Representing linear systems as matrix-vector equations

$$\begin{aligned} 10x_1 - 7x_2 &= 7 \\ -3x_1 + 2x_2 + 6x_3 &= 4 \\ 5x_1 - x_2 + 5x_3 &= 6 \end{aligned}$$

- Represent it as a matrix-vector equation (linear system)
- We will apply the familiar elimination technique, and then see its matrix equivalent

$$\begin{pmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 4 \\ 6 \end{pmatrix}$$

Easy systems to solve

- Identity
- Diagonal
- Triangular

$$U = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 5 & 6 & 7 \\ 0 & 0 & 8 & 9 \\ 0 & 0 & 0 & 10 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 5 & 1 & 0 \\ 4 & 6 & 7 & 1 \end{pmatrix}$$

Solving a triangular system

```
x = zeros(n,1);
for k = n:-1:1
    x(k) = b(k)/U(k,k);
    i = (1:k-1)';
    b(i) = b(i) - x(k)*U(i,k);
end
```

Cost of solving a triangular system
Loop of size n. Each loop has a cost of k (or n-k)
So total cost is
$n*1 + n*2 + \ldots n*n = n^2$

Gaussian Elimination

- Zero elements of first column below 1st row
  - multiplying 1st row by 0.3 and add to 2nd row
  - multiplying 1st row by -0.5 and add to 3rd row
  - Results in
- Zero elements of first column below 2nd row
  - Swap rows
  - Multiply 2nd row by 0.04 and add to 3rd

$$\begin{pmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 4 \\ 6 \end{pmatrix}$$

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & -0.1 & 6 \\ 0 & 2.5 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 6.1 \\ 2.5 \end{pmatrix}$$

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & -0.1 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 2.5 \\ 6.1 \end{pmatrix}$$

$$\begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & 0 & 6.2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 2.5 \\ 6.2 \end{pmatrix}$$

Solution

- Start from last equation which can be solved by division
- Next substitute in the previous line and continue

$$6.2x_3 = 6.2$$

$$2.5x_2 + (5)(1) = 2.5.$$

$$10x_1 + (-7)(-1) = 7$$

$$x = \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix}$$

- This describes the way to do the algorithm by hand
- How to represent it using matrices?
- Suppose we wished to solve another system that has the same matrix?

$$\begin{aligned} 10x_1 - 7x_2 &= 7 \\ -3x_1 + 2x_2 + 6x_3 &= 3.901 \\ 5x_1 - x_2 + 5x_3 &= 6 \end{aligned}$$

  - Upper triangular matrix we end up with will be the same, but the sequence of operations on the r.h.s needs to be repeated
  - Can we do that efficiently?

Gaussian Elimination: LU Matrix decomposition

- Gaussian elimination corresponds to a LU decomposition
  - Product of permutation, lower triangular and upper triangular matrices
- Permutation Matrix
  - Multiplying swaps order of rows in a matrix/vector
  - Rearrangement of the identity
  - Nice property: transpose is its inverse: $PP^T=I$

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

$$Px = b$$

$$x = P^T b$$

LU Decomposition

- $LU=PA$
- upper triangular matrix is the final matrix we end up with
  - Elements below diagonal are zero
- Lower triangular matrix
  - Elements above diagonal are zero
  - Ones along diagonal
  - Has the multipliers used in the elimination
  - Multiplication leaves output first row same
  - Second row output is a combination of first row with multiplier plus second row

$LU=PA$

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ -0.3 & -0.04 & 1 \end{pmatrix} \qquad U = \begin{pmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & 0 & 6.2 \end{pmatrix}$$

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

- L has the multipliers we used in the elimination steps
- P has a record of the row swaps we did to avoid dividing by small numbers

$$L_1 L_2 \cdots L_{n-1} U = P_{n-1} \cdots P_2 P_1 A$$

Solving a system with the LU decomposition

$$Ax=b$$

$$LU=PA$$

$$P^T LUx=b$$

$$L[Ux]=Pb$$

Solve $Ly=Pb$

Then $Ux=y$

LU Remarks

- Operations count: $n^3/3$ multiplications.
- Matlab's **backslash** operator solves linear systems, using LU, without forming the inverse:

```
x = A \ b;
```

- If you have k right-hand sides involving the same matrix, store them as columns in a matrix B of size $n \times k$ and then solve using, for example

```
X = A \ B;
```

What about sparsity?

If A has lots of zeros, we would like our algorithms to take advantage of this, and not to ruin the structure by introducing many nonzeros.

If A is initialized as a sparse matrix in Matlab, then backslash and the `lu` algorithm both try to preserve sparsity.

Other computations

- Cholesky decomposition
  - Real symmetric positive definite matrices
  - No need for pivoting
  - $A=LDU = LDL^t=LL^t$
- Determinant
  - Usual formula (minors/cofactors) has factorial cost
  - Instead observe that
    - Determinant of individual matrices in a product multiply
    - Determinant of triangular matrix is obtained by multiplying diagonal
- Inverse
  - Get LU decomposition
  - Solve with N right hand sides (columns of identity matrix)
  - Arrange solutions in columns to get inverse matrix

Is pivoting necessary in LU?

- Consider

$$\begin{bmatrix} \delta & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

- Exact solution is

$$x = \begin{bmatrix} -\frac{1}{1-\delta} \\ \frac{1}{1-\delta} \end{bmatrix}$$

- Let $\delta < 0.5 * \varepsilon$
- Solution without pivoting gives

$$\begin{bmatrix} \delta & 1 \\ 0 & -1/\delta \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -1/\delta \end{bmatrix}$$

$$x_2 = 1, \;\; x_1 = 0$$

Is pivoting necessary?

- With pivoting $$\begin{bmatrix} 1 & 1 \\ \delta & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$
- Elimination gives $$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$
- With answers $$x_2 = 1, \;\; x_1 = -1 \, .$$
- Close to exact

How accurate are answers from LU?

- We solve the equation $Ax=b$
- Let true solution be x^*
- Let obtained solution be x
- Then error is $e= x^*-x$
  - Error is not computable ("Forward" error)
- New concept "residual" ("Backward error")
  - Residual is the difference between the original right hand side and the right hand side obtained with the obtained solution

$$r=b-Ax$$

- Guarantee: LU produces answers with small residuals
  - on computers with IEEE floating point
- Do small residuals mean small errors?

Return to our example

- Compute residual

$$\begin{aligned} r &\equiv b - Ax \\ &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ \delta & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 \\ \delta \end{bmatrix}. \end{aligned}$$

- We have exactly solved a nearby problem

$$Ax = b - r$$

Another example

assume 3-digit decimal arithmetic.

$$\begin{bmatrix} .780 & .563 \\ .913 & .659 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} .217 \\ .254 \end{bmatrix}$$

If we compute the solution with pivoting, we obtain

$$x = \begin{bmatrix} -.443 \\ 1.000 \end{bmatrix}, \quad r = \begin{bmatrix} -.000460 \\ -.000541 \end{bmatrix}$$

$$x_{true} = \begin{bmatrix} 1.000 \\ -1.000 \end{bmatrix}$$

- Solution has small residual but very large error
- In fact signs of the solution are opposite!

Condition numbers

The first problem is **well-conditioned**; small changes in the data produce small changes in the answer.

The second problem is **ill-conditioned**; small changes in the data can produce large changes in the answer.

- Recall definition of condition number

Condition Number of a Matrix

A measure of how close a matrix is to singular

$$\text{cond}(A) = \kappa(A) = \|A\| \cdot \|A^{-1}\|$$

$$= \frac{\text{maximum stretch}}{\text{maximum shrink}} = \frac{\max_i |\lambda_i|}{\min_i |\lambda_i|}$$

- cond(I) = 1
- cond(singular matrix) = ∞

Properties of the condition number

Some properties:

- $\kappa(A) \geq 1$ for all matrices.
- $\kappa(A) = \infty$ for singular matrices.
- $\kappa(cA) = \kappa(A)$ for any nonzero scalar c.
- $\kappa(D) = \max |d_{ii}| / \min |d_{ii}|$ if D is diagonal.
- κ measures closeness to singularity better than the determinant.

Relation between condition number and error

$$Ax_{true} = b \quad \rightarrow \quad \|b\| = \|Ax_{true}\| \leq \|A\| \, \|x_{true}\|$$

$$\|x_{true}\| \geq \frac{\|b\|}{\|A\|} \quad \rightarrow \quad \frac{\mathbf{1}}{\|\mathbf{x_{true}}\|} \leq \frac{\|\mathbf{A}\|}{\|\mathbf{b}\|}$$

$$Ax = b - r \quad \rightarrow \quad A(x_{true} - x) = r$$

$$(x_{true} - x) = A^{-1} r \quad \rightarrow \quad \|\mathbf{x_{true} - x}\| \leq \|\mathbf{A^{-1}}\| \, \|\mathbf{r}\|$$

$$\frac{\|\mathbf{x_{true} - x}\|}{\|\mathbf{x_{true}}\|} \leq \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|} \|\mathbf{A}\| \|\mathbf{A^{-1}}\|$$

$$= \frac{\|r\|}{\|b\|} \kappa(A) \, .$$

- In words: relative error is smaller than norm of residual divided by norm of rhs times condition number
- So larger condition number means larger error

Closing remarks

- Never compute matrix inverse
- Use a stable algorithm
- Check residual and condition number of problem
- If condition number is large, do not trust solution
  - Can problem be reformulated somehow?

LU code

- Initialize
  - Matrix size
  - Permutation vector
- Second output argument to `max` is index of max element
- If max element is zero then we need not eliminate
- Exchange rows
- update permutation vector

```
%LU Triangular factorization
%   [L,U,p] = lutx(A) produces a unit lower triangular
%   matrix L, an upper triangular matrix U, and a
%   permutation vector p, so that L*U = A(p,:).

[n,n] = size(A);
p = (1:n)'

for k = 1:n-1

   % Find largest element below diagonal in k-th column
   [r,m] = max(abs(A(k:n,k)));
   m = m+k-1;

   % Skip elimination if column is zero
   if (A(m,k) ~= 0)

      % Swap pivot row
      if (m ~= k)
         A([k m],:) = A([m k],:);
         p([k m]) = p([m k]);
      end
```

Look at LU code

- Multipliers for each row below diagonal
  - Note multipliers are stored in the lower triangular part of A
- Vectorized update
  - A(i,k)*A(k,j) multiplies column vector by row vector to produce a square, rank 1 matrix of order n-k.
  - matrix is then subtracted from the submatrix of the same size in the bottom right corner of A.
  - In a programming language without vector and matrix operations, this update of a portion of A would be done with doubly nested loops on i and j.
  - Cost is n^2 and done n times for a total cost of n^3
- Computes decomposition in the matrix A itself
- Here they are separated, but when memory is important it can be left there

```
        % Compute multipliers
        i = k+1:n;
        A(i,k) = A(i,k)/A(k,k);

        % Update the remainder of the matr
        j = k+1:n;
        A(i,j) = A(i,j) - A(i,k)*A(k,j);
    end
end

% Separate result
L = tril(A,-1) + eye(n,n);
U = triu(A);
```

Code to solve linear system using LU

- In Matlab the backslash operator can be used to solve linear systems.
- For square matrices it employs LU or special variants
  - Lower triangular
  - Upper triangular
  - symmetric
- Symmetric LU is called Cholesky decomposition
  - $A=LL^T$
  - Upper and lower triangular are equal (transposes)
  - If matrix not positive-definite go to regular solution

```
function x = bslashtx(A,b)
% BSLASHTX  Solve linear system (backslash)
% x = bslashtx(A,b) solves A*x = b

[n,n] = size(A);
if isequal(triu(A,1),zeros(n,n))
   % Lower triangular
   x = forward(A,b);
   return
elseif isequal(tril(A,-1),zeros(n,n))
   % Upper triangular
   x = backsubs(A,b);
   return
elseif isequal(A,A')
   [R,fail] = chol(A);
   if ~fail
      % Positive definite
      y = forward(R',b);
      x = backsubs(R,y);
      return
   end
end
```

Code continues

- Call LU
  - Solve y=Lb
  - Solve x=Uy

```
% Triangular factorization
[L,U,p] = lutx(A);

% Permutation and forward elimination
y = forward(L,b(p));
x = backsubs(U,y);
```

```
function x = forward(L,x)
% FORWARD. Forward elimination.
% For lower triangular L, x = forward(L,b) solves L*x = b.
[n,n] = size(L);
for k = 1:n
   j = 1:k-1;
   x(k) = (x(k) - L(k,j)*x(j))/L(k,k);
end
```

```
function x = backsubs(U,x)
% BACKSUBS.  Back substitution.
% For upper triangular U, x = backsubs(U,b) solves U*x = b.
[n,n] = size(U);
for k = n:-1:1
   j = k+1:n;
   x(k) = (x(k) - U(k,j)*x(j))/U(k,k);
end
```