Lab Session 3

MA423: Matrix Computations July-November, 2025 S. Bora

Important instructions:

(i) **Switch to `format long e` for all experiments.**

(ii) **Submit a single livescript program that contains all comments, answers and codes necessary to produce the required outputs. Ensure that the answers are correctly numbered and the file does not include any irrelevant material. The livescript program should be saved as MA423YourrollnumberLab3.mlx**

(iii) **Read the notes before attempting the questions 2 to 5.**

Note 1: The *Rule-of-thumb* of ill-conditioning says that if

1. $\kappa(A) \approx 10^t$;
2. the computed solution of $Ax = b$ has been obtained from a backward stable algorithm;
3. the entries of A and b are accurate to about s decimal places where $s > t$;

then one should expect an agreement of at least $s - t$ decimal places between the corresponding entries of the exact solution and the computed solution of the system.

Note 2. The famous Hilbert matrix H given by $H(i, j) := 1/(i + j - 1)$.

Origin: Suppose a function f is approximated by a polynomial p of degree $n - 1$, that is, $p(x) = a_0 + a_1 t + \cdots + a_{n-1} t^{n-1}$ on $[0, 1]$. The method of least squares (more on LSP, later in the course) determines a_j such that the error

$$E := \|f - p\|_2^2 = \int_0^1 (f(t) - p(t))^2 dt$$

is minimized. Differentiating E w.r.t a_k and setting the partial derivative to 0 (necessary condition for minima), we have

$$\sum_{j=1}^{n-1} a_j \int_0^1 t^{j+k} dt = \int_0^1 t^k f(t) dt, \ k = 0 : n - 1$$

which can be written in the matrix form $Ha = b$. Another interesting feature of the Hilbert matrix is that there is a special formula for generating its inverse.

There are built-in functions in Matlab for generating Hilbert matrix and its inverse. Type `help hilb` and `help invhilb` for details.

1. Write a function program `G = mychol(A)` that recursively computes the Cholesky factor of an $n \times n$ positive definite matrix A from its outer product formulation in $\frac{n^3}{3} + O(n^2)$ flops.

   Compare your output with that of the built in Matlab function program `chol` for several different choices of randomly generated positive definite matrices. For `G = mychol(A)` and $\hat{\mathtt{G}} = \mathtt{chol(A)}$, you should check whether $\mathtt{norm(G} - \hat{\mathtt{G}}\mathtt{)}$ is small.

The following commands may be used to randomly generate positive definite matrices with arbitrary choices of $0 < a < b$, and positive integers n :

$\gg$ `r = a + (b-a).*rand(n,1); D = diag(r)`
(Here `r` is a length `n` column vector of values randomly generated from an uniform distribution on the interval `[a,b]` and `D` is an $n \times n$ diagonal matrix with the entries of `r` on the diagonal.)
$\gg$ `B = randn(n); [Q,R] = qr(B)`
(Here B is an $n \times n$ matrix containing pseudorandom values drawn from the standard normal distribution and Q is an orthogonal matrix such that $B = QR$ is a QR decomposition of B.)
$\gg$ `A = Q'*D*Q.`

(The positive definite matrices are also precisely symmetric matrices with positive eigenvalues and the above commands generate them randomly.)

2. The purpose of this experiment is to note the extreme ill-conditioning of the Hilbert matrix. Convince yourself that the condition number of H grows quickly with n. Try

```
>> C=[];
>> N= 2:2:16;
for n=N
H=hilb(n); C=[C; cond(H)];
end
>> semilogy(N,C)
```

Based on this graph formulate a conjecture as to the relationship between cond(H) and n. [Note: The Matlab command `cond(H)` computes the 2-norm condition number of H. Type `help cond` for details.]

Modify the above code and compute the condition number in 1-norm and ∞-norm.

3. Given a Hilbert matrix H, the aim of this' exercise is to verify whether the rule of thumb for solutions of the system $Hx = b$ obtained via a number of different solution methods holds. In practice, the exact solution is unavailable. But for the experiments we can use the following trick to gain access to it:

Choose x first and set $b := Hx$ and then solve $Hx = b$ by setting $x1 = H\backslash b$. Then x is the exact solution of $Hx = b$ and $x1$ is the computed solution!

The system $Hx = b$ may be solved by using the `invhilb` command that generates the inverse of a Hilbert matrix. Moreover you can also use your `gepp` function formulated in the first problem to solve $Hx = b$. You may have to use `format long e` to see more digits. Note that $H \setminus b$ will NOT find a solution via GEPP in this case as H is positive definite. Instead it will use Cholesky method to find the solution. Now execute the following steps:

```
>> n=8;
>> H=hilb(n); HI = invhilb(n);
>> x= rand(n,1);
>> b =H*x;
>> x1 = H\ b; x2 = HI*b;
% obtain x3 by solving Hx=b using GEPP.
>> [x x1 x2 x3]
>> [cond(H) norm(x-x1)/norm(x) norm(x-x2)/norm(x) norm(x-x3)/norm(x)]
```

Repeat for $n = 10$ and $n = 11$ and list the results corresponding to $n = 8, 10, 12$, and determine correct digits in `x1, x2, x3.` Now answer the following questions:

How many digits are lost in computing `x1, x2` and `x3`? Does the loss of accuracy in each case agree with the value predicted by the *Rule-of-thumb* mentioned earlier? Note that since the experiments are performed on randomly generated data and their is no error other than rounding error, you should take $s \approx 16$ (wait for a few more classes to know why!) in the *Rule-of-thumb* analysis.

Which is better among `x1, x2` and `x3` or isn't there much of a difference?

4. The purpose of this experiment is to illustrate that a small value of the norm of the residual alone is not enough to guarantee an accurate answer.

   If $\hat{x}$ is the computed solution of $Ax = b$ then $r := A\hat{x} - b$ is called the **residual**. Of course $r = 0$ if and only if $x = \hat{x}$. But usually $r \neq 0$. Does a small $\|r\|/\|b\|$ imply $\|x - \hat{x}\|/\|x\|$ small? Try the following:

   ```
   >> n=10;
   >> H=hilb(n); x = randn(n,1);
   >> b = H*x;
   >> x1= H \ b;
   >> r = H*xt-b;
   >> disp( [norm(r)/norm(b) norm(x-xt)/norm(x)])
   ```

   What is your conclusion?

5. The purpose of this experiment is two-fold. Firstly, the aim is to show that only the coefficient matrix A having a small condition number is not enough to ensure accuracy in the computed solution of the system $Ax = b$. Secondly, the aim is to understand the role of the assumption of backward stability of the algorithm for the predictions from Rule-of-thumb analysis to hold.

   Recall the Wilkinson's matrix that you had generated in one of your previous classes. For $n = 32$, pick a random x and then compute $b := W * x$. Store the solution obtained by using GEPP in $\hat{x}$ and compute the (forward) error $\|x - \hat{x}\|_\infty / \|x\|_\infty$. Also compute cond$(A)$ in the infinity norm and $\|r\|_\infty / \|b\|_\infty$. Repeat the test for $n = 64$.

   Further repeat the above experiment using QR decomposition (More about this later but finding it is easy in MATLAB. Just typing `[Q,R] = qr(A)` gives unitary Q and upper triangular R such that $A = QR$.) Solve $Wx = b$ using QR decomposition (using `x = colbackward(Q'*b)`).

   Tabulate all the quantities for the different experiments. Noting that $s \approx 16$ in the *rule-of-thumb* analysis, answer the following.

   (a) Which of the two methods appear to give a lower error in the computed solution?

   (b) For which of the two methods is the *Rule-of-thumb* predicting the correct answer reasonably well?

   (c) Which of the two methods give rise to a lower value of $\|r\|_\infty / \|b\|_\infty$?

   (d) What can you say about the backward stability of GEPP and QR decomposition methods from the experiments?