

UNIVERSITY COLLEGE DUBLIN

THE NATIONAL UNIVERSITY OF IRELAND

An Coláiste Ollscoile, Baile Átha Cliath

Ollscoil na hÉireann, Baile Átha Cliath

SUMMER EXAMINATIONS 2004

SCHDF0025 – H.Dip. Computational Science – Year 1

MATHEMATICAL PHYSICS

MAPH P401 — NUMERICAL ALGORITHMS

Professor E. Corrigan, FRS

Professor A.C. Ottewill

*Dr Derek O'Connor**

SOLUTIONS

Instructions for Candidates

Full marks for complete answers to any 5 questions

Time : 3 hours

Notes for Invigilators

Non-programmable calculators are permitted

1. GENERAL NOTES

1. Each question has the same weight (mark).
2. Each question has three parts (a), (b), (c). These may have different weights depending on their difficulty.
3. Most questions have been covered in the lectures and in the lecture, lab, and assignment notes which are on the class webpage :

<http://www.derekroconnor.net/na/na2col.html>

In the solutions below I will refer to these notes by chapter and page number.

4. The exam tries to discriminate between good, mediocre, and bad students by asking very specific questions of varying difficulty. There is no chance to ‘waffle’. It also asks questions based on the 6 laboratory exercises and 3 assignments to discriminate between those who have worked hard on the exercises and assignments, and those who have not.

2. SOLUTION NOTES

Question 1.

- (a). Describe what is meant by a floating-point number system and derive an expressions for the *machine epsilon* ϵ_M and *unit roundoff error* u in terms of the parameters of the system.
- (b). Is the fraction $\frac{1}{10}$ representable in the floating point system $F(b, p, L, U) = F(2, 6, -20, +20)$? If not, explain why and find the closest floating point number.
- (c). Why does the harmonic series $\sum_{k=1}^{\infty} 1/k$ converge in any finite-precision number system? Estimate the number of terms needed for convergence in the IEEE double-precision floating point number system $F(b, p, L, U) = F(2, 53, -1021, 1024)$

Solution 1. Chap. 2 & Lab Exer. 1

- (a). Chap. 2, pages 2.16 – 2.19

Roundoff error occurs when a number x falls between two adjacent floating point numbers. The maximum distance between these two floating point numbers is $\epsilon_M|x|$. Hence the maximum roundoff error is half the distance between them, i.e.,

$$\frac{|x - fl(x)|}{|x|} = \frac{1}{2} \frac{\epsilon_M|x|}{|x|} = \frac{1}{2}\epsilon_M = u.$$

Thus we see that the unit roundoff error u is half machine epsilon, i.e.,

$$u = \frac{\epsilon_M}{2} = \frac{1}{2} b^{1-p}$$

There is some confusion in the textbooks about the definitions of machine epsilon, ϵ_M , and unit roundoff, u . We use Higham's definitions (Higham, 1996, page 41) for ϵ_M and u . Others (Trefethen & Bau and Golub & Van Loan) define $\epsilon_M = u = \frac{1}{2}b^{1-p}$.

- (b). Chap. 2, page 2.9, and

$(\frac{1}{10})_{10} = (0.000110011001100\dots)_2 = (0.110011001100\dots)_2 \times 2^{-3}$ is a recurring binary number and so it is not representable in any binary f.p. system. The nearest number after rounding to $p = 6$ bits is $(0.110011)_2 \times 2^{-3} = (1/2^1 + 1/2^2 + 0 + 0 + 1/2^5 + 1/2^6) \times 2^{-3} = 0.099609375$ which has a relative rounding error $|fl(x) - x|/|x| = 0.00390625 \approx 0.004 = 0.4\%$. The maximum relative error we can get in this system is $u = \frac{1}{2}2^{1-6} = 2^{-6} = 0.015625 \approx 1.6\%$

- (c). An infinite series that is divergent in theory can have a finite sum in finite-precision, floating-point arithmetic. The sum ceases to change when the next term to be added is negligible relative to the partial sum thus far, and hence the computed sum is finite.

Finding the number of terms to convergence is not simple because we do not know the finite-precision limit. Let $S_n = \sum_{k=1}^n 1/k$ be the partial sum of n terms at convergence.

This means that $fl(S_n + 1/(n+1)) = S_n$. This implies that $(1/(n+1))/S_n \leq \epsilon_M$. Thus we have, roughly,

$$n \approx 1/\epsilon_M S_n = b^{(p-1)}/S_n = 2^{52}/S_n \approx 10^{16}/S_n \approx 10^{14}.$$

Note that *underflow* of $1/k$ is not the problem here because $1/10^{14} = 10^{-14}$ is far from the underflow threshold : the magnitude of $fl(1/k)$ relative to $fl(S_{k-1})$ is effectively 0.

The main point here is that it will take many terms to reach convergence with double-precision arithmetic. Lab. Exercise 1 asked the students to calculate, in MATLAB, the partial sums for $n = 100, 1000, 1000000$. There is also a class webpage link to an online Java program at the University of Illinois that allows them to calculate the partial harmonic sums for different FP precisions. See http://www.cse.uiuc.edu/eot/modules/floating_point/infinite_series [I do not expect many students to get the second part of (c).]

Question 2.

- (a). Prove that in any floating point system $F(b, p, -, -)$, without guard digits, the relative error in $fl(x - y)$ can be as large as $b - 1$. This is 100% for $b = 2$ and 900% for $b = 10$.

[HINT : Let $x = .(100 \dots 0) \times b^1$ and $y = .((b-1)(b-1) \dots (b-1)) \times b^0$]

- (b). The following formulas for *sample variance* are mathematically equivalent :

$$s_n^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2, \text{ where } \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i, \text{ and } s_n^2 = \frac{1}{n-1} \left(\sum_{i=1}^n x_i^2 - \frac{1}{n} \left(\sum_{i=1}^n x_i \right)^2 \right).$$

Explain why the second formula is a poor choice in the presence of roundoff error. Calculate s_n^2 using both methods with $x = (10000, 10001, 10002)^T$ and unit roundoff error $u = 6 \times 10^{-8}$, i.e., 7 digits precision. (from Higham, 1996)

- (c). You have been asked by the finance journal *Modern Enronics* to write a review of a new financial software package from *SoftSell* that evaluates 'exotic options'. In reviewing the MATLAB source code for it you come across the line

```
if abs(xnew - xold) < 1.0e-6 ... etc.
```

As a prospective master of computational science what would you write in your review? Explain.

Solution 2. Chaps. 2 & 4

- (a). From the Lecture Notes page 2.22 we have :

Let $x = .(100 \dots 0) \times b^1$ and $y = .((b-1)(b-1) \dots (b-1)) \times b^0$. The exact difference $(x - y)$ is b^{-p} . When performing the floating point subtraction the digits of y are aligned with x by shifting them to the right. Thus we get

$$fl(x - y) = .(100 \dots 0) \times b^1 - .(0(b-1) \dots (b-1)) \times b^1 = (00 \dots 1) \times b^1 = b^{1-p}$$

$$\text{The relative error is } \frac{|(x - y) - fl(x - y)|}{|x - y|} = \frac{|b^{-p} - b^{1-p}|}{b^{-p}} = |1 - b| = b - 1, \quad b \geq 2.$$

- (b). For certain values of the vector x the two squared terms in the second formula will be nearly equal and will lead to cancellation error. For $x = (10000, 10001, 10002)^T$ we have, using the first formula, $\bar{x} = fl((10000+10001+10002)/3) = fl(30003/3) = 10001.0$ and $s_n^2 = fl((1 + 0 + 1)/2) = 1.0$

The second formula gives $fl(\sum_{i=1}^n x_i^2) = fl(300060005) = 0.3000600 \times 10^9$ and

$fl((\sum_{i=1}^n x_i)^2/n) = fl(900180009/3) = fl(300060003) = 0.3000600 \times 10^9$.

Thus $s_n^2 = fl((0.3000600 \times 10^9 - 0.3000600 \times 10^9)/2) = 0.0$ — completely wrong.

In the second formula the squaring operation gives results that require 9 digits for full accuracy but with $u = 6 \times 10^{-8}$ we have only about 6 or 7 digits of precision. The rounding of these intermediate results causes catastrophic cancellation in the final subtraction operation. The real culprit here is the squaring of large numbers, which forces rounding. The squaring in the first formula is done on small numbers.

- (c). I would write in the review : ‘Don’t invest in *SoftSell*’. From the *Lecture Notes* page 4.20 we have :

Let us examine an x -convergence test that is too often seen in textbooks and amateur programs :

$$\text{IF } |x_k - x_{k-1}| \leq 10^{-6} \text{ THEN STOP}$$

If the magnitudes of x_{k-1} and x_k are of the order 10^{12} then the floating point number spacing around x_k is $\epsilon_M \times |x_k| = 10^{-16} \times 10^{12} = 10^{-4}$. This means that the two floating point numbers x_k, x_{k-1} can never be closer than 10^{-4} and so passing this test is impossible. If this was the only convergence test used in the algorithm then it would never stop.

If $|x_k| = 10^{-3}$ then the floating point number spacing around x_k is $10^{-16} \times 10^{-3} = 10^{-19}$ and so the test is valid but crude by a factor of 10^{13} .

This example shows that we must account for floating point number spacing when choosing ϵ_x , the x -convergence parameter.

Question 3.

- (a). Write a MATLAB function that implements the Bisection method for zero-finding. Discuss the pitfalls a naïve implementer of this method may face. How many function evaluations/iteration does your implementation require?
- (b). Demonstrate the action of the Bisection Method on the function $f(x) = (x - 1.0)^2 - 3.0$. Start with $[a, b] = [0, 20]$ and continue until this interval has been reduced to 1/10th of its starting length. Use round-to-nearest, 4-digit decimal arithmetic.
- (c). Analyse the Bisection Method. In particular, find how many iterations are required to reduce the initial interval of length $|b - a|$ to a length ϵ . Does the algorithm work if $b < a$? Explain and demonstrate.

Solution 3. Chap. 4 & Assign. 2(a). From the *Assignment 2 Solution Notes* :

```

%-----
function [z,flag,itors] = Bisect(a,b,tol,Maxits,flag);
%-----
flag = inf;
z = inf;
itors = 0;
fa = FUN(a); fb = FUN(b);
if sign(fa) == sign(fb)
    return
% cannot proceed, signs wrong.
end;
for iters = 1:Maxits
    m = a + 0.5*(b-a);
    fm = FUN(m);
    if (fm == 0.0)
        z = m;
        flag = 1;
        return
    end;
    if abs(a-b) <= tol + eps * abs(a)
        z = m;
        flag = 0;
        return
    end;
    if sign(fa) == sign(fm)
        a = m;
        fa = fm;
    else
        b = m;
        fb = fm;
    end;
end; % end of iteration loop
flag = -1; % run out of iterations
% ---- end of Bisect -----
end;

```

Pitfalls : (i) No initial sign test. (ii) Bad x -convergence test (see 2(c) above). (iii) Too many function evaluations/iteration. Only 1 f -eval above. (iv) No fail-safe for iters loop.

(b).

BISECTION ALGORITHM

k	a	b	$f(a)$	$f(b)$	m	$f(m)$
0	0	20	-2	358	10	78
1	0	10	-2	78	5	13
2	0	5	-2	13	2.5	-0.75
3	2.5	5	-0.75	13	3.75	4.563
4	2.5	3.75	-0.75	4.563	3.125	1.516
5	2.5	3.125	-0.75	1.516	2.813	0.2870

After 4 iterations the interval is $(3.75 - 2.5) = 1.25$, so stop. The exact answer is the solution of $(x - 1.0)^2 - 3.0 = 0$, i.e., $x = 1 \pm \sqrt{3}$ or $x = 2.73205081, -0.732050808$.

(c). From the *Lecture Notes* page 4.6 we have :

It is obvious that the interval is halved (contracted) each iteration. Hence $e_k = \frac{1}{2} e_{k-1}$.

Thus the Bisect Algorithm has order 1 or linear convergence and it gains 1 bit of precision for each iteration. The error after k iterations is $e_k = 2^{-k}|b - a|$. The algorithm stops after k iterations with $2^{-k}|b - a| = \epsilon$, or $2^k \epsilon = |b - a|$, or $2^k = |b - a|/\epsilon$. Taking logs of both sides we get $k = \left\lceil \log_2 \frac{|b - a|}{\epsilon} \right\rceil$.

The algorithm works whether $a < b$ or $b < a$ because it is symmetrical with respect to these parameters, i.e., we can substitute a for b and *vice versa* : (i) $m = a + 0.5 * (b - a) = b + 0.5 * (a - b)$ and (ii) $\text{abs}(a - b) = \text{abs}(b - a)$.

Question 4.

- (a). Write an algorithm for the LU Decomposition of the matrix A without pivoting. If A_k is the matrix after $k - 1$ steps of the decomposition process, what is the matrix M_k such that $A_{k+1} = M_k A_k$? What is M_k^{-1} ?
- (b). Explain what *pivoting* is and demonstrate the need for it by solving $Ax = b$ *without* and *with* pivoting.

$$A = \begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \epsilon < 10^{-16} = \epsilon_M.$$

Assume IEEE double-precision floating point arithmetic.

- (c). Given the factors L and U , where $LU = A$, show how the system of equations $A^2x = b$ could be solved in $O(n^2)$ time. [HINT : LULU]

Solution 4. Chaps. 2 & 5

- (a). See *Lecture Notes* pages 5.16, 5.20, 5.21.
- (b). From the *Lecture Notes* page 5.25 we have:
Need for Pivoting 2. Here is a more general example that clearly shows the need for pivoting when using finite precision arithmetic. The problem is

$$\text{Solve } \begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \epsilon < 10^{-16}, \quad \text{using d.p. floating point arithmetic.}$$

Let us find the exact solution first so that we can see what is happening later.

With no pivoting we get

$$L = \begin{bmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{bmatrix}, \quad U = \begin{bmatrix} \epsilon & 1 \\ 0 & 1 - 1/\epsilon \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1/(1 - \epsilon) \\ 1 - \epsilon/(1 - \epsilon) \end{bmatrix}.$$

With pivoting we get

$$L = \begin{bmatrix} 1 & 0 \\ \epsilon & 1 \end{bmatrix}, \quad U = \begin{bmatrix} \epsilon & 1 \\ 0 & 1 - 1/\epsilon \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1/(1 - \epsilon) \\ 1 - \epsilon/(1 - \epsilon) \end{bmatrix}.$$

The correctly-rounded exact answer is

$$fl \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} = \begin{bmatrix} 1/fl(1 - \epsilon) \\ fl(1 - \epsilon/fl(1 - \epsilon)) \end{bmatrix} = \begin{bmatrix} 1/1 \\ fl(1 - \epsilon) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \text{because } \epsilon < \epsilon_M.$$

The hat (^) over x indicates that it is not exact.

Now we perform the same calculations in floating point arithmetic.

With no pivoting we get

$$A = A^{(1)} = \begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix}, M_1 = \begin{bmatrix} 1 & 0 \\ -1/\epsilon & 1 \end{bmatrix}, \hat{A}^{(2)} = M_1 A^{(1)} = \begin{bmatrix} \epsilon & 1 \\ 0 & fl(1 - 1/\epsilon) \end{bmatrix} = \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix}.$$

Now we have

$$L = M_1^{-1} = \begin{bmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{bmatrix}, \quad \text{and} \quad \hat{U} = A^{(2)} = \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix}.$$

with the hat (^) over U indicating that it is not exact. The product $L\hat{U}$ should give A but

$$L\hat{U} = \begin{bmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{bmatrix} \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix} = \begin{bmatrix} \epsilon & 1 \\ 1 & 0 \end{bmatrix} = \hat{A} \neq A.$$

If we use the factors L and \hat{U} to solve the original problem we get

$$L\hat{U}\hat{x} = b : \quad \begin{bmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{bmatrix} \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix} \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

Forward substitution gives

$$\hat{U}\hat{x} = L^{-1}b = Mb : \quad \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix} \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -1/\epsilon & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ fl(2 - 1/\epsilon) \end{bmatrix} = \begin{bmatrix} 1 \\ -1/\epsilon \end{bmatrix}.$$

Back substitution gives

$$\hat{x} = \hat{U}^{-1}Mb = A^{(2)}Mb : \quad \begin{bmatrix} \epsilon & 1 \\ 0 & -1/\epsilon \end{bmatrix} \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -1/\epsilon \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} = \begin{bmatrix} (1 - 1)/\epsilon \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

The second element \hat{x}_2 is the correctly-rounded exact value but \hat{x}_1 is has no digits correct and is completely wrong.

With pivoting we get

$$A = A^{(1)} = \begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix}, \quad P_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad P_1 A^{(1)} = \begin{bmatrix} 1 & 1 \\ \epsilon & 1 \end{bmatrix}, \quad M_1 = \begin{bmatrix} 1 & 0 \\ -\epsilon & 1 \end{bmatrix},$$

$$\hat{A}^{(2)} = M_1 P_1 A^{(1)} = \begin{bmatrix} 1 & 1 \\ 0 & fl(1 - \epsilon) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \hat{U} \quad \text{and} \quad L = M_1^{-1} = \begin{bmatrix} 1 & 0 \\ \epsilon & 1 \end{bmatrix}.$$

Check that $L\hat{U} = PA$:

$$\begin{bmatrix} 1 & 0 \\ \epsilon & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \epsilon & fl(1 + \epsilon) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \epsilon & 1 \end{bmatrix} = PA.$$

If we use these pivot factors L and \hat{U} to solve the original problem we get

$$L\hat{U}\hat{x} = Pb : \quad \begin{bmatrix} 1 & 0 \\ \epsilon & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}.$$

Forward substitution gives

$$\hat{U}\hat{x} = L^{-1}b = MPb : \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\epsilon & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ fl(1-2\epsilon) \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}.$$

Back substitution gives

$$\hat{x} = \hat{U}^{-1}MPb = A^{(2)}MPb : \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

the exact answer rounded to machine precision.

[I do not expect the students to go into this level of detail but I do expect them to solve the equations symbolically, with and without pivoting, and to show where rounding occurs.]

(c). We have $A = LU$ and so we wish to solve $A^2x = LULUx = b$. Do the following steps :

1. Solve $Lz = b$ using forward substitution. $z = L^{-1}b$.
2. Solve $Uy = z$ using back substitution. $y = U^{-1}z = U^{-1}L^{-1}b$.
3. Solve $Lw = y$ using forward substitution. $w = L^{-1}y = L^{-1}U^{-1}L^{-1}b$.
4. Solve $Ux = w$ using back substitution. $x = U^{-1}w = U^{-1}L^{-1}U^{-1}L^{-1}b$.

These steps give the correct solution because $x = (LULU)^{-1}b = U^{-1}L^{-1}U^{-1}L^{-1}b$. Forward and back substitution are both $O(n^2)$ so the total is $O(n^2)$.

[Only the better students will get this part of the question]

Question 5.

- (a). If $Ax = b$ then determine the *relative change in x* due to a change δA in A . Define *condition number* of a matrix A and comment on its significance.
- (b). The two-by-two diagonal system

$$Ax = b : \begin{bmatrix} \alpha & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \alpha \\ 1 \end{bmatrix}, \quad \alpha > 1,$$

has the computed solution $\hat{x}_1 = fl(b_1/a_{11})$, $\hat{x}_2 = fl(b_2/a_{22})$ which is accurate to machine precision, i.e., $\|\hat{x} - x\|/\|x\| \leq \epsilon_M$. If the matrix $A = \text{diag}(\alpha, 1)$ is changed to $A + \delta A = \text{diag}(\alpha(1 + \epsilon_M), 1 + \epsilon_M)$, calculate the relative change in \hat{x} , for $\alpha = 2$ and $\alpha = 2^{100}$. Use the $\|\cdot\|_\infty$ norm. Comment.

- (c). Explain why the residual $r = b - A\hat{x}$ is not a good measure of the error $x - \hat{x}$.

Solution 5. Chap. 5

(a). From the *Lecture Notes* page 5.31. If A is perturbed by δA then we have

$$(A + \delta A)(x + \delta x) = b$$

$$Ax + A\delta x + \delta Ax + \delta A\delta x = b$$

$$A\delta x = -\delta A(x + \delta x)$$

$$\delta x = -A^{-1}\delta A(x + \delta x)$$

Taking norms and using the triangle inequality we have

$$\begin{aligned} \|\delta x\| &= \|A^{-1}\delta A(x + \delta x)\| \\ &\leq \|A^{-1}\| \|\delta A\| (\|x\| + \|\delta x\|) \end{aligned}$$

$$\|\delta x\| (1 - \|A^{-1}\| \|\delta A\|) \leq \|A^{-1}\| \|\delta A\| \|x\|$$

$$\begin{aligned} \frac{\|\delta x\|}{\|x\|} &\leq \frac{\|A^{-1}\| \|\delta A\|}{1 - \|A^{-1}\| \|\delta A\|} \\ &= \frac{\|A\| \|A^{-1}\| \|\delta A\|}{(1 - \|A^{-1}\| \|\delta A\|) \|A\|} \end{aligned}$$

If $\|A^{-1}\| \|\delta A\| < 1$ we have

$$\frac{\|\delta x\|}{\|x\|} \leq \text{cond}(A) \frac{\|\delta A\|}{\|A\|}$$

Theorem 1.1 (DISTANCE TO A SINGULAR MATRIX). If A is non-singular, and

$$\frac{\|\delta A\|}{\|A\|} < \frac{1}{\text{cond}(A)}$$

then $A + \delta A$ is also non-singular. □

This theorem tells us the condition number measures the distance from A to the nearest singular matrix.

[I expect students to use a less detailed derivation of the bound above. I do not require the theorem.]

(b). We have, from above,

$$\frac{\|\delta x\|}{\|x\|} \leq \text{cond}(A) \frac{\|\delta A\|}{\|A\|}.$$

Using the ∞ -norm we get $\text{cond}(A) = \|A\| \|A^{-1}\| = \alpha$, $\|\delta A\| = \alpha\epsilon_M$, and $\|A\| = \alpha$. Thus we have

$$\frac{\|\delta x\|}{\|x\|} \leq \alpha \frac{\alpha\epsilon_M}{\alpha} = \alpha\epsilon_M.$$

This upper bound (and $\text{cond}(A)$) can be made arbitrarily large (let $\alpha = 2^{100}$), yet the calculated solution is

$$\hat{x} = \begin{bmatrix} fl(\alpha/\alpha(1 + \epsilon_M)) \\ fl(1/(1 + \epsilon_M)) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Conclusion : The $\text{cond}(A)$ bound can be very crude.

- (c). From the *Lecture Notes* pages 5.34,5.35. We now examine the relationship between r and errors in x . Let \hat{x} be the computed solution to $Ax = b$. Then

$$\delta x = x - \hat{x}$$

$$r = b - A\hat{x}$$

$$A\delta x = Ax - A\hat{x} = b - A\hat{x} = r$$

$$\delta x = A^{-1}r$$

$$\|\delta x\| \leq \|A^{-1}\| \|r\|$$

$$\frac{\|\delta x\|}{\|x\|} \leq \|A^{-1}\| \frac{\|r\|}{\|x\|}$$

$$= \|A\| \|A^{-1}\| \frac{\|r\|}{\|A\| \|x\|}$$

$$\leq \|A\| \|A^{-1}\| \frac{\|r\|}{\|Ax\|}$$

$$= \|A\| \|A^{-1}\| \frac{\|r\|}{\|b\|}$$

Thus

$$\frac{\|\delta x\|}{\|x\|} \leq \text{cond}(A) \frac{\|r\|}{\|b\|}$$

If A is ill-conditioned then small $\|r\|$ does not imply small $\|\delta x\|/\|x\|$.

LUP Decomposition tends to give solutions with small residuals but can give large errors in \hat{x} if A is ill-conditioned.

Question 6.

- (a). Explain the difference between *dense* and *sparse* storage of sparse matrices. Draw the sparse storage diagram for the matrix below, using the storage scheme of *Assignment 3*.

$$A_{6 \times 6} = \begin{bmatrix} 0 & 0 & 0 & 21 & 35 & 0 \\ 0 & 34 & 0 & 55 & 0 & 51 \\ 0 & 0 & 0 & 0 & 95 & 0 \\ 66 & 0 & 96 & 43 & 0 & 0 \\ 0 & 0 & 0 & 0 & 82 & 71 \\ 0 & 31 & 0 & 51 & 0 & 0 \end{bmatrix}$$

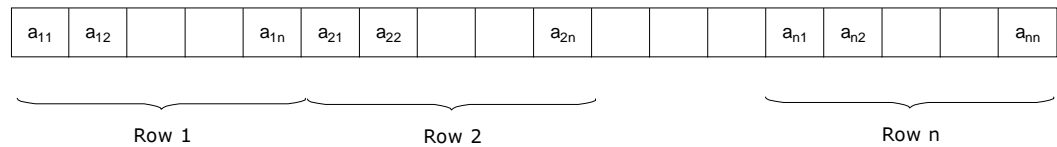
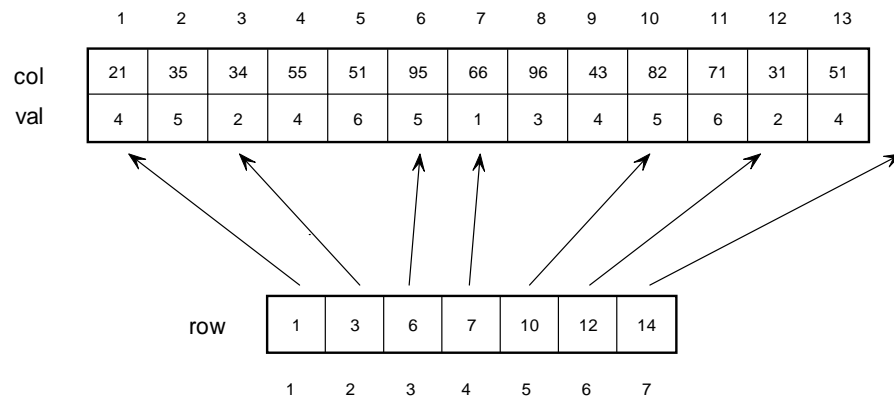
- (b). Describe the *Successive Over-Relaxation* algorithm for solving $Ax = b$ when A is sparse. How many operations per iteration does it require?
- (c). Explain why it is not necessary to maintain two vectors to implement the x -convergence test

$$\text{IF } \|x_{old} - x_{new}\|_{\infty} \leq tol + 4.0 * \epsilon_M \|x_{new}\|_{\infty} \text{ THEN}$$

in the SOR algorithm above.

Solution 6. Chap. 5 and Assign. 3

- (a). The dense storage of an $n \times n$ matrix requires n^2 boxes as shown in Figure 1. All n^2 values of the matrix are stored. Sparse storage schemes store the t non-zeros only, along with indexing information for these non-zeros. If $t \ll n^2$ then much space and processing time can be saved. Many real problems are modelled by huge matrices with only a few non-zeros per row. (The GOOGLE page rank algorithm uses adjacency matrices of size $n = 10^9$.) Hence the number of non-zeros $t = cn$ and so t is $O(n)$. The sparse storage scheme in Figure 2 requires $2t + n + 1$ space.

**Figure 1.1.** Dense Matrix Storage Q6(a).**Figure 1.2.** Sparse Matrix Storage Q6(a).

- (b). From the *Lecture Notes* page 5.46 we have

```

algorithm Sparse-SOR( $a, b, n, maxits, tol, \omega$ )
The sparse matrix structure  $a$  contains  $val, col, row$   $d$ 
 $k := 0$ 
while (NOT CONVERGED) AND  $k \leq maxits$  do
  for  $i := 1$  to  $n$  do
     $sum := 0$ 
    for  $j := row[i]$  to  $row[i + 1] - 1$  do
       $sum := sum + val[j] \times x[col[j]]$ 
    endfor  $j$ 
     $x[i] := x[i] + \omega \times (b[i] - sum) / d[i]$  (*)
  endfor  $i$ 
endwhile  $k$ 
return ( $x, k$ )
endalg Sparse-SOR

```

- (c). From the *Assignment 3 Solution Notes* :
To implement the x -convergence test

$$\text{IF } \|x_{old} - x_{new}\|_{\infty} \leq tol + 4.0 \epsilon_M \|x_{new}\|_{\infty} \text{ THEN}$$

we do not need two vectors, nor do we need an $O(n)$ `norm(x)` function. We must remember that in an algorithm or program statement such as (*) above, the $x[i]$ on the left of an assignment operator (`:=`) is a ‘new’ value while that on the right is an ‘old’ value. So, apart from the convergence test, we need one vector only.

The term $\|x_{old} - x_{new}\|_{\infty}$, on the face of it, seems to require two vectors. However this term is really $\|\delta x\|_{\infty}$, where $x_{new} = x_{old} + \delta x$. Looking at the (*) statement above we see that $\delta x_i = \omega * (b[i] - sum) / d[i]$. If we store this (scalar) value in the variable `delx` we can use it to incrementally calculate $\|x_{old} - x_{new}\|_{\infty}$. Likewise, we can incrementally calculate $\|x_{new}\|_{\infty}$, and so we do not need a separate `norm(x)` function. These ideas are implemented in the updated *Sparse-SOR* algorithm below.

algorithm *Sparse-SOR*($a, b, n, maxits, tol, \omega$)

The sparse matrix structure a contains val, col, row, d

```

for  $k := 1$  to  $maxits$  do
   $normx := 0; dnormx := 0$ 
  for  $i := 1$  to  $n$  do
     $sum := 0$ 
    for  $j := row[i]$  to  $row[i + 1] - 1$  do
       $sum := sum + val[j] \times [col[j]]$ 
    endfor  $j$ 
     $delx := \omega \times (b[i] - sum) / d[i]$  (*)
     $x[i] := x[i] + delx$ 
     $normx := \max(normx, \text{abs}(x[i]))$ 
     $dnormx := \max(dnormx, \text{abs}(delx))$ 
  endfor  $i$ 
  if  $dnormx \leq tol + 4 \times \epsilon_M \times normx$  then return  $(x, k)$ 
endfor  $k$ 
endalg Sparse-SOR

```

Notice that $normx$ and $dnormx$ are incrementally calculated in the i -loop for each iteration of the k -loop. If we want the $\|\cdot\|_1$ norm then we just change ‘max’ to ‘add’ in the norm statements above.

Question 7.

- (a). Show how the polynomial

$$p_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n,$$

can be evaluated in $O(n)$ operations.

- (b). Derive an expression $N_{ops}(m, n, p, q)$ for the number of operations need to form the matrix product $D = ABC$, where A is $m \times n$, B is $n \times p$, and C is $p \times q$, and D is $m \times q$. Although matrix multiplication is associative, i.e., $A(BC) = (AB)C$, show that

$$N_{ops}(A(BC)) \neq N_{ops}((AB)C), \text{ in general.}$$

That is, $N_{ops}(m, n, p, q)$ depends on the order in which the product ABC is formed or *parenthesized*. Give some examples to illustrate your answer.

- (c). If A is an $n \times n$ matrix the naïve calculation of A^k requires $(k-1)n^3$ ops. If $k = 2^p$ show that there is a better way which requires $\log_2 k n^3$ ops. Write the MATLAB code for this method (one line).

Solution 7. Chaps. 4, 5 & Lab Exer. 6

- (a). From the
- Lecture Notes*
- page 4.24 we have :

If we evaluate $p_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$ term-by-term then this will require $(1 + 2 + 3 + \cdots + n - 1) = O(n^2)$ multiplications. We can re-arrange the polynomial as follows :

$$\begin{aligned} p_n(x) &= a_0 + x(a_1 + a_2x + a_3x^2 + \cdots + a_nx^{n-1}) \\ &= a_0 + x(a_1 + x(a_2 + a_3x + \cdots + a_nx^{n-2})) \\ &\quad \dots \\ &= a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + x(a_n)) \cdots)). \end{aligned}$$

This nested multiplication expression requires only n multiplications. We implement this idea in the algorithm below, where a and b are arrays of coefficients, n is the degree, and α is the point at which the polynomial is to be evaluated.

algorithm EvalPoly (a, α, n, b)

```

    b[n] := a[n]
    for k := n - 1 downto 0 do
        b[k] := a[k] + alpha * b[k + 1]
    endfor
endalg EvalPoly

```

This algorithm has a single k -loop which has a single addition and multiplication. It obviously requires only n operations.

(b). From the *Lab Exercise 6 Solution Notes* we have :

Standard matrix multiplication, $C = AB$, where A is $m \times n$, B is $n \times p$, and C is $m \times p$ is as follows :

$$c_{ij} = \sum_{k=1}^n a_{ik} \times b_{kj}, \quad i = 1, 2, \dots, m, \quad j = 1, 2, \dots, p.$$

There are $m \times p$ elements c_{ij} and each requires the summation

$$a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + \dots + a_{in} \times b_{nj},$$

which requires n mults and $n - 1$ adds. Hence we get a total of $2mnp - mp = O(mnp)$ operations.

The matrix-triple multiplication operation $D = ABC$, where A is $m \times n$, B is $n \times p$, and C is $p \times q$, and D is $m \times q$, is defined in terms of the matrix-pair multiplication above. This gives two possible orders of multiplication :

$$D_1 = (AB)C \quad \text{or} \quad D_2 = A(BC).$$

Mathematically, D_1 and D_2 are identical, but computationally they are not. Using $O(mnp)$ for matrix-pair multiplication we have

$$N_{ops}(A(BC)) = N_{ops}(R = BC) + N_{ops}(D = AR) = O(npq) + O(mnq)$$

$$N_{ops}((AB)C) = N_{ops}(R = AB) + N_{ops}(D = RC) = O(mnp) + O(mpq)$$

We will drop the $O(-)$ formalism and simply say that

$$N_{ops}(A(BC)) = N_1 = npq + mnq \quad \text{and} \quad N_{ops}((AB)C) = N_2 = mnp + mpq.$$

It is not possible to say in general when these two functions have different or equal values.

(c). From the *Lab Exercise 6 Solution Notes* we have :

Calculating A^k . The naïve calculation $R = A \times (A \times (\dots \times A(A \times A)) \dots)$ requires $(k - 1)$ matrix multiplications, each of which requires n^3 ops. This gives a total of $(k - 1)n^3$ ops. If $k = 2^p$ there is a better way :

$$A^2 = A \times A, \quad A^4 = A^2 \times A^2, \quad \dots, \quad A^{2^p} = A^{2^{p-1}} \times A^{2^{p-1}},$$

which requires p multiplications or $pn^3 = \log_2 k n^3$ ops.

The MATLAB program is trivial

```
R = A;   for i = 1 : log2(k),   R = R * R;   end;
```

The naïve program is

```
R = A;   for i = 1 : k,           R = A * R;   end;
```

1.3 Criticisms

Just two general criticisms :

1. Correct indexing in Numerical Linear Algebra software is very important. Near misses will not do. Use the convention that i indexes the rows and j indexes the columns of a vector or matrix. Where a triply-nested loop is needed use k to index the outer-most or inner-most loop, whichever is appropriate. Remember : the hardest part of writing LA software is the indexing.
2. Confusion about `minreal`, machine epsilon `eps` and underflow. Underlying this confusion is a general lack of understanding of floating point numbers and arithmetic. If you do not understand floating point arithmetic you are not a computational scientist — no matter what the letters after your name say.

1.4 Exam Results

Seventeen(17) students took this exam. The final mark comprised 10% Lab. Exercises, 15% Assignments, and 75% this exam. Note that these weights are now 15%, 15%, 70%.

The statistics were Min: 19, Avg: 50, Max: 75, SDev: 15