Lab Session 8

MA-581: Numerical Computation Lab R. Alam October 22, 2025

MATLAB Scripts and functions for $Ax = b$

1. A *magic square* is an n-by-n matrix in which each integer $1, 2, \ldots, n^2$ appears once and for which all the row, column, and diagonal sums are identical. The MATLAB command `magic` returns magic squares. Check its output for a few values of n and use MATLAB to verify the summation property. (The antidiagonal sum will be the trickiest. Look for help on how to "flip" a matrix.)

2. Consider the magic square `A = magic(n)` for `n = 3, 4, or 5`. What does

   ```
   p = randperm(n); q = randperm(n); A = A(p,q);
   ```

   do to

   ```
   sum(A), sum(A')', sum(diag(A)), sum(diag(flipud(A))), rank(A)
   ```

   The magic square `A = magic(4)` is singular. What do

   ```
   null(A), null(A,'r'), null(sym(A)), and rref(A)
   ```

   tell you about linear dependence of columns of `A`?

3. Are the following true or false? Assume A is a generic n-by-n matrix.
   (a) `A^ (-1)` equals `1/A` (b) `A.^ (-1)` equals `1./A`

4. (a) Look up `diag` in the online help and use it (more than once) to build the 16-by-16 matrix

   $$D = \begin{bmatrix} -2 & 1 & 0 & 0 & \cdots & 0 & 1 \\ 1 & -2 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 & -2 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 1 & -2 \end{bmatrix}$$

   (b) Now read about `toeplitz` and use it to build D.

5. The system $Ax = b$ can be solved as `x = A\b` and also via LU factorization of A. The LU factorization is more efficient (in terms of time taken) than backslash operator when the system is solved for multiple right hand sides. This can be checked as follow.

   ```
   A = randn(500, 500);    % 500x500 with normal random entries
   tic; for k=1:50; A \ rand(500, 1); end; toc
   ```

   Now solve the systems using LU factorization.

```
[L, U, p] = lu(A, 'vector');    % keep factorization result
tic
for k=1:50
b = rand(500, 1);
U \ (L \ b(p));
end
toc
```

What is your conclusion?

6. Given data $(t_1, y_1), \ldots, (t_n, y_n)$, we seek a polynomial $p(t) = c_1 + c_2 t + \cdots + c_n t^{n-1}$ such that $p(t_j) = y_j$ for $j = 1 : n$. This gives the linear system

$$\begin{bmatrix} 1 & t_1 & \cdots & t_1^{n-1} \\ 1 & t_2 & \cdots & t_2^{n-1} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & t_n & \cdots & t_n^{n-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

Set $\mathtt{c} := [\mathtt{c_1}, \ldots, \mathtt{c_n}]^\top$ and $\mathtt{y} := [\mathtt{y_1}, \ldots, \mathtt{y_n}]^\top$. The coefficient matrix V is the Vandermonde matrix and can be generated from the column vector $\mathtt{t} := [\mathtt{t_1}, \ldots, \mathtt{t_n}]^\top$ by

```
V = [t.^(1:n-1)]
```

The MATLAB command `vander` also generates `V` when $p(t)$ is written as $p(t) = c_n t^{n-1} + \cdots + c_2 t + c_1$.

The system `Vc = y` can be solved using backslash operator `c = V\y`. The following data gives the population of China

```
year = [1982; 2000; 2010; 2015];
pop = [1008.18; 1262.64; 1337.82; 1374.62];
```

It is convenient to measure time in years since 1980. So, consider `t := year-1980` and `y = pop`. Consider `V = vander(t)` and solve the system `Vc = y`. Check the residual vector `V*c-y`. We can use the polynomial to estimate the population of China in 2005:

```
p = @(t) polyval(c, t - 1980);  % include the 1980 time shift
p(2005)
```

The official population value for 2005 was 1303.72. Does $p(2005)$ give a good estimate of the actual population in 2005? We can visualize the interpolation process as follows.

```
scatter(year, y)
xlabel("years since 1980")
ylabel("population (millions)")
title("Population of China")
tt = linspace(1980, 2015, 500);    % 500 points in the interval [1980, 2015]
yy = p(tt);                        % evaluate p at all the vector elements
hold on
plot(tt, yy)
legend("data", "interpolant", "location", "northwest");
```

The Vandermonde matrix is known to be ill-conditioned for large large n. This can be seen by computing its condition number. Choose a column vector `t` with n distinct component and compute `cond(vander(t))` for various values of n.

7. Suppose that t the running time of an algorithm of complexity $\mathcal{O}(n^p)$. Then $t \approx Cn^p$ for sufficiently large n. Then $\log(t) \approx p\log(n) + \log(C)$. So, we expect that a graph of $\log(t)$ as a function of $\log(n)$ to be a straight line of slope p. Also, if t_n and t_N are running times with $n < N$ then $t_n/t_N \approx (n/N)^p \Longrightarrow t_n \approx t_N(n/N)^p$.

   We now check the efficiency of LU factorization. We plot the timings on a log-log graph and compare it with $\mathcal{O}(n^3)$ flops. The result could vary significantly from machine to machine, but in theory the data should start to parallel the line as $n \to \infty$.

```
n = (200:100:2400)';
t = zeros(size(n));
for k = 1:length(n)
A = randn(n(k), n(k));
tic    % start a timer
for j = 1:6,  [L, U] = lu(A);  end
time = toc;
t(k) = time / 6;
end

clf
loglog(n, t, 'o-')
hold on
loglog(n,  t(end) * (n/n(end)).^3, 'k--')
axis tight
xlabel('size of matrix'), ylabel('time (sec)')
title('Timing of LU factorization')
legend('lu','O(n^3)','location','southeast');
```

8. Write a MATLAB function that implements Gaussian elimination without pivoting. Your function should look like the following.

```
function [L, U] = GENP(A);
% [L U] = GENP(A) produces a unit lower triangular matrix L and an upper
% triangular matrix U so that A= LU.

[n, n] = size(A);
for k = 1:n-1
    % compute multipliers for k-th step
    A(k+1:n,k) = A(k+1:n,k)/A(k,k);
    % update A(k+1:n,k+1:n)
    j = k+1:n;
    A(j,j) = A(j,j)-A(j,k)*A(k,j);
end
% strict lower triangle of A, plus I
L = eye(n,n)+ tril(A,-1);
```

```
U = triu(A); % upper triangle of A
```

Consider $A = \begin{bmatrix} 10^{-16} & 1 \\ 1 & 5 \end{bmatrix}$, $x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $b := Ax$. Then $Ax = b$. Now solve $Ax = b$ using `GENP` as follows

```
[L,U] = GENP(A); xx = U\(L\b);
```

Compute `L*U`, `abs(A-L*U)` and `abs(x-xx)`. What do your observe? Compute the condition number $\|A\|_\infty \|A^{-1}\|_\infty$ by `cond(A, inf)`. Is the inaccurate result due to ill-conditioned A?. What can you conclude about GENP from this example? Can you identify the step at which things start to go wrong?

Next, consider $A = \begin{bmatrix} 1 & 0 & 0 & 0 & 10^{12} \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$ and $x = \begin{bmatrix} 0 \\ 1/3 \\ 2/3 \\ 1 \\ 4/3 \end{bmatrix}$ and $b := Ax$. Solve the system $Ax = b$ using the GENP as follows

```
[L,U] = GENP(A); xx = U\(L\b);
```

Again, compute `L*U`, `abs(A-L*U)` and `abs(x-xx)`. What do your observe? Compute the condition number $\|A\|_\infty \|A^{-1}\|_\infty$ by `cond(A, inf)`. Is the inaccurate result due to ill-conditioned A?. What can you conclude about GENP from this example?

Finally, repeat the above computations using `matlab` command `[L, U, P] = lu(A)` which uses GEPP. What do you observe?

9. Let $A = \begin{bmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}$ be an $n \times n$ tridiagonal matrix. Compute LU factorization of A for $n = 10$ using GENP. Is A positive definite? Try to compute Cholesky factorization of A. Use MATLAB command `chol`.

10. (**Homework**)The solution of a system of equations $Ax = b$ can be obtained in MATLAB by setting `x = A\b`. MATLAB uses GEPP (Gaussian Elimination with Partial Pivoting) to find x for this command. The same may also be found by setting `x = inv(A) * b`.

    Write an M-file which finds the time taken by both these commands (use `tic` and `toc` commands of MATLAB for this) for matrices with sizes `n = (200:100:2400)'`. Plot the result on a log-log graph in which the time is taken along the y-axis and the matrix sizes along the x-axis. The plots for both the methods should be on the same graph. Use `legends` to distinguish between curves.

*** End ***