Numerical Concerns

- (Recall) computers have only finite representations of numbers
  - Easy consequence: not all real numbers can be stored
  - Do we “lose” any important ones? How do we decide which?

Correct Rounding

- A single numerical procedure should follow the following algorithm:
  - Compute the exact result
  - Round to the nearest computer number
- i.e. the final result is the exact result, rounded
- This obviously leads to problems when intermediate arithmetic would lose precision
  - Consider 1 – 1e10
  - What might happen?

Precision with Linear Alg.

- Recall during the Gauss-Jordan algorithm implementation:
  - NaN values were pretty easy to come by!
  - Why?
- Pivoting with 0’s in the diagonal causes bad problems
  - One way to avoid is to swap with another row (annoying in parallel, but it works)

Precision in Elimination

- Consider the system $\begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} x = \begin{pmatrix} 1+\epsilon \\ 2 \end{pmatrix}$
- Has solution x=(1,1) – check!
- What will the Gauss-Jordan algorithm do when ϵ is less than the machine precision?
- After first round, we get $\begin{pmatrix} \epsilon & 1 \\ 0 & 1-\frac{1}{\epsilon} \end{pmatrix} x = \begin{pmatrix} 1+\epsilon \\ 2-\frac{1+\epsilon}{\epsilon} \end{pmatrix}.$
- So if ϵ is too small, we get (0,1)
  - Very wrong!

Precision in Elimination

- Simplest way to avoid:
  - “Partial pivoting” - always pivot to put the largest remaining diagonal element in the pivot row
    - Do a row swap, then go about standard parallel algorithm
- Better way: “diagonal pivoting”
  - Exchange both row and column (equivalent to re-numbering the unknowns)
  - Makes algorithm more parallel!

Precision with Eigenvalues

- Consider matrix $A = \begin{pmatrix} 1 & \epsilon \\ \epsilon & 1 \end{pmatrix}$
  - Where $\epsilon_{\text{mach}} < |\epsilon| < \sqrt{\epsilon_{\text{mach}}}$
- Has eigenvalues 1+ϵ and 1- ϵ
- Also consider characteristic polynomial
  - $\begin{vmatrix} 1-\lambda & \epsilon \\ \epsilon & 1-\lambda \end{vmatrix} = \lambda^2 - 2\lambda + (1-\epsilon^2) = \lambda^2 - 2\lambda + 1.$
  - Using this, we would get both eigenvalues equal to 1. But the are both expressible!
  - So we need a better algorithm!

Much worse example

- Consider: $$A = \begin{pmatrix} 20 & 20 & & & \emptyset \\ & 19 & 20 & & \\ & & \ddots & \ddots & \\ & & & 2 & 20 \\ \emptyset & & & & 1 \end{pmatrix}.$$

  By linear algebra, the eigenvalues should be exactly the diagonal elements, because it is upper-triangular

- However, if we set the bottom-left to 1e-6, we see the following eigenvalues:

$$\lambda = 20.6 \pm 1.9i, 20.0 \pm 3.8i, 21.2, 16.6 \pm 5.4i, \ldots$$

Approaching better algs

- Specifically if having to solve multiple linear systems, we can “save” some of the pivoting information for later
  - Recall: if the matrix is not square, inversion is not a good tool! (why?)
- Consider: can we write each single pivot step in a single, concise, matrix formula?
  - Yes!

LU Decomposition

- Consider the first pivot operation in the elimination algorithm:
  - Example: $A = \begin{pmatrix} 6 & -2 & 2 \\ 12 & -8 & 6 \\ 3 & -13 & 3 \end{pmatrix}$
  - First pivot phase is the same as multiplying A on the left by

$$L_1 = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1/2 & 0 & 1 \end{pmatrix}$$

  - Notice the divisors in the first column!
  - Other entries are the identity matrix

LU Decomposition

- We can do the same with the second step!
  - The second pivot is a left-multiplication by the matrix

$$L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -3 & 1 \end{pmatrix}$$

- So now we have $L_2 L_1 A\,x = L_2 L_1\,b$
  - Where L_1 and L_2 are triangular
- If we define $U = L_2 L_1\,A$
- Then we get $A = (L_1)^{-1}(L_2)^{-1}\,U$

LU Decomposition

- Observe that we have:

$$L_1 = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1/2 & 0 & 1 \end{pmatrix} \qquad L_1^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1/2 & 0 & 1 \end{pmatrix}$$

$$L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -3 & 1 \end{pmatrix} \qquad L_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 3 & 1 \end{pmatrix}$$

- And, importantly:

$$L_1^{-1} L_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1/2 & 3 & 1 \end{pmatrix}$$

LU Decomposition

- Finally, if we let $L = (L_1)^{-1} (L_2)^{-1}$ then we have $A = LU$, where L and U are both triangular!
  - So what?
  - We can compute L and U in-place and over-write the values inside A, which saves a lot of space (if we don't mind losing A)
- Going back to elimination…
  - How does this help?

LU Decomposition

- Given L and U, we can solve $Ax = LUx = b$ in two steps:
  - Solve $Ly = b$ for y (easy because L is triangular!)
  - (Again, easily) Solve $Ux = y$ for x
- Now, how to compute L and U?
  - Still not too bad, just use same G-J structure

LU Decomposition

- Compare with Gauss-Jordan:

$\langle LU$ factorization$\rangle$:
  for $k = 1, n-1$:
    for $i = k+1$ to n:
      $a_{ik} \leftarrow a_{ik}/a_{kk}$
      for $j = k+1$ to n:
        $a_{ij} \leftarrow a_{ij} - a_{ik} * a_{kj}$

- Leaves L and U in the off-diagonal entries of A
  - Leaves pivots in the diagonal

Next steps...

- Still need to worry about numerical concerns
  - That division is still dangerous!
- Is it possible that L and U are the same, but transposed?
  - Surprisingly, yes!
  - This would be twice as fast to compute!

Some definitions

- A matrix is *symmetric positive definite* (SPD) if it is symmetric ($A = A^T$) and if for all vectors x we have $x^T A x > 0$
- This will make things easier because an SPD matrix always has positive diagonal entries
- Even better, for any A we have $B = A^T A$ is SPD!

Cholesky Factorization

- Aka “Cholesky Decomposition”
  - Given: A is symmetric
  - We want to write $A = L\,L^T$
- Has a “simple” recursive formulation

$$\begin{bmatrix} a_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} l_{11} & L_{21}^T \\ 0 & L_{22}^T \end{bmatrix}$$

$$= \begin{bmatrix} l_{11}^2 & l_{11} L_{21}^T \\ l_{11} L_{21} & L_{21} L_{21}^T + L_{22} L_{22}^T \end{bmatrix}$$

Cholesky Factorization

- To formulate the algorithm recursively:

1) Compute

$$l_{11} = \sqrt{a_{11}}, \qquad L_{21} = \frac{1}{l_{11}} A_{21}$$

2) Recursively find L_{22} by factoring:

$$A_{22} - L_{21} L_{21}^T = L_{22} L_{22}^T$$

Properties

- An LU decomposition is not unique!
- Suppose $A = L_1U_1 = L_2U_2$ where the L's and U's are lower and upper-trianguar, resp.
- Then $(L_2)^{-1}L_1 = U_2(U_1)^{-1}$ where the left is lower-triangular and the right is upper-triangular
  - Contradiction? No!
  - They must be *diagonal*
- So, we may have different diagonal scaling in the factorization