Lab Session 3

MA-423: Matrix Computations July-November, 2021 S. Bora

Instructions: The reports for all experiments are to be written clearly in a single pdf file. This can be a .doc/.odt file converted to pdf format. Additionally, the workspaces of experiments where random matrices and vectors are generated are to be saved (as *.mat files) in the folder containing the work of this lab. The names of the matrices and vectors in these workspaces for which the results are tabulated are to be clearly mentioned in the report. This is essential so that these can be accessed by loading the workspace when checking the work. The following needs to be carefully noted before performing the experiments.

Note 1: *Significant digits* in a number and how to estimate the agreement between significant digits in corresponding entries of vectors.

(a) The number of significant digits in a number is the first nonzero digits and the number of all subsequent digits. For example 0.0123 has 3 significant digits while 1.0123 has 5 such digits.

(b) If the norm is the 1, ∞ or 2 norm and x and $\hat{x}$ are two vectors such that $\|x - \hat{x}\|/\|x\| \leq 0.5 \times 10^{-p}$, then this means that $x(i)$ and $\hat{x}(i)$ agree to at least p significant digits for all indices i.

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 $Ha = b$. (Show that H is always a positive definite matrix from this!) 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.

Note 3. The *Rule-of-thumb* of ill-conditioning says that if

1. $\kappa(A) = 10^t$;
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.

The reason for this is as follows. Backward stability implies that the backward errors $\frac{\|r\|_2}{\|b\|_2}$ and $\frac{\|r\|_2}{\|x_c\|_2\|A\|_2}$ in the computed solutions of $Ax = b$ are of the order of unit roundoff $u \approx 10^{-16}$. Therefore, the algorithm finds exact answers of perturbed problems where the perturbations in relative sense are of the same order as those induced by rounding error which is already in the data that is put to the algorithm. So if there is error induced by prior calculations/measurement errors in the entries of A and b that is significantly more than a modest multiple of u, then this dominates the error analysis rather than those present due to rounding or backward errors.

Since computations are done in double precision by default in Matlab (this means using 64 bits to represent any number in binary form which roughly translates to 16 digits after the decimal), you will have $s \approx 16$ in Matlab experiments if the only error in the entries of A and b is rounding error. This is also almost always the maximum possible value of s. (Wait for next week's lectures to know the details!)

1. 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)
   ```

   Modify the above code and repeat the experiment for the condition number in 1-norm and ∞-norm. [Note: The Matlab command `cond(H)` computes the 2-norm condition number of H. Type `help cond` for the details of finding the 1 and ∞-norm condition numbers.]

   Based on this graph formulate a conjecture as to the heuristic relationship between the condition number of H and its size.

2. Given a Hilbert matrix H, one of the aims of this exercise is to examine the effect of its ill conditioning on the accuracy of solutions of systems of equations $Hx = b$. Another aim 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. You can also use the `gepp` function written in earlier lab sessions 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 $n = 12$ and tabulate the results corresponding to $n = 8, 10, 12$, and determine correct digits in `x1, x2, x3`. Now answer the following questions:

(a) How many digits are lost in computing `x1, x2` and `x3`?

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

(c) 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 $s \approx 16$ in the *Rule-of-thumb* analysis.

3. The purpose of this experiment is to illustrate that a small value of the norm of the residual 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;
>> xt= H \ b;
>> r = H*xt-b;
>> disp( [norm(r)/norm(b) norm(x-xt)/norm(x)])
```

   What is your conclusion?

4. 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 and answer the following.

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

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

   (c) For which of the two methods is the *Rule-of-thumb* predicting the correct answer reasonably well? [Note that $s \approx 16$ in the *Rule-of-thumb* analysis]

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

5. The next exercise shows why small pivots should be avoided and also demonstrates the better numerical properties of Gaussian Elimination with Partial Pivoting over no pivoting in dealing with matrices with small entries in pivotal positions. Generate random matrices of different sizes (for example, $n = 20, 40, 80, 100$ etc.) with small entries in $(1, 1)$ positions. To do this just type

   ```
   >> A = rand(n); A(1,1) = 50*eps*A(1,1);
   ```

   Find L and U by using the `genp` code written in Lab session 1 and compute $\|L\|, \|U\|$ and $\frac{\|LU-A\|}{\|A\|}$ with respect to the 1, 2 and ∞-norms.

   Repeat the process with the L and U generated from your `gepp` code. The only difference is that this time you will have to compute $\frac{\|LU-PA\|}{\|A\|}$ instead of $\frac{\|LU-A\|}{\|A\|}$.

   Repeat this for all the different sizes and tabulate the values of the norms. Which of the two sets of LU decompositions produce the larger norms?