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 from last week (15 min)

- 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: $O(n!)$
- LU factorization of matrices

Today's Agenda

- Matrix and vector norms
- Discussion of Homework questions
  - Overview of Gaussian elimination
  - Householder's matrices
  - Extracting submatrices/vectors from a matrix
- Cholesky decomposition - algorithm, order of computation

Practicing solving linear equations in MATLAB (20 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).

Try the following code in MATLAB.

- Forward1 uses two for loops to do forward substitution.
- Forward2 uses one for loop (looping over rows)
- Forward3 uses one for loop (looping over columns)

Run the tester code below and observe the time taken for each of the forward substitution processes.

```
function [x] = forward1(L,b)
%naive forward substitution by looping
%over rows and columns
    n = length(b);
    x = zeros(n,1);
    x(1) = b(1)/L(1,1);
    for i=2:n
        s=0;
        for j=2:n
            s = s+L(i,j-1)*x(j-1);
        end
        x(i) = (b(i)-s)/L(i,i);
    end
end
```

```
function [x] = forward2(L,b)
%looping over rows
    n = length(b);
    x = zeros(n,1);
    x(1) = b(1)/L(1,1);
    for i=2:n
       x(i) = (b(i)-L(i,1:i-1)*x(1:i-1))/L(i,i);
    end
end
```

```
function [x] = forward3(L,b)
%looping over columns
    n = length(b);
    for j=1:n-1
        b(j) = b(j)/L(j,j);
        b(j+1:n) = b(j+1:n) - b(j)*L(j+1:n,j);
    end
    b(n) = b(n)/L(n,n);
    x=b;
end
```

```
%test forward1,forward2,forward3
N=10000;
A = rand(N); %generates NxN random matrix
b = rand(N,1);
L = tril(A); %converts A to lower triangular matrix
tic %starts measuring time taken
c0 = L\b;
toc %stops measuring time taken
tic
c1 = forward1(L,b);
toc
tic
c2 = forward2(L,b);
toc
tic
c3 = forward3(L,b);
toc
```

- 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.
  - [L U P] = lu(A)
  - **help lu**

Norms and inequalities (15 min)

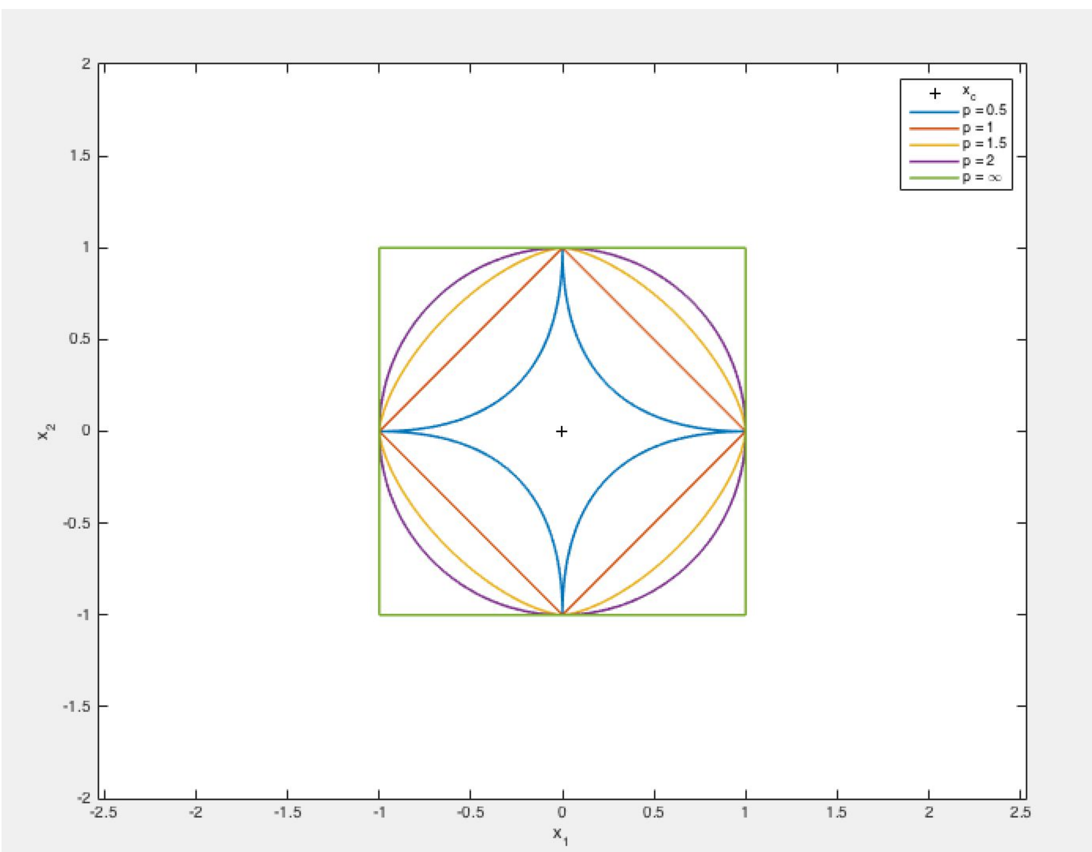
- What is a norm?
- An Lp norm is defined as $||x||_p = (\sum_{i=1}^{n} |x|^p)^{1/p}$
- 2-norm or Euclidean norm: $||x||_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$
- 1-norm: $||x||_2 = \sum_{i=1}^{n} |x_i|$
- $\infty -$norm: $||x||_\infty = max(|x_1|, |x_2|, ..., |x_n|)$

- Visualizing norms in MATLAB

```
function [] = VisualizeNorms(x_c,r,p)
%plots a p-norm ball with radius r, centred at x_c
   low = min(x_c);
   high = max(x_c);
   x = linspace(low-2*r,high+2*r);
   y = linspace(low-2*r,high+2*r);
   theta = 0:pi/100:3*pi;
   xp = r*sign(cos(theta)).*abs(cos(theta)).^(2/p);
   yp = r*sign(sin(theta)).*abs(sin(theta)).^(2/p);
   plot(xp,yp,'-','LineWidth',1.5)
end
```

```
%tester code
x_c = [0,0]; %center at origin
r = 1; %radius of ball is 1
plot(x_c(1),x_c(2),'k+') %plots the center
hold on
xlabel('x_1')
ylabel('x_2')
axis([-(x_c(1)+2*r) x_c(1)+2*r -(x_c(2)+2*r) x_c(2)+2*r])
axis equal
VisualizeNorms(x_c,r,0.5) %p = 0.5
VisualizeNorms(x_c,r,1) %p = 1
VisualizeNorms(x_c,r,1.5) %p = 1.5
VisualizeNorms(x_c,r,2) %p = 2
VisualizeNorms(x_c,r,inf) %p = inf
legend('x_c','p = 0.5','p = 1','p = 1.5','p = 2', 'p = \infty')
```

You should get a plot which looks like this:



- Think about why this happens. Read the references to know more about norms.
- Can also define matrix norms

- $sup_{v \in R^n} \frac{||Av||}{||v||}$

Special matrices

- Banded matrices
  - $a_{ij} = 0$ for all i and j such that $|i - j| < k$ for a k-band matrix
  - Eg. tri-diagonal matrix
- Positive-definite matrices
  - $x^T A x > 0$ for all non-zero x in real space
- What is a Householder matrix? Where is it used (hint: QR algorithm)? Why is it useful?

Discussion about Cholesky decomposition (15 min)

- What is a Cholesky decomposition? When can it be done? Why is it useful?
- If a matrix is positive semi-definite, it has a Cholesky decomposition $A = LL^T$.
- $Ax = b \Rightarrow LL^T x = b \Rightarrow L^T x = y$ (backward substitution) and $Ly = b$ (forward substitution)

write $A = A^{(1)}$ as

$$A^{(1)} = \left[\begin{array}{c|c} a_{11} & z^T \\ \hline z & B^{(1)} \end{array}\right],$$

where $z = (a_{12}, \ldots, a_{1n})^T$ and after one elimination step we obtain

$$A^{(2)} = L_1 A^{(1)} = \left[\begin{array}{c|c} a_{11} & z^T \\ \hline 0 & \\ \vdots & B^{(2)} \\ 0 & \end{array}\right] \quad \text{with} \quad L_1 = \begin{bmatrix} 1 & & & \\ -l_{21} & 1 & & \\ \vdots & & \ddots & \\ -l_{n1} & & & 1 \end{bmatrix}.$$

Now if we premultiply $A^{(2)}$ with L_1^T, then z^T in the first row is also eliminated and the remainder matrix $B^{(2)}$ remains unchanged, i.e.,

$$L_1 A^{(1)} L_1^T = \left[\begin{array}{c|ccc} a_{11} & 0 & \cdots & 0 \\ \hline 0 & & & \\ \vdots & & B^{(2)} & \\ 0 & & & \end{array}\right].$$

[Image source: Numerical Analysis in Modern Scientific Computing, Deuflhard & Hohmann (Ch. 1)]

- Cholesky algorithm -

$$
\begin{aligned}
&\textbf{for } k := 1 \textbf{ to } \mathrm{n} \textbf{ do} \\
&\qquad l_{kk} := (a_{kk} - \textstyle\sum_{j=1}^{k-1} l_{kj}^2)^{1/2}; \\
&\qquad \textbf{for } i := k+1 \textbf{ to } n \textbf{ do} \\
&\qquad\qquad l_{ik} = (a_{ik} - \textstyle\sum_{j=1}^{k-1} l_{ij} l_{kj})/l_{kk}; \\
&\qquad \textbf{end for} \\
&\textbf{end for}
\end{aligned}
$$

[Image source: Numerical Analysis in Modern Scientific Computing, Deuflhard & Hohmann (Ch. 1)]

- Computational cost = $n^3/3$ = half of LU decomposition

Helpful resources

1. Why columns and rows are not the same in MATLAB
2. Notes on vector norms
3. Notes on matrix norms
4. Intuition behind norms
5. Cholesky decomposition explained
6. About householder transformation - 1
7. About householder transformation - 2
8. *Numerical Mathematics (Quarteroni et al)* is a very good resource for LU/Cholesky/etc. It's also freely available for NYU students via Springer!