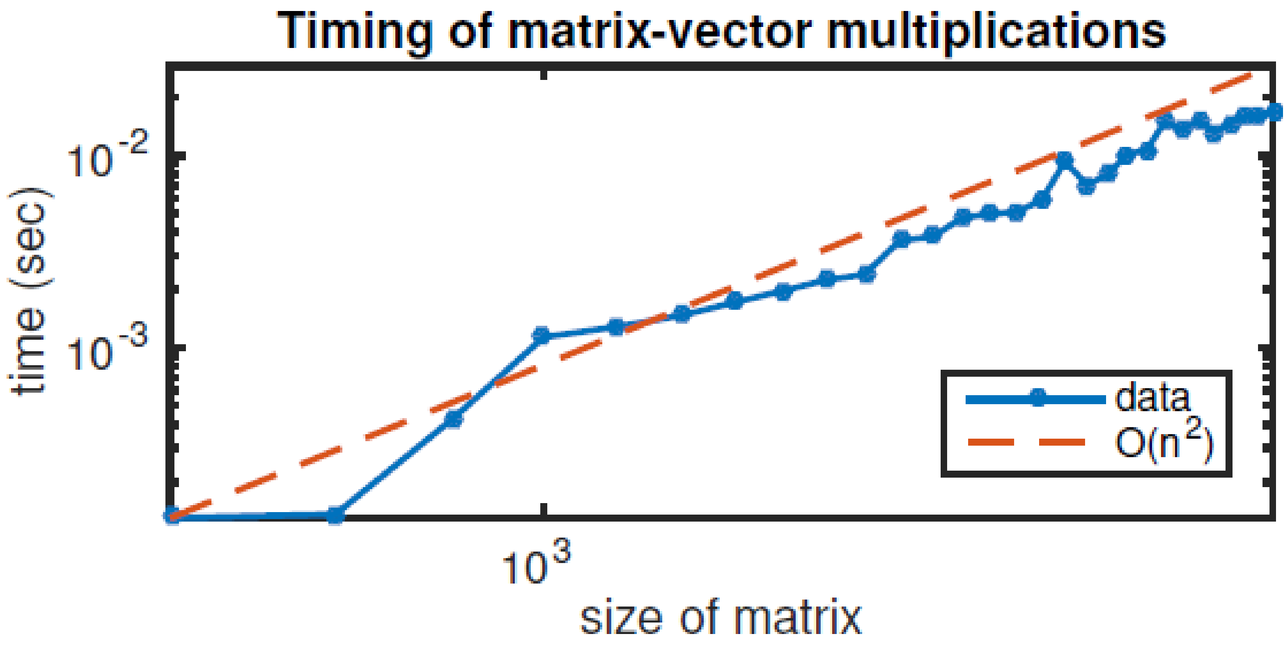
```
loglog(n_,t_,'.-')
xlabel('size of matrix'), ylabel('time (sec)')
title('Timing of matrix-vector multiplications')
```



Operation Counts: Matlab

- To see the slope one could plot a known function
- Try Cn^2 here because of theory
- Pick C to put line in a convenient place, or use t_1/n_1^2

```
hold on, loglog(n_,t_(1)*(n_/n_(1)).^2,'--')
axis tight
legend('data','O(n^2)','location','southeast')
```



Operation Count: LU

- While some time is needed, no FLOPs here, so neglected
- Inside nested loops, one FLOP here
- Multiply part of row (j to n, for n-j+1 elements) and subtract from same part
- For operations inside both loops total is then
$$1 + 2(n - j + 1)$$
$$= 2(n - j) + 3$$
- Line 19 ignored
- Now total over loops

```
8   n = length(A);
9   L = eye(n);    % ones on diagonal
10
11  % Gaussian elimination
12  for j = 1:n-1
13    for i = j+1:n
14      L(i,j) = A(i,j) / A(j,j);    % row multiplier
15      A(i,j:n) = A(i,j:n) - L(i,j)*A(j,j:n);
16    end
17  end
18
19  U = triu(A);
```

Operation Count: LU

- Inner loop: i ranges from $j+1$ to n (rows below diag)
- Outer loop: j ranges from 1 to n (all rows)

$$\sum_{j=1}^{n-1}\sum_{i=j+1}^{n}[2(n-j)+3]$$

```
n = length(A);
L = eye(n);    % ones on diagonal

% Gaussian elimination
for j = 1:n-1
  for i = j+1:n
    L(i,j) = A(i,j) / A(j,j);    % row multiplier
    A(i,j:n) = A(i,j:n) - L(i,j)*A(j,j:n);
  end
end

U = triu(A);
```

Note how the index for the outer sum j is in the limit of the inner sum: We must do the inner sum first to get all the j's into the summand

Operation Count: LU

- Do the inner sum:

$$\sum_{j=1}^{n-1} \sum_{i=j+1}^{n} [2(n-j)+3] = \sum_{j=1}^{n-1} (n-j)[2(n-j)+3]$$

- Make it easier: change variable with k = n-j.
- When $j = 1, k = n - 1$; when $j = n - 1, k = 1$. Then we have

$$\sum_{k=1}^{n-1} k(2k+3)$$

- Distributing the sum, we have sum involving k and a sum involving k^2

Operation Count: LU

- For most situations, we only care about the leading factor in the sum, and can use the results at right

$$\sum_{k=1}^{n-1} k(2k+3)$$

- One term is proportional to n^3, from k^2
- The other is proportional to n^2, from k
- Our sum then results in two parts:

$$\frac{2n^3}{3} + \frac{3n^2}{2}$$

- For large n, the cubic term is dominant: LU factorization is $O(n^3)$, and more

$$\sum_{k=1}^{n} k \sim \frac{n^2}{2} = O(n^2), \text{ as } n \to \infty,$$

$$\sum_{k=1}^{n} k^2 \sim \frac{n^3}{3} = O(n^3), \text{ as } n \to \infty,$$

$$\vdots$$

$$\sum_{k=1}^{n} k^p \sim \frac{n^{p+1}}{p+1} = O(n^{p+1}), \text{ as } n \to \infty,$$

Operation Count: LU

- Our sum then results in two parts:

$$\frac{2n^3}{3} + \frac{3n^2}{2}$$

- For large n, the cubic term is dominant: LU factorization is $O(n^3)$
- More specifically, the FLOP count is asymptotic to

$$\frac{2n^3}{3}$$

- How does this stack up against actual computation time?

Operation Counts: LU

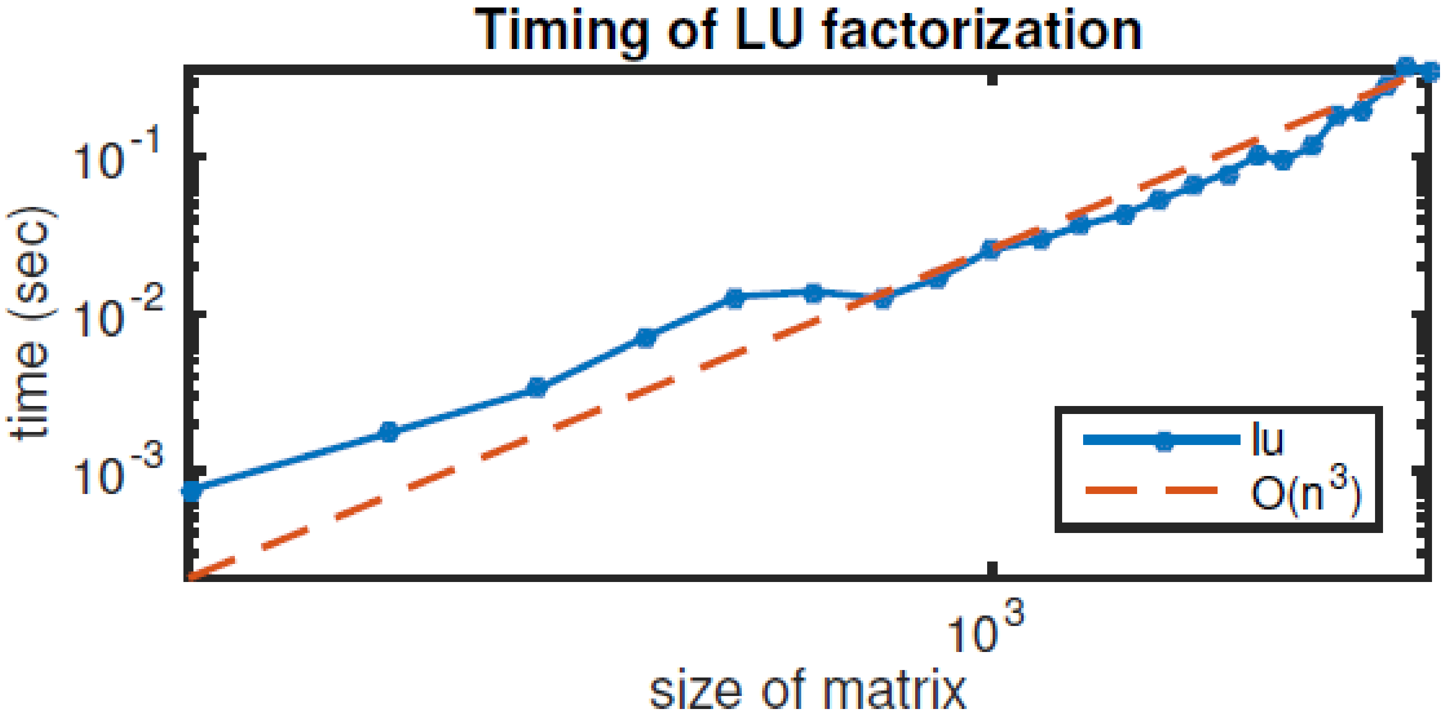
- Let's test with Matlab
- Use functions tic and toc: wall clock time
- Increase matrix size n
- Repeat to get more reliable timing

```
t_ = [];
n_ = 200:100:2400;
for n = n_
    A = randn(n,n);
    tic
    for j = 1:6,  [L,U] = lu(A);  end
    t = toc;
    t_ = [t_,t/6];
end
```

How fast is the time growing with n?

Operation Counts: Matlab

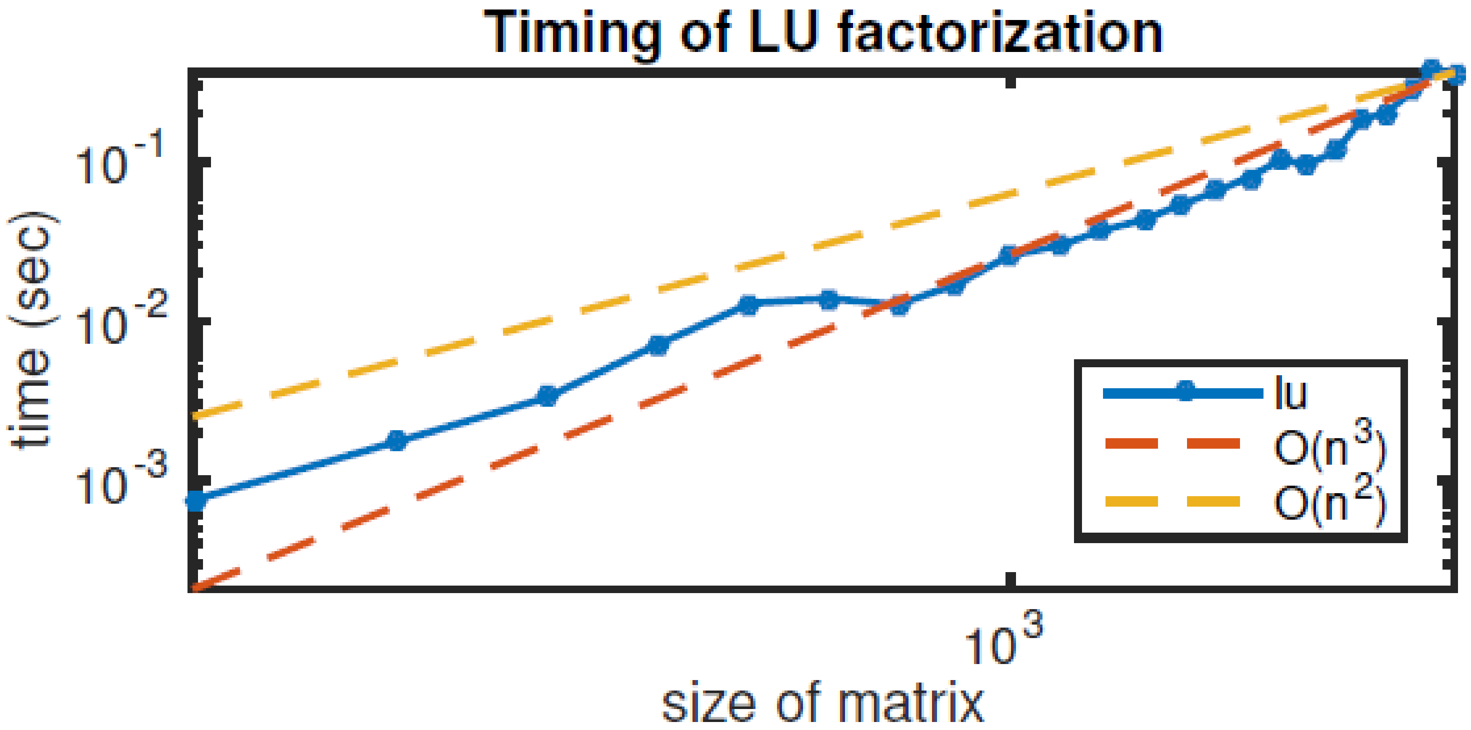
```
loglog(n_,t_,'.-')
hold on, loglog(n_,t_(end)*(n_/n_(end)).^3,'--')
axis tight
xlabel('size of matrix'), ylabel('time (sec)')
title('Timing of LU factorization')
legend('lu','O(n^3)','location','southeast')
```



- Note that we picked a convenient constant
- Is this behaving like n^3?
- It's not perfect fit by any means

Operation Counts: Matlab

```
hold on, loglog(n_,t_(end)*(n_/n_(end)).^2,'--')
legend('lu','O(n^3)','O(n^2)','location','southeast')
```



- Let's add an n^2 curve
- It looks like it is between n^2 and n^3
- For larger matrix size, seems closer to n^3
- What is contributing to this?

Operation Count: LU

- We assumed all that mattered was time to do one flop and that they were sequential
- CPUs can have multiple cores, and can have vectorized operations
- These can violate our assumptions
- Parallel computation is even more different: time to send data to different CPUs can even dominate the computation
- The details of the specific hardware matter
- If computational time is important to your project, test it!!!!

Fixing up issues with naïve GE

- To carry out GE, we needed to compute the multiplier
$$L(i+1,i) = A(i+1,i)/A(i,i).$$
- What if A(i,i)=0?
- Switch rows so that there is nonzero element there: “row pivoting”
- Smart way to do that: move the largest element of A(i+1:n,i), the part of column i below the pivot, to the pivot.
- Consider this example…

Our previous example

```
A = [2 0 4 3 ; -4 5 -7 -10 ; 1 15 2 -4.5 ; -2 0 2 -13]
b = [ 4; 9; 29; 40 ]
```

```
A =
    2.0000         0    4.0000    3.0000
   -4.0000    5.0000   -7.0000  -10.0000
    1.0000   15.0000    2.0000   -4.5000
   -2.0000         0    2.0000  -13.0000
b =
     4
     9
    29
    40
```

```
[L,U] = lufact(A);
x = backsub( U, forwardsub(L,b) )
```

```
x =
    -3
     1
     4
    -2
```

- No difficulties here
- But, after finishing with the first column, we had $A_{24} = 0$
- What if that were at (2,2) location instead?

Our previous example

- We can use the following to change the order of the equations with $R\_2 \leftrightarrow R\_4$

```
A([2 4],:) = A([4 2],:);  b([2 4]) = b([4 2]);
```

- Theoretically, the answer does not change, and the \ gets it right
- But, lufact fails!

```
[L,U] = lufact(A);
L
```

```
L =
    1.0000         0         0         0
   -1.0000    1.0000         0         0
    0.5000       Inf    1.0000         0
   -2.0000       Inf       NaN    1.0000
```

Our previous example

- We can use the following to change the order of the equations with $R\_2 \leftrightarrow R\_4$

```
A([2 4],:) = A([4 2],:);  b([2 4]) = b([4 2]);
```

- Theoretically, the answer does not change, and the \ gets it right
- But, lufact fails!

```
[L,U] = lufact(A);
L
```

```
L =
    1.0000         0         0         0
   -1.0000    1.0000         0         0
    0.5000       Inf    1.0000         0
   -2.0000       Inf       NaN    1.0000
```

$$\begin{bmatrix} 2 & 0 & 4 & 3 \\ 0 & 0 & 6 & -10 \\ 0 & 15 & 0 & -6 \\ 0 & 5 & 1 & -4 \end{bmatrix}$$

- Why? Zero pivot after first column finished

Fixing up issues with naïve GE

- The fact that we get a zero pivot can be fixed by switching rows, IF the column below the pivot is not all zero
- If the A(i:n,i)=0, then the columns 1:i are linearly dependent, and there is no unique solution
- This implies that the original matrix **A** is singular
- Theorem 2: If a pivot element and all the elements below it are zero, then the original matrix **A** is singular. In other words, if **A** is nonsingular, then Gaussian elimination with row pivoting will run to completion.
- This tells us that using row pivoting is well worth implementing

Our previous example

- How to swap rows?
- Use a "permutation matrix"
- If we take the identity matrix I that is 3x3, and switch the first two rows, we get

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- The effect of left multiplying by this P is to switch rows 1 and 2! In terms

$$\boldsymbol{PB} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 3 & 1 \\ 6 & 5 & 4 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 6 & 5 & 4 \\ 2 & 3 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

- We could also look at ***P*** as $\boldsymbol{P} = [\boldsymbol{e}_2 \quad \boldsymbol{e}_1 \quad \boldsymbol{e}_3]$

Permutations and row switches

- Permutation matrices have interesting properties
- Say left multiplying by P switches some rows; left multiplying by P^T will switch them back.
- This suggests that $P^T = P^{-1}$!! (Proof in exercises)
- Let's use them to keep track of row switches.
- For our example where we got the zero pivot, we had

$$\begin{bmatrix} 2 & 0 & 4 & 3 \\ 0 & 0 & 6 & -10 \\ 0 & 15 & 0 & -6 \\ 0 & 5 & 1 & -4 \end{bmatrix}$$

- We can switch rows 2 and 3 to keep going with GE
- Write this in matrix terms…

Row switches in LU

- After finishing with the first column and doing row switch, we have
$$PA_1 = P_1 L_{41} L_{31} L_{21} A$$
- Finishing up the factorization gives
$$U = L_{43} L_{42} L_{32} P_1 L_{41} L_{31} L_{21} A$$
- Working toward undoing the right side, we get
$$P_1^T L_{32}^{-1} L_{42}^{-1} L_{43}^{-1} U = L_{41} L_{31} L_{21} A$$
- Continuing, we get
$$L_{21}^{-1} L_{31}^{-1} L_{41}^{-1} P_1^T L_{32}^{-1} L_{42}^{-1} L_{43}^{-1} U = A$$
- The way that we use this is to post facto create P so that
$$LU = PA$$
- Or $P^T LU = A$

Row switches in LU: MATLAB

- We will use MATLAB's builtin function lu
- The syntax is [L,U,P] = lu(A)
- To use this, we first view the system at Pax=Pb
- Then use L and U as before, because row switching won't be needed now
- Thus, PA = LU and let Ux = z.
- Then, Lz = Pb
- Solving the system can then be done by:

  using forwardsub on Lz=Pb to get z,

  then using backsub on Ux=z to get x

Partial pivoting in MATLAB

- Solving systems using lu in MATLAB and our functions is then as follows:

1. Find L, U and P from [L,U,P] = lu(A);
2. Solve Lz=Pb using z = forwardsub(L,P*b);
3. Solve Ux=z using x = backsub(U,z).

- Using MATLAB’s lu and \, one does:

1. Find L, U and P from [L,U,P] = lu(A);
2. Solve Lz=Pb using z = L\(P*b);
3. Solve Ux=z using x = U\z.

Example using LU

```
A = [ 2 0 4 3; -2 0 2 -13 ; 1 15 2 -4.5 ; -4 5 -7 -10 ];
b = [ 4; 40; 29; 9 ];
```

- Swapped row system at left

Example using LU

```
A = [ 2 0 4 3; -2 0 2 -13 ; 1 15 2 -4.5 ; -4 5 -7 -10 ];
b = [ 4; 40; 29; 9 ];

[L,U,P] = lu(A)
```

- Swapped row system at left
- lu gives L,U,P

```
L =
    1.0000         0         0         0
   -0.2500    1.0000         0         0
    0.5000   -0.1538    1.0000         0
   -0.5000    0.1538    0.0833    1.0000
U =
   -4.0000    5.0000   -7.0000  -10.0000
         0   16.2500    0.2500   -7.0000
         0         0    5.5385   -9.0769
         0         0         0   -0.1667
P =
     0     0     0     1
     0     0     1     0
     0     1     0     0
     1     0     0     0
```

Example using LU

```
A = [ 2 0 4 3; -2 0 2 -13 ; 1 15 2 -4.5 ; -4 5 -7 -10 ];
b = [ 4; 40; 29; 9 ];

[L,U,P] = lu(A)
```

```
L =
    1.0000         0         0         0
   -0.2500    1.0000         0         0
    0.5000   -0.1538    1.0000         0
   -0.5000    0.1538    0.0833    1.0000
U =
   -4.0000    5.0000   -7.0000  -10.0000
         0   16.2500    0.2500   -7.0000
         0         0    5.5385   -9.0769
         0         0         0   -0.1667
P =
     0     0     0     1
     0     0     1     0
     0     1     0     0
     1     0     0     0
```

- Swapped row system at left
- lu gives L,U,P
- In one line, forwardsub, then backsub

```
x = backsub( U, forwardsub(L,P*b) )
```

```
x =
   -3.0000
    1.0000
    4.0000
   -2.0000
```

Example using PtLU

```
[PtL,U] = lu(A)
```

```
PtL =
   -0.5000    0.1538    0.0833    1.0000
    0.5000   -0.1538    1.0000         0
   -0.2500    1.0000         0         0
    1.0000         0         0         0
U =
   -4.0000    5.0000   -7.0000  -10.0000
         0   16.2500    0.2500   -7.0000
         0         0    5.5385   -9.0769
         0         0         0   -0.1667
```

```
x = U \ (PtL\b)
```

```
x =
   -3.0000
    1.0000
    4.0000
   -2.0000
```

- Use only two output arguments
- lu gives PtL and U
- The backslash is better than forwardsub and can use the row-switched L (i.e, PtL)
- Using the backslash can solve the two system sequence in one line

Example using only MATLAB's \

```
x = A\b
```

```
x =
    -3
     1
     4
    -2
```

```
x =
   -3.0000
    1.0000
    4.0000
   -2.0000
```

- Using \ solves the system directly
- Any row switching is transparent
- Compares well with previous solution

What about small pivots

- We have row swapped when the pivot was zero
- However, bad things happen when pivots are small ($\epsilon \ll 1$).
- The system at left has a small pivot at (1,1); do GE on the augmented matrix
- If we don't switch it out, then we get large multiplier and result at right
- If ε is small enough, then possible subtractive cancellation
- How to avoid?

$$\mathbf{A} = \begin{bmatrix} -\epsilon & 1 \\ 1 & -1 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 1-\epsilon \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} -\epsilon & 1 & 1-\epsilon \\ 0 & -1+\epsilon^{-1} & \epsilon^{-1}-1 \end{bmatrix} \Rightarrow \begin{aligned} x_2 &= 1 \\ x_1 &= \frac{(1-\epsilon)-1}{-\epsilon} \end{aligned}$$

What about small pivots

$$\mathbf{A} = \begin{bmatrix} -\epsilon & 1 \\ 1 & -1 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 1-\epsilon \\ 0 \end{bmatrix}$$

- Now try with rows swapped
- This time, multiplier is small, and we get a very good answer

$$\begin{bmatrix} 1 & -1 & 0 \\ 0 & 1-\epsilon & 1-\epsilon \end{bmatrix} \Rightarrow \begin{aligned} x_2 &= 1 \\ x_1 &= \frac{0-(-1)}{1} \end{aligned}$$

- No loss of significance this way
- This suggests a strategy: When working with row i, find the biggest element in column i from the diagonal and below, and put that element in the pivot
- That is max A(i:n,i) (say row p) becomes the pivot ($R_p \leftrightarrow R_i$)
- Then, multipliers are never bigger than 1, so this avoids the large multiplier problem

Vector Norms

- We need to be able to measure the size of a vector or matrix, so that we can order them
- We do this with norms, which can be thought of as functions that map $\mathbb{R}^n \mapsto \mathbb{R}$
- A norm of a vector $||v||$ must have the following properties:

1. $\|\mathrm{x}\| \geq 0$ for all $\mathrm{x} \in \mathbb{R}^n$
2. $\|\mathrm{x}\| = 0$ if and only if $\mathrm{x} = 0$
3. $\|\alpha \mathrm{x}\| = |\alpha| \, \|\mathrm{x}\|$ for any $\mathrm{x} \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$
4. $\|\mathrm{x} + \mathrm{y}\| \leq \|\mathrm{x}\| + \|\mathrm{y}\|$ for all $\mathrm{x}, \mathrm{y} \in \mathbb{R}^n$ (the triangle inequality)

Norms of vectors

- The general vector norm of interest is the *p*-norm:

$$\|v\|_p = \left[\sum_{i=0}^{n} |v_i|^p\right]^{1/p}$$

- We are most interested in only three values of *p:* 1,2, or ∞

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} |x_i|^2\right)^{\frac{1}{2}} = \sqrt{\mathbf{x}^T\mathbf{x}}$$

$$\|\mathbf{x}\|_\infty = \max_{i=1,\ldots,n} |x_i|$$

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$$

Norms of vectors

- Examples:

$$u = [1 \;\; -5 \;\; 3], \qquad \mathrm{v} = \begin{bmatrix} -2 \\ 2 \\ 0 \end{bmatrix}$$

- Shape of vector doesn't matter
- 1-norm uses p=1, so that

$$\|u\|_1 = |1| + |-5| + |3| = 9, \quad \|v\|_1 = |-2| + |2| + 0 = 4$$

- For the 2-norm

$$\|u\|_2 = (|1|^2 + |-5|^2 + |3|^2)^{1/2} = \sqrt{35},$$

$$\|v\|_2 = (|-2|^2 + |2|^2 + 0)^{1/2} = \sqrt{8} = 2\sqrt{2}$$

Norms of vectors

- Examples:

$$u = [1 \ -5 \ \ 3], \qquad \mathrm{v} = \begin{bmatrix} -2 \\ 2 \\ 0 \end{bmatrix}$$

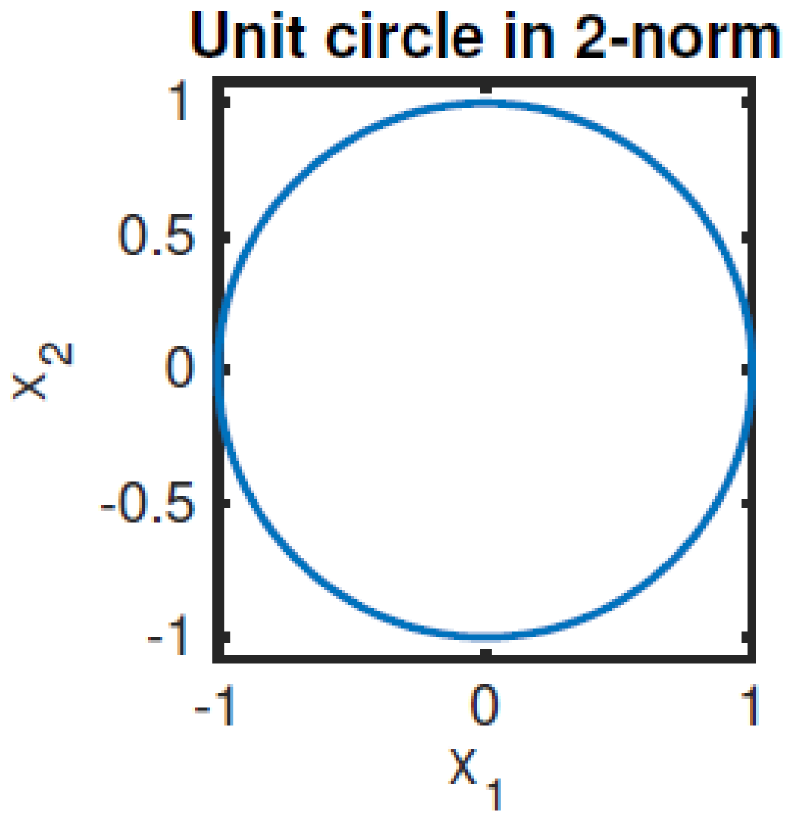
- ∞-norm now:

$$||u||_\infty = \max(|1|, |-5|, |3|) = 5$$

- Multiply by a scalar?
- Always ≥ 0?
- Only 0 if **0** vector?

Norms of vectors

- What do unit vectors look like?
- Given $\boldsymbol{x} = [x_1\ x_2]$, we can create the unit vector or direction $\boldsymbol{x}/\|\boldsymbol{x}\|$
- For $\boldsymbol{x} = [x_1\ x_2]$ and $\|\boldsymbol{x}\|_2$, unit vectors have $\|\boldsymbol{x}\|_2 = (|x_1|^2 + |x_2|^2)^{1/2} = 1$
- In the (x_1, x_2)-plane, the set of all unit vector is a circle
- What about ∞-norm?
- $\|\boldsymbol{x}\|_\infty = \max(|x_1|, |x_2|) = 1$; what is it?



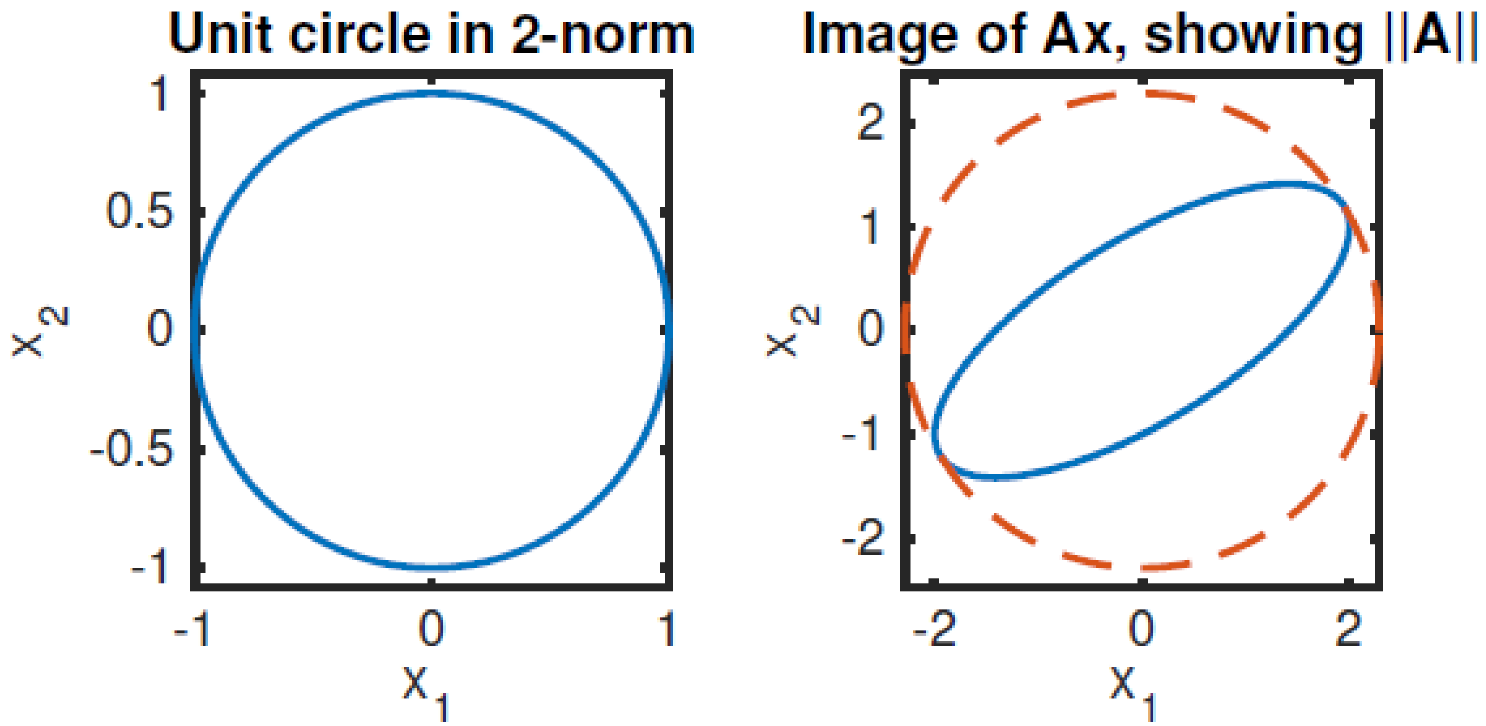
Norms of matrices

- Measuring the “size” of a matrix should be associated with a vector norm: Induced matrix norms
- For $\boldsymbol{x} = [x_1\ x_2]$ and $\|\boldsymbol{x}\|_2$, the induced matrix norm is $\|\boldsymbol{A}\|_2 = \max\limits_{\|\boldsymbol{x}\|_2=1}(\|\boldsymbol{Ax}\|_2) = \max\limits_{\|\boldsymbol{x}\|_2\neq 0}(\|\boldsymbol{Ax}\|_2/\|\boldsymbol{x}\|_2)$
- When we multiply ***x*** by a matrix, then it is stretched and pointed in a different direction, generally
- The magnitude of the biggest stretching is the norm of $\boldsymbol{A}$

Norms of matrices

- We can multiply the set of unit vectors by A and see what the distortion does to all of those unit vectors
- The magnitude of the biggest stretching is the norm of ***A***

```
subplot(1,2,2), plot(Ax(1,:),Ax(2,:)), axis equal
hold on, plot(twonorm*x(1,:),twonorm*x(2,:),'--')
title('Image of Ax, showing ||A||')
xlabel('x_1'),  ylabel('x_2')
```



Norms of matrices

- The induced norms for the other matrices are a bit easier to compute
- The ∞-norm is the maximum row sum of the matrix A
- The 1-norm is the maximum column sum of A
- Mnemonic: think of the "direction" of 1 or ∞
- Example: $\boldsymbol{A} = \begin{bmatrix} 1 & 3 \\ -5 & 8 \end{bmatrix}$
- $\|\boldsymbol{A}\|_1 = \max(1+5, 3+8) = 11$
- $\|\boldsymbol{A}\|_\infty = \max(1+3, 5+8) = 13$

Norms of matrices

- We need some properties of matrix norms for future use.
- One can prove the following for any of the norms we have used.

For any $n \times n$ matrix $\mathbf{A}$ and induced matrix norm,

$$\begin{aligned}
\|\mathbf{A}\mathbf{x}\| &\leq \|\mathbf{A}\| \cdot \|\mathbf{x}\|, && \textit{for any } \mathbf{x} \in \mathbb{R}^n, \\
\|\mathbf{A}\mathbf{B}\| &\leq \|\mathbf{A}\| \cdot \|\mathbf{B}\|, && \textit{for any } \mathbf{B} \in \mathbb{R}^{n \times n}, \\
\|\mathbf{A}^k\| &\leq \|\mathbf{A}\|^k, && \textit{for any integer } k \geq 0.
\end{aligned}$$

Stability analysis of solving systems

- We want to solve $\boldsymbol{Ax} = \boldsymbol{b}$
- We want to analyze how robust our answers are to perturbations of $\boldsymbol{A}$ and $\boldsymbol{b}$
- To measure what happens to the sizes of vectors and matrices, use norms
- We know that $\left||\boldsymbol{b}|\right| = ||\boldsymbol{Ax}|| \leq ||\boldsymbol{A}||\ ||\boldsymbol{x}||$
- We also know that $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$, now take norms
- $||\boldsymbol{x}|| = ||\boldsymbol{A}^{-1}\boldsymbol{b}|| \leq \left||\boldsymbol{A}^{-1}|\right|\ ||\boldsymbol{b}||$
- If we change $\boldsymbol{b}$ to $\boldsymbol{b} + \boldsymbol{d}$, then the solution changes somewhat to $\boldsymbol{x} + \boldsymbol{h}$, say, such that $\boldsymbol{A}(\boldsymbol{x} + \boldsymbol{h}) = \boldsymbol{b} + \boldsymbol{d}$
- But, using the original equation $\boldsymbol{Ax} = \boldsymbol{b},$ the first term on each side cancels

Stability analysis of solving systems

- We then have $\boldsymbol{Ah} = \boldsymbol{d}$
- Then $\boldsymbol{h} = \boldsymbol{A}^{-1}\boldsymbol{d}$, now take norms
- $||\boldsymbol{h}|| = ||\boldsymbol{A}^{-1}\boldsymbol{d}|| \leq \left|\left|\boldsymbol{A}^{-1}\right|\right| \, ||\boldsymbol{d}||$
- Let's now look at the relative change in the solution $\left|\left|\boldsymbol{h}\right|\right| / ||\boldsymbol{x}||$ compared to the relative change in the right hand side $\left|\left|\boldsymbol{d}\right|\right| / ||\boldsymbol{b}||$
- Taking the ratio of those two,
- The last quantity is

the condition number

$\kappa(\boldsymbol{A}) = ||\boldsymbol{A}^{-1}|| \, ||\boldsymbol{A}\,||$

$$\frac{\frac{\|\mathbf{h}\|}{\|\mathbf{x}\|}}{\frac{\|\mathbf{d}\|}{\|\mathbf{b}\|}} = \frac{\|\mathbf{h}\| \cdot \|\mathbf{b}\|}{\|\mathbf{d}\| \cdot \|\mathbf{x}\|} \leq \frac{\left(\|\mathbf{A}^{-1}\| \, \|\mathbf{d}\|\right)\left(\|\mathbf{A}\| \, \|\mathbf{x}\|\right)}{\|\mathbf{d}\| \, \|\mathbf{x}\|} = \|\mathbf{A}^{-1}\| \, \|\mathbf{A}\|$$

Stability analysis of solving systems

The condition number
$\kappa(A) = ||A^{-1}||||A||$
thus tells us the worst case magnification of the relative change in the answer

compared to

the relative change in the right hand side

$$\frac{\frac{\|\mathbf{h}\|}{\|\mathbf{x}\|}}{\frac{\|\mathbf{d}\|}{\|\mathbf{b}\|}} = \frac{\|\mathbf{h}\| \cdot \|\mathbf{b}\|}{\|\mathbf{d}\| \cdot \|\mathbf{x}\|} \le \frac{\left(\|\mathbf{A}^{-1}\| \, \|\mathbf{d}\|\right)\left(\|\mathbf{A}\| \, \|\mathbf{x}\|\right)}{\|\mathbf{d}\| \, \|\mathbf{x}\|} = \|\mathbf{A}^{-1}\| \, \|\mathbf{A}\|$$

The condition number depends on the matrix A and the norm, and it measures the sensitivity of its solutions to perturbations

Stability analysis of solving systems

- What if we perturb A and solve $\boldsymbol{Ax} = \boldsymbol{b}$?
- If we change $\boldsymbol{A}$ to $\boldsymbol{A} + \boldsymbol{E}$, then the solution changes somewhat to $\boldsymbol{x} + \boldsymbol{h}$, say, such that $(\boldsymbol{A} + \boldsymbol{E})(\boldsymbol{x} + \boldsymbol{h}) = \boldsymbol{b}$
- Expand again to get $\boldsymbol{Ax} + \boldsymbol{Ex} + \boldsymbol{Ah} + \boldsymbol{Eh} = \boldsymbol{b}$
- Then use $\boldsymbol{Ax} = \boldsymbol{b}$**,** and neglect $\boldsymbol{Eh}$ because it is the product of two small terms; then one has $\boldsymbol{Ex} + \boldsymbol{Ah} = \boldsymbol{0}$
- Solve for $\boldsymbol{h}$ and use norms:

$$\mathbf{h} = -\mathbf{A}^{-1}\mathbf{E}\mathbf{x}$$

$$\|\mathbf{h}\| \leq \|\mathbf{A}^{-1}\|\,\|\mathbf{E}\|\,\|\mathbf{x}\|$$

- Now compute relative changes again…

Stability analysis of solving systems

The condition number

$\kappa(\boldsymbol{A}) = ||\boldsymbol{A}^{-1}||||\boldsymbol{A}\,||$

thus tells us the worst case magnification of the relative change in the answer

compared to

the relative change in the matrix ***A***

$$\frac{\|\mathbf{h}\|\,\|\mathbf{A}\|}{\|\mathbf{E}\|\,\|\mathbf{x}\|} \leq \frac{\left(\|\mathbf{A}^{-1}\|\,\|\mathbf{E}\|\,\|\mathbf{x}\|\right)\|\mathbf{A}\|}{\|\mathbf{E}\|\,\|\mathbf{x}\|} = \kappa(\mathbf{A}).$$

The condition number measures the sensitivity of its solutions to perturbations in the matrix as well

Condition number

- In summary, the worst case changes in the answer from perturbations to the coefficients is given by the condition number:

$$\frac{\|\Delta \mathbf{x}\|}{\|\mathbf{x}\|} \le \kappa(\mathbf{A}) \frac{\|\Delta \mathbf{b}\|}{\|\mathbf{b}\|}$$

$$\frac{\|\Delta \mathbf{x}\|}{\|\mathbf{x}\|} \le \kappa(\mathbf{A}) \frac{\|\Delta \mathbf{A}\|}{\|\mathbf{A}\|}$$

- Note that the best we can get is a condition number of unity:

$$1 = \|\mathbf{I}\| = \|\mathbf{A}\mathbf{A}^{-1}\| \le \|\mathbf{A}\| \, \|\mathbf{A}^{-1}\| = \kappa(\mathbf{A})$$

Condition number: example

- The $n \times n$ Hilbert matrix has elements
$$H_{ij} = 1/(i + j - 1)$$
- It's a builtin function in MATLAB:

```
>> hilb(5)
ans =
    1.0000    0.5000    0.3333    0.2500    0.2000
    0.5000    0.3333    0.2500    0.2000    0.1667
    0.3333    0.2500    0.2000    0.1667    0.1429
    0.2500    0.2000    0.1667    0.1429    0.1250
    0.2000    0.1667    0.1429    0.1250    0.1111
```

- It is not singular, but conditioning?

```
>> det(hilb(5))
ans =
   3.7493e-12
>> cond(hilb(5))
ans =
   4.7661e+05
```

Condition number: example

- Explore the Hilbert matrix for solving sytems
- Increase n gets even worse conditioning fast.
- The “gallery” has a collection of commonly used matrices in numerical methods and analysis.
- Try “help gallery” at prompt or search for gallery in help browser
- You can find other ill-conditioned, non-singular matrices there: e.g., dorr,

Residuals and ill-conditioned systems

- What happens to the residual $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$?
- For the computed solution (tilde) we have $\mathbf{r} = \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}.$
- Manipulate into the useful form

$$\mathbf{r} = \mathbf{A}(\mathbf{A}^{-1}\mathbf{b} - \tilde{\mathbf{x}}) = \mathbf{A}(\mathbf{x} - \tilde{\mathbf{x}}),$$

- Then use solve for change in solution and use norms: $\|\mathbf{x} - \tilde{\mathbf{x}}\| \leq \|\mathbf{A}^{-1}\| \, \|\mathbf{r}\|.$
- Now compute relative changes again:

$$\frac{\|\mathbf{x} - \tilde{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \kappa(\mathbf{A}) \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}$$

- We can expect *small residual*, but…

the *change in the answer may not be small* for ill-conditioned systems!!!

- Commercial FEM, e.g., tells you *residuals:* use with care!!!

Special matrices

- We can take advantage of the structure of some kinds of matrices to create faster algorithms or get additional information.
- Possibilities include:
  - Triangular matrices: Been there, reduces solves for n unknowns from $O(n^3)$ to $O(n^2)$
  - Banded matrices: only certain diagonals have non-zero elements
  - Sparse matrices: only a (preferably small) minority of elements are nonzero
  - Symmetric matrices
  - Symmetric positive definite matrices

Banded matrices

- A tridiagonal matrix has three nonzero diagonals: e.g.,

```
A = gallery('tridiag',c,d,e)
```

- This matrix would have vectors `c` in the first subdiagonal, `d` on the main diagonal, and `e` on the first superdiagonal
- The bandwidth is three. How to compute?
- Upper bandwidth is p, with $A_{ij} = 0$ if $j - i > p$
- Lower bandwidth is q, with $A_{ij} = 0$ if $i - j > q$
- Total bandwidth is $p + q - 1$
- For this example, we'd get 3, with $p = 2$ and $q = 2$

Banded matrices

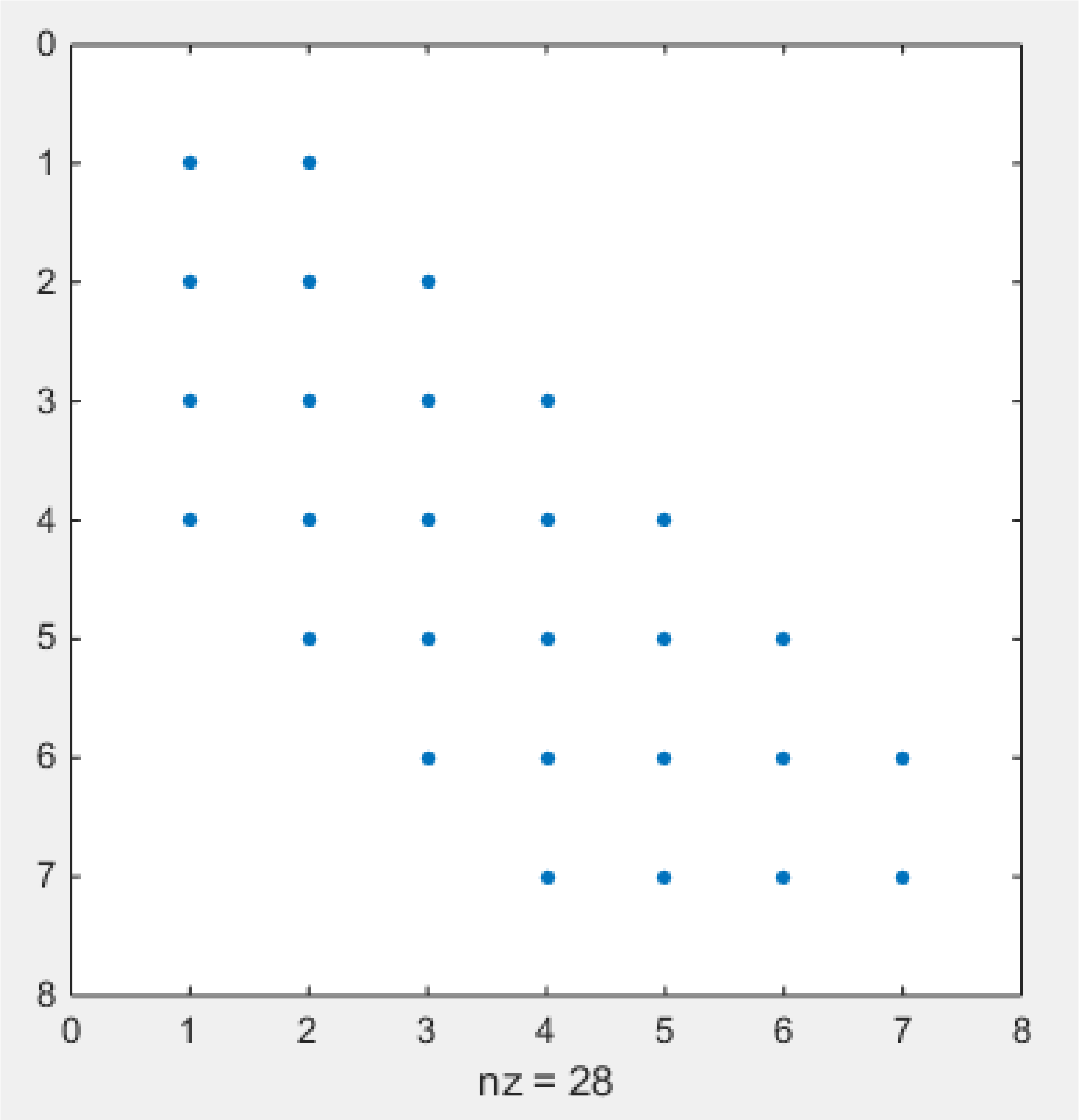
- Consider

```
>> triu(tril(rand(7),1),-3)

ans =

    0.4465    0.3725         0         0         0         0         0
    0.6463    0.5932    0.4067         0         0         0         0
    0.5212    0.8726    0.6669    0.9880         0         0         0
    0.3723    0.9335    0.9337    0.8641    0.5583         0         0
         0    0.6685    0.8110    0.3889    0.5989    0.8825         0
         0         0    0.4845    0.4547    0.1489    0.2850    0.2834
         0         0         0    0.2467    0.8997    0.6732    0.8962
```

- Upper bandwidth is $p = 1$, lower bandwidth is $q = 3$
- Total bandwidth is $p + q + 1 = 5$

Banded matrices

- We can visualize nonzero elements with sparsity plot from spy(triu(tril(rand(7),1,-3))
- Dots are shown where the nonzero elements were
- The total number of nonzeros is at the bottom
- Handy to see where nonzeros are in large matrices



Banded matrices

- How do we get diagonals?
- How can we build banded matrices
- What happens with LU factorization?
- Start with example at right
- We can extract one diagonal at a time with the diag command

```
A =
     2    -1     0     0     0     0
     4     2    -1     0     0     0
     0     3     0    -1     0     0
     0     0     2     2    -1     0
     0     0     0     1     1    -1
     0     0     0     0     0     2

diag_main = diag(A,0)'

diag_main =
     2     2     0     2     1     2

diag_plusone = diag(A,1)'

diag_plusone =
    -1    -1    -1    -1    -1

diag_minusone = diag(A,-1)'

diag_minusone =
     4     3     2     1     0
```

Banded matrices

- How do we get diagonals?
- How can we build banded matrices?
- What happens with LU factorization?
- Start with example at right
- We can extract one diagonal at a time with the diag command
- We can also create the matrix with the diag command:

```
A =
     2    -1     0     0     0     0
     4     2    -1     0     0     0
     0     3     0    -1     0     0
     0     0     2     2    -1     0
     0     0     0     1     1    -1
     0     0     0     0     0     2
```

```
>> c = [4 3 2 1 0];
>> d = [2 2 0 2 1 2];
>> e = [-1 -1 -1 -1 -1];
>> B = diag(c,-1)+diag(d,0)+diag(e,1);
>> norm(A-B,2)
ans =
     0
```

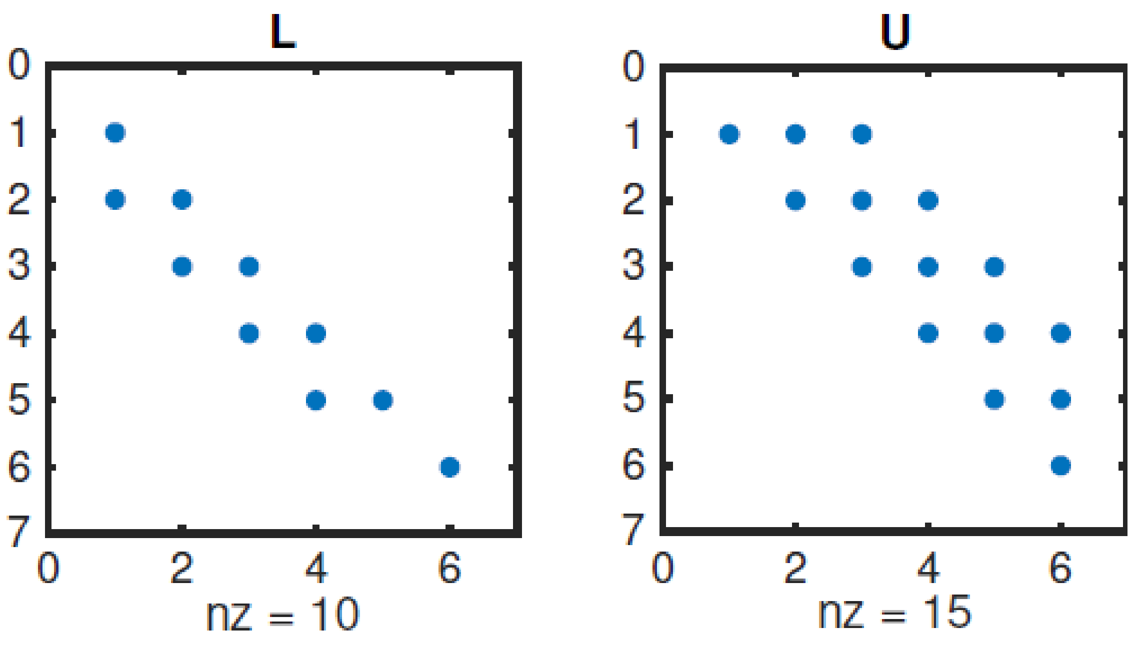
Banded matrices

- What happens with LU factorization?
- Modify A with extra diag
- Do LU factorization without partial pivoting
- The banded structure is preserved:

```
A = A + diag([5 8 6 7],2)
```

```
A =
     2    -1     5     0     0     0
     4     2    -1     8     0     0
     0     3     0    -1     6     0
     0     0     2     2    -1     7
     0     0     0     1     1    -1
     0     0     0     0     0     2
```

```
[L,U] = lufact(A);
subplot(1,2,1), spy(L), title('L')
subplot(1,2,2), spy(U), title('U')
```



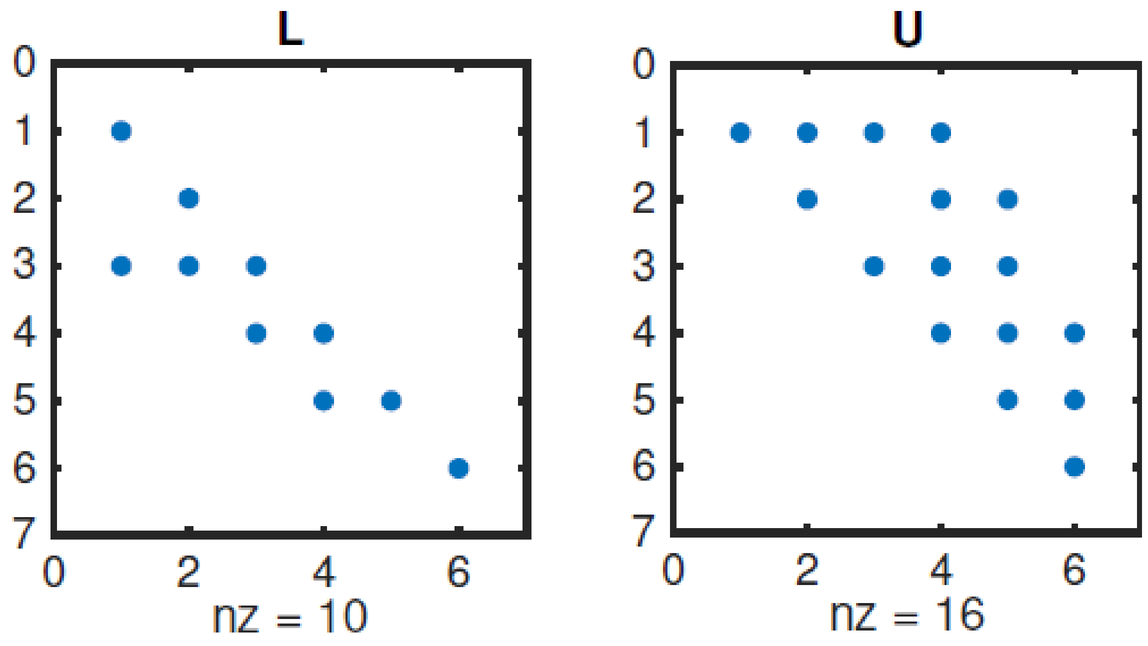
Banded matrices

- What happens with LU factorization?
- Modify A with extra diag
- Now do LU factorization with partial pivoting
- The banded structure is not preserved:
- We can improve the computing time by telling matlab that the matrices may be sparse (mostly zeros)

```
A = A + diag([5 8 6 7],2)
```

```
A =
     2    -1     5     0     0     0
     4     2    -1     8     0     0
     0     3     0    -1     6     0
     0     0     2     2    -1     7
     0     0     0     1     1    -1
     0     0     0     0     0     2
```

```
[L,U,P] = lu(A);
subplot(1,2,1), spy(L), title('L')
subplot(1,2,2), spy(U), title('U')
```



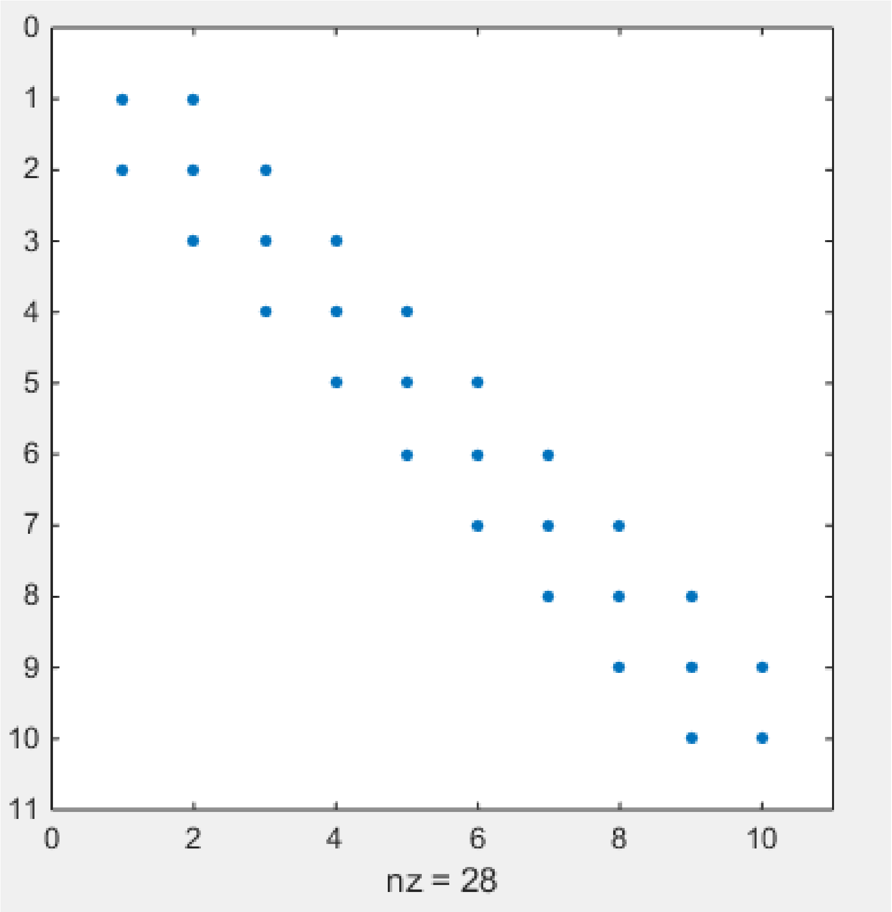
Sparse matrices

- Consider tridiagonal matrices

```
A = gallery('tridiag',n)
```

- This creates a $n \times n$ tridiagonal matrix with 2 on the main diagonal and -1 on the sub- and super-diagonals
- But, only the non-zeros are stored (let n=10 and try it!) Thus, only 28 numbers are stored for this element that would have 100 elements: 28%
- If $n = 100$, then nz=298, out of 10^4 possible elements: 2.98% nonzero
- In this case, $3n - 2$ elements nonzero out of n^2 possible; more sparse an n increases

```
>> A = gallery('tridiag',10);
>> spy(A)
```



Sparse matrices

- Computations can be sped up significantly if we only work with the nonzeros (need to code it)

```
n = 8000;
A = diag(1:n) + diag(n-1:-1:1,1) + diag(ones(n-1,1),-1);

tic, [L,U] = lu(A); toc
```

```
Elapsed time is 5.850532 seconds.
```

```
tic, [L,U] = lu(sparse(A)); toc
```

```
Elapsed time is 0.182472 seconds.
```

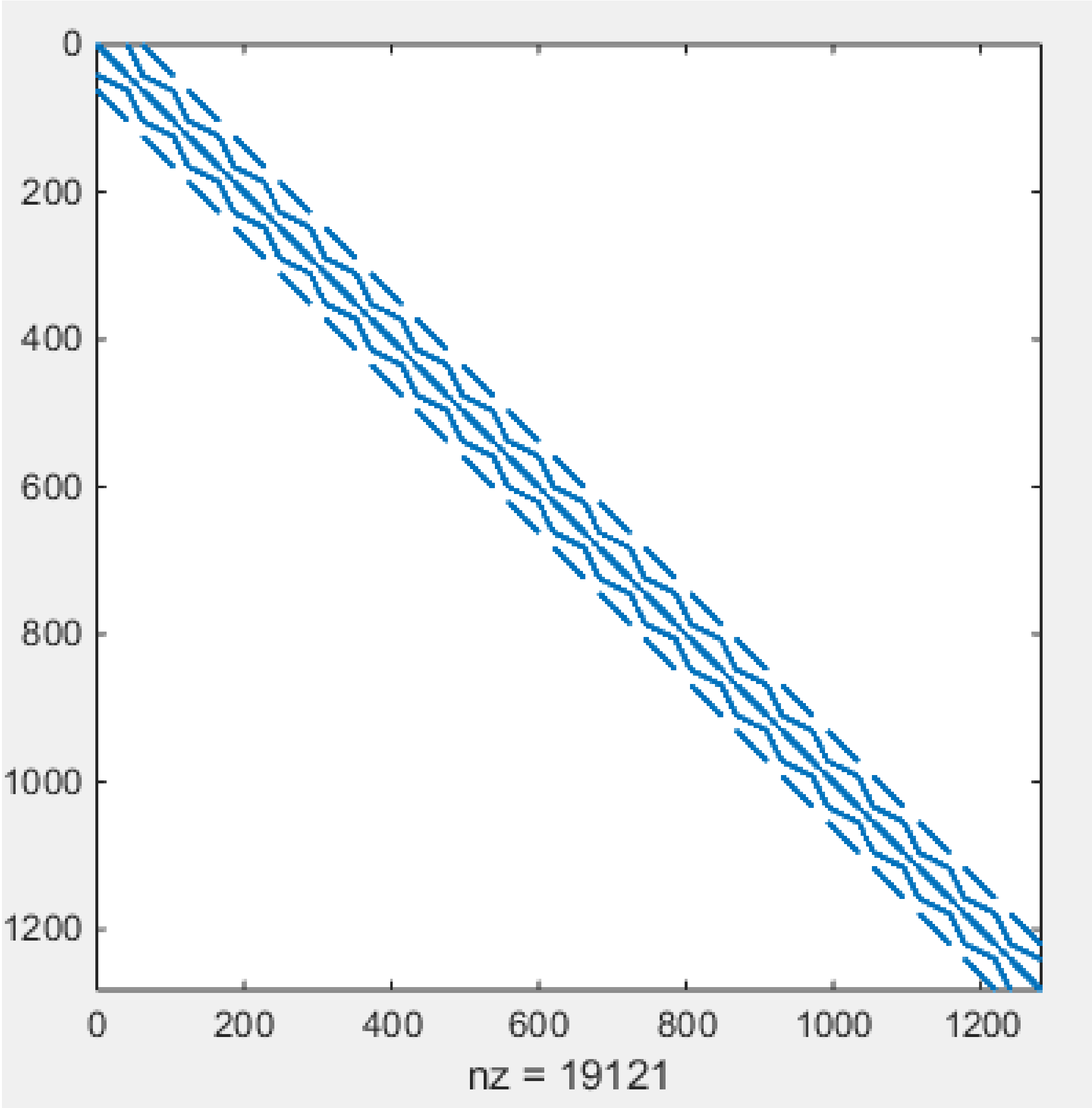
- If A is sparse, full(A) will make it a full matrix

Sparse matrices

- Consider tridiagonal matrices

A = gallery(‘wathen',n,n)

- This creates a large sparse matrix; if $n = 20$, the matrix is 1281×1281 with
- But, things can go wrong: LU factorization leads to fill-in
- In this example, about 1.17% nonzeros in A, but look how many nonzeros after LU…

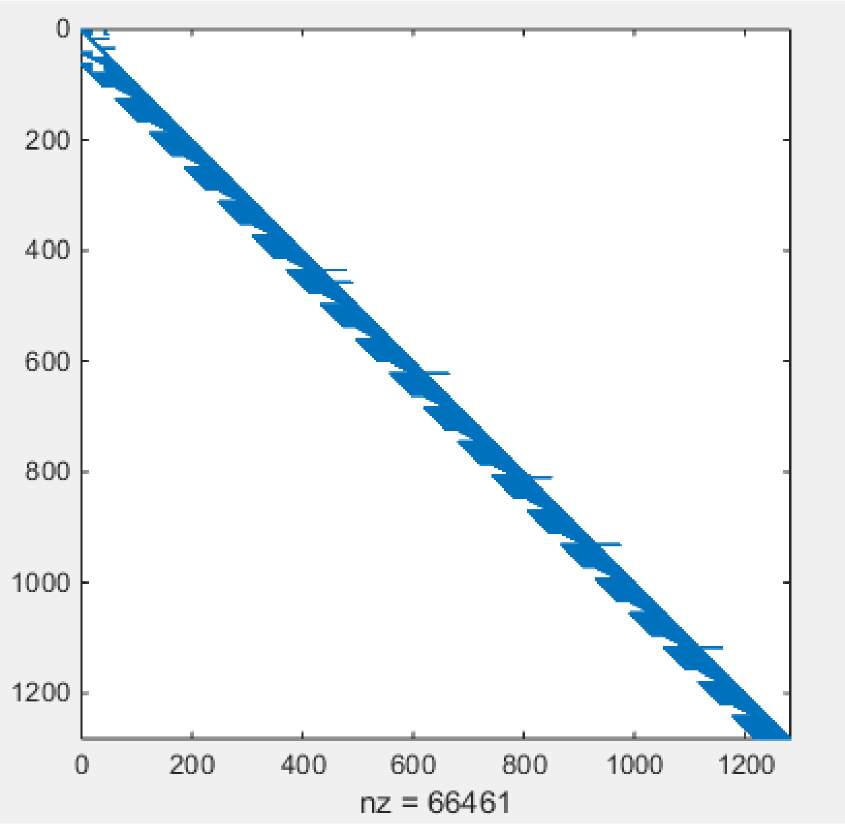


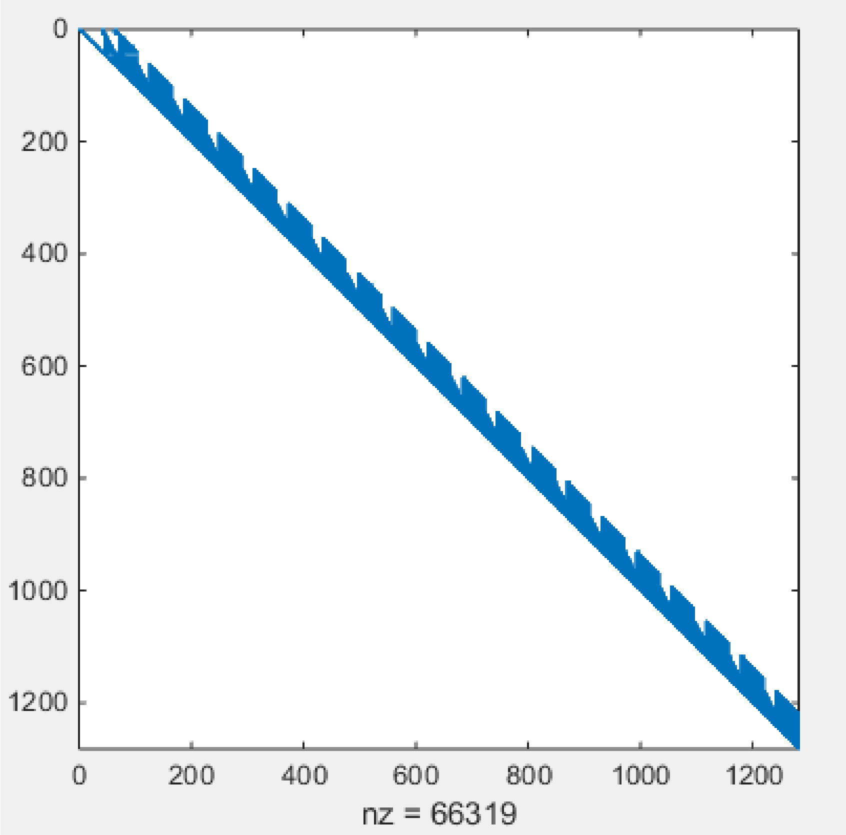
Sparse matrices

- The commands:

```
>> A=gallery('wathen',20,20);
>> spy(A)
>> [L,U]=lu(A);
>> spy(L)
>> spy(U)
```

- Each factor ***L*** (left) and ***U*** (right) has about 6.6e4 nonzeros now! (10x more nonzeros)





Symmetric matrices

- If $\boldsymbol{A}^T = \boldsymbol{A}$, then $\boldsymbol{A}$ is symmetric
- We can modify the LU factorization:

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U} = \boldsymbol{L}\boldsymbol{I}\boldsymbol{U} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{D}^{-1}\boldsymbol{U}$$

- $\boldsymbol{D}$ is diagonal, with the elements that would have been in $\boldsymbol{U}$ from standard factorization
- Now it turns out that $\boldsymbol{D}^{-1}\boldsymbol{U} = \boldsymbol{L}^T$
- Then $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^T$

Symmetric positive definite matrices

- If $\boldsymbol{A}^T = \boldsymbol{A}$, and
- if $\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} \geq 0$ for all $x \in R^n$ and $x \neq 0$,
- Then $\boldsymbol{A}$ is positive definite
- We can modify the LU factorization again to the Cholesky factorization:

$$\boldsymbol{A} = \boldsymbol{R}^T \boldsymbol{R}, \qquad \boldsymbol{R} = \boldsymbol{D}^{1/2} \boldsymbol{L}^T$$

- $\boldsymbol{R}$ has all positive entries on the diagonal
- The MATLAB function `chol` will compute the Cholesky factorization

Symmetric positive definite matrices

- Start with `A=magic(5);`

```
B = A'*A

B =
        1055         865         695         770         840
         865        1105         815         670         770
         695         815        1205         815         695
         770         670         815        1105         865
         840         770         695         865        1055

R = chol(B)

R =
   32.4808   26.6311   21.3973   23.7063   25.8615
         0   19.8943   12.3234    1.9439    4.0856
         0         0   24.3985   11.6316    3.7415
         0         0         0   20.0982    9.9739
         0         0         0         0   16.0005

norm( R'*R - B )

ans =
     0
```

Symmetric positive definite matrices

- If $\boldsymbol{A}^T = \boldsymbol{A}$, and
- if $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \geq 0$ for all $x \in R^n$ and $x \neq 0$,
- Then $\boldsymbol{A}$ is positive definite
- We can modify the LU factorization again to the Cholesky factorization:

$$\boldsymbol{A} = \boldsymbol{R}^T\boldsymbol{R}, \qquad \boldsymbol{R} = \boldsymbol{D}^{1/2}\boldsymbol{L}^T$$

- $\boldsymbol{R}$ has all positive entries on the diagonal
- The MATLAB function `chol` will compute the Cholesky factorization