Recitation 4

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 (5 min)

- Two kinds of linear system solvers:
    - Factorization-based solvers (LU/Gaussian, Cholesky, LDL etc.)
    - Iteration-based solvers (Jacobi, Gauss-Seidel etc.)
- 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 - $f(x) = O(g(x))$ ⇔ $|f(x)| \leq M|g(x)|, \;\; x \geq x_0.$ (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: $O(n!)$
- 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.
- Read [1] to know more about cost of diff. matrix ops!

LU factorization has two steps: (5 min)

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

$$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2$$
$$\dots$$
$$a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n = b_n$$

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

$$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1$$
$$(a_{21} - l_{21}a_{11})x_1 + ... + (a_{2n} - l_{21}a_{11})x_n = b_2$$
$$\dots$$
$$(a_{n1} - l_{n1}a_{11})x_1 + ... + ((a_{nn} - l_{n1}a_{11})x_n = b_n$$

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

$$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1$$
$$a_{22}x_2 + ... + a_{2n}x_n = b_2$$
$$\dots \quad \dots$$
$$a_{kk}x_k + ... + a_{kn}x_n = b_k$$
$$a_{nk}x_k + ... + a_{nn}x_n = b_n$$

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

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

Total # of operations = $\approx 2n^3/3$

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

$$r_{11}x_1 + r_{12}x_2 + ... + r_{1n}x_n = z_1$$
$$r_{22}x_2 + ... + r_{2n}x_n = z_2$$
$$\dots$$
$$r_{nn}x_n = z_n$$

Or, $Rx = z$.

$x_n = z_n / r_{nn}$
$x_{n-1} = (z_{n-1} - r_{n-1,n} x_n) / r_{n-1,n-1}$
$\ldots$
$x_1 = (z_1 - r_{1,2} x_2 - \ldots - r_{1n} x_n) / r_{11}$

# 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.

Try solving the following system(s) using LU decomposition! (home exercise)

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]

Try this with and without pivoting. What do you observe?

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.

Pivoting not needed if matrix is diagonally dominant. Why?

- Diagonally dominant matrix: $|A_{ii}| \geq \sum_{j \neq i} |A_{ij}|$ for all i (rows)

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.
    - The MATLAB '\\' operator is the most efficient and takes least time, as expected.
    - The 'two for loops' implementation (forward1) should take around 7 seconds, while forward2 and forward3 take around 0.75 and 0.3 seconds respectively.
    - 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 $k = m + (l-1)n$, $1 \le l, 1 \le m \le n$. 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, and comparison with '\' operator and inverse (cramer's rule).
    - [L U P] = lu(A)
    - **help lu**

Try the following code in MATLAB (left followed by right):

%dense solver
disp('For dense matrix')
n = 5000;
A = rand(n,n);
b = rand(n,1);
 	
disp('For A\b');
tic;
A\b;
toc;
disp('For LU');
tic;
[L,U,P]=lu(A);
b_new = P*b;
y = L\b_new;
U\y;
toc;
disp('Inv(A)*B');
tic;
inv(A)*b;
toc;
	%sparse solver
	disp('For sparse matrix')
	C = diag(rand(n,1));
	
	disp('For C\b');
	tic;
	C\b;
	toc;
	disp('For LU');
	tic;
	[L,U,P]=lu(C);
	b_new = P*b;
	y = L\b_new;
	U\y;
	toc;
	disp('Inv(C)*B');
	tic;
	inv(C)*b;
	toc;

Output should look like:

```
For dense matrix
For A\b
Elapsed time is 2.310573 seconds.
For LU
Elapsed time is 2.018139 seconds.
Inv(A)*B
Elapsed time is 4.974864 seconds.
For sparse matrix
For C\b
Elapsed time is 0.109042 seconds.
For LU
Elapsed time is 2.002971 seconds.
Inv(C)*B
Elapsed time is 4.922749 seconds.
```

- Can you explain this behaviour? How is MATLAB's backslash '\' operator so awesome?

    - Determines properties of matrix and applies algorithm accordingly
        - Eg. Triangular -> forward/backward substitution, symmetric -> LDL
    - Behind-the-scenes hardware optimizations
    - Read [7] this for more info.
    - If you're curious, MATLAB's sparse solver uses a package called UMFPACK. Read more about it in the "backslash" book [9]: *Direct methods for sparse linear systems*, Tim Davis, SIAM, 2006.
    - Note: A\b == mldivide(A,b) [another 'name' for 'backslash' operator]

To see in live motion what MATLAB's backslash operator is doing, run the command **spparms('spumoni',1);** before solving for **A\b**. To know why MATLAB chooses an algorithm, see [8].

Example from [7]:

Input:

```
spparms('spumoni',1);
A = delsq(numgrid('B',256));
b = rand(size(A,2),1);
mldivide(A,b); % another way to write A\b
```

Output:

```
sp\: bandwidth = 254+1+254.
sp\: is A diagonal? no.
sp\: is band density (0.01) > bandden (0.50) to try banded solver? no.
sp\: is A triangular? no.
sp\: is A morally triangular? no.
sp\: is A a candidate for Cholesky (symmetric, real positive diagonal)? yes.
sp\: is CHOLMOD's symbolic Cholesky factorization (with automatic reordering)
successful? yes.
sp\: is CHOLMOD's numeric Cholesky factorization successful? yes.
sp\: is CHOLMOD's triangular solve successful? yes.
```

Helpful links

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
7. Stackoverflow answer to how MATLAB \ solver works
8. Documentation for MATLAB \ solver
9. “Backslash” book