Recitation 3

Recitation Instructor: Shivam Verma
Email: shivamverma@nyu.edu
Ph: 718-362-7836
Office hours: WWH 605 (2.50 - 4.50 pm, Tuesdays)

Brief overview

- About solving linear systems
- Big Oh notation and "order" of computation
- Gaussian elimination
- LU decomposition for solving linear systems
- Example/exercise of LU decomposition
- LU decomposition in MATLAB

Solving linear systems (15 min)

- We use Gaussian elimination to solve systems of linear equations
- Why do we care about Gaussian elimination / LU decomposition etc.? Why not just calculate the inverse? (Hint: flops!)
- Kinds of linear systems include
  - Sparse / dense
  - Symmetry, positive-definiteness, diagonally dominant (special factorizations)
  - Triangular systems (forward/backward substitution is cheap)
- Big Oh notation - $f(x) = O(g(x)) \Leftrightarrow |f(x)| \leq M|g(x)|,\ \ x \geq x_0.$ (measures the limiting behaviour of a function)
- Big Oh notation used in numerical analysis to measure # of mathematical operations (flops) for a numerical computation
- Eg. Matrix-vector, matrix-matrix multiplication
- How many operations does calculating a determinant take using the usual method?
- Ans: ????
- Why is this important. Consider a matrix of order 100. What is O(n!)? 9.3326 x 10^157! This will take roughly 10^127 years to calculate using a supercomputer.

LU factorization has two steps: (20 min)

- Gaussian Elimination:
    - Discovered by Gauss in 1800s. Known over 2000 years ago by the Chinese!

$$
\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1\\
&a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2\\
&\ldots\\
&a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n
\end{aligned}
$$

Apply row transformation: $R_i -> R_i - l_{i1}R_1$

$$
\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1\\
&(a_{21} - l_{21}a_{11})x_1 + \ldots + (a_{2n} - l_{21}a_{11})x_n = b_2\\
&\ldots\\
&(a_{n1} - l_{n1}a_{11})x_1 + \ldots + ((a_{nn} - l_{n1}a_{11})x_n = b_n
\end{aligned}
$$

We choose $l_{ik} = a_{ik}/a_{kk}$ so that after k such sequences:

$$
\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \ldots. + a_{1n}x_n = b_1\\
&\qquad a_{22}x_2 + \ldots\ldots. + a_{2n}x_n = b_2\\
&\ldots \qquad \ldots\\
&\qquad\qquad a_{kk}x_k + \ldots + a_{kn}x_n = b_k\\
&\qquad\qquad a_{nk}x_k + \ldots + a_{nn}x_n = b_n
\end{aligned}
$$

\# of mult operations = $\sum_{k=1}^{n-1} k^2 \approx n^3/3$

- Solution of a right (left) triangular system using backward (forward) substitution:

$$
\begin{aligned}
&r_{11}x_1 + r_{12}x_2 + \ldots + r_{1n}x_n = z_1\\
&\qquad r_{22}x_2 + \ldots + r_{2n}x_n = z_2\\
&\qquad \ldots\\
&\qquad\qquad\qquad r_{nn}x_n = z_n
\end{aligned}
$$

Or, $Rx = z$.

$$
\begin{aligned}
&x_n = z_n/r_{nn}\\
&x_{n-1} = (z_{n-1} - r_{n-1,n}x_n)/r_{n-1,n-1}
\end{aligned}
$$

…

$x_1 = (z_1 - r_{1,2}x_2 - ... - r_{1n}x_n)/r_{11}$

# of operations for i'th row = (n-i) additions, multiplications and 1 division

Thus, total add/mult operations = $\sum_{i=1}^{n}(i-1) = n(n-1)/2 \approx n^2/2$]

- **Important**: LU decomposition does NOT depend on 'b' (RHS), i.e. same LU decomposition for a matrix A.

Let's try solving the following system(s) using LU decomposition! (10 min)

1.

A = [6, 0, 2;
 24, 1, 8;
 -12, 1, -3];

b = [4;
 19;
 -6];

Ans:

L = [1, 0, 0;
 4, 1, 0;
 -2, 1, 1];

U = [6, 0, 2;
 0, 1, 0;
 0, 0, 1];

x = [1;
 3;
 -1];

2.
https://www.utdallas.edu/dept/abp/PDF_Files/LinearAlgebra_Folder/LU_Decomposition.pdf

3.

A =
[4, 2, 1;
2, 1, 1;
1, 1, 1]

Ans.

L =
[1, 0, 0;
½, 0, 1;
¼, 1, 0]

U =
[4, 2, 1;
0, ½, ¾;
0, 0, ½]

Hint: Recall Theorem 2.2 from book (pg. 50). Is there a principal diagonal submatrix (minor) which is singular?

Pivoting

When do we need pivoting?

- Any leading diagonal submatrix is singular
- A(1,1) is zero
- Eg. See example 3 in prev. section.

Practicing solving linear equations in MATLAB (30 min)

- The **'\' operator** in MATLAB is used for solving all kinds of linear equations. Try it!

- **Forward substitution in MATLAB** (example)

- Comparing **three different implementations** of forward substitution in MATLAB - looping over rows and columns (forward1), looping over rows only (forward2), looping over columns only (forward3).

- Why do the **timings** of these three implementations of the **same algorithm differ** so much?
- Ans. Despite the fact that each method requires roughly the **same number of floating point operations**, timings differ significantly. This is mainly due to **memory access** and how **matrices are stored in memory. MATLAB stores matrices as one-dimensional arrays, column by column**. You can see that by accessing matrices with only one index, i.e., using $A(k)$ for an n×n matrix returns the entry $A(m,l)$, where $k = m + (l-1)n$, $1 \leq l, 1 \leq m \leq n$. Numbers that are next to each other in memory can be read from memory much faster than numbers that are stored further away from each other.
- **Lesson: Columns and rows are NOT the same in MATLAB!**

- Finally, LU decomposition in MATLAB.
  - [L U P] = lu(A)
  - **help lu**

Helpful resources

1. More about cost of computation
2. Wiki page for LU decomp.
3. MIT OCW example for LU
4. LU decomp. explained very well here
5. Detailed notes on LU decomp. and linear systems
6. Why columns and rows are not the same in MATLAB