

CS127-Introduction to Scientific Computing

First Exam: Solutions

October 22, 2009

1. The exponential function $y = e^x$ can be evaluated as the sum of an infinite series:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{j=0}^{\infty} \frac{x^j}{j!}.$$

(Note that $0! = 1$, so the $j = 0$ term in the right-hand expression makes sense.)

(a) The MATLAB script below is designed to compute an approximation to e^{12} by summing the first 21 terms of this series. The script contains two errors: One is a syntax error that prevents it from executing. The other is an awkward use of a MATLAB control structure that causes the script to give an incorrect result. Explain what the errors are and how to correct them.

```
sum=1;
fact=1;
power=1;
for j=1:20
    j=j+1;
    power=12*power;
    fact=j*fact;
    sum=sum+power/fact;
sum
```

Solution The `for` statement requires a matching `end`, which should be placed just after the last indented line of code. The increment `j=j+1;` causes the program to give an incorrect result, and should be removed. As a general rule, you should not change the value of the control variable of a `for` statement within the loop, as it is incremented automatically.

(b) The same calculation can be performed by using vector operations, typing just a couple of commands at the prompt, and without using any looping control structures like `for`. Show how do do this. (It is helpful to know that MATLAB has a built-in function `factorial`, which can be applied to vectors.)

Solution.

```
>> v=0:20;  
>> sum((12.^v)./factorial(v))
```

```
ans =  
  
1.6087e+05
```

(c) Suppose we applied the following strategy to evaluating the infinite sum above: Keep adding terms until it makes no difference—that is, *when two successive values of the variable sum are equal*, stop adding. Show how to revise the script above so that it carries out this algorithm. (HINT: You will need, among other things, to keep track of both the present value of the sum and the previous one.)

Solution. Here is the code. Generally speaking, it is a bad idea to run an iterative numerical algorithm until two successive estimates of the answer become equal, but in this instance we reach this situation as soon as the ratio of the next term to the accumulated sum is less than machine epsilon, so we don't get an unusually large number of terms—the algorithm terminates after 52 steps.

```
sum=1;  
oldsum=0;  
fact=1;  
power=1;  
j=1;  
while oldsum~=sum  
    oldsum=sum;  
    power=12*power;  
    fact=j*fact;  
    j=j+1;  
    sum=sum+power/fact;  
end  
sum
```

2. Consider the matrix

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 3 & 4 \\ 11 & 12 & 13 \end{pmatrix}.$$

(a) Find a pair of 3×3 matrices L and U , where L is lower triangular with 1's on the diagonal, and U is upper triangular so that $A = LU$. There is only one answer to this question—the fact that no permutation matrix is specified means that when you perform Gaussian Elimination to find this factorization, there will be no row interchanges

Solution.

Here are the successive matrices:

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 11 & 12 & 13 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{pmatrix}$$

The last matrix is the desired U . L is the matrix obtained by placing the multipliers we used during elimination (2,11,and 1) below the diagonal of the identity matrix.

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 11 & 1 & 1 \end{pmatrix}$$

(b) Recall that an $n \times n$ matrix B is *nonsingular* if the linear system $B\mathbf{x} = \mathbf{b}$ has a unique solution for every choice of the n -dimensional column vector \mathbf{b} , and *singular* if each such system has either no solution or infinitely many solutions. Is the matrix A above singular or nonsingular? Explain?

Solution. A is singular if and only if the matrix U we obtain at the end of Gaussian elimination is singular. Because the last row of U is all zeros, $U\mathbf{x} = \mathbf{b}$ never has a unique solution—it will have infinitely many solutions if the last component of \mathbf{b} is zero, and no solutions otherwise.

(c) I tried to do part (a) of this problem in MATLAB, and found the following:

```
>> a=[1 1 1;2 3 4;11 12 13]
```

```
a =
```

```
1    1    1
```

```
    2    3    4
   11   12   13
```

```
>> [L,U]=lu(a)
```

```
L =
```

```
  0.090909090909091  -0.111111111111111  1.000000000000000
  0.181818181818182   1.000000000000000           0
  1.000000000000000           0           0
```

```
U =
```

```
 11.00 12.00 13.00
  0 0.818 1.636
  0 0 -0.00000
```

Why is this different from the answer you obtained in (a)?

Solution. The `lu` function in MATLAB interchanges rows in order to pivot on the largest value in a column. So, for example, the first step with the above matrix will be to interchange the third and the first row. These row interchanges are recorded in the form of the matrix L printed above.

(d) The problems above suggest that you can test to see whether a square matrix is singular or not by testing the lower right entry of the upper-triangular part of its LU-decomposition. Write a MATLAB function

```
function v = nonsingular(M,n)
```

that takes as parameters a square matrix M and its number of rows n , and returns 1 if the matrix is determined to be nonsingular, and 0 if it is determined to be singular. The function code is very short—don't worry about checking whether M is square and really has n rows.

Solution. It's just a one-liner:

```
function v = nonsingular(M,n)
v=(M(n,n)~=0);
```

(e) Assuming that the function above has been written, write the MATLAB command or commands you would type to set a variable `sing` to 0 if the matrix A defined at the start of this problem is singular, and to 1 if A is nonsingular.

Solution.

```
A=[1 1 1;1 2 3;11 12 13];  
sing=nonsingular(A,3);
```

(f) I tried (d) and (e), did everything right, and got the wrong answer as to the singularity of A . Why?

Solution. Because there is some roundoff error, the lower-right entry of U will seldom turn out to be exactly equal to 0. You can see this in the version of U MATLAB printed above: The last entry is written as `-0.00000` which indicates that MATLAB actually computed a very small negative value.

3. (a) Find a polynomial $p(x) = ax^2 + bx + c$ whose graph passes through the points $(0, 0)$, $(1, 1)$ and $(3, 1)$. How many solutions are there to this problem? (The problem has been designed to make the arithmetic very easy so that you can solve this by hand.)

Solution. Easily $c = 0$ because the graph passes through $(0, 0)$. When we plug the other two points in we get the system of equations

$$a + b = 1$$

$$9a + 3b = 1$$

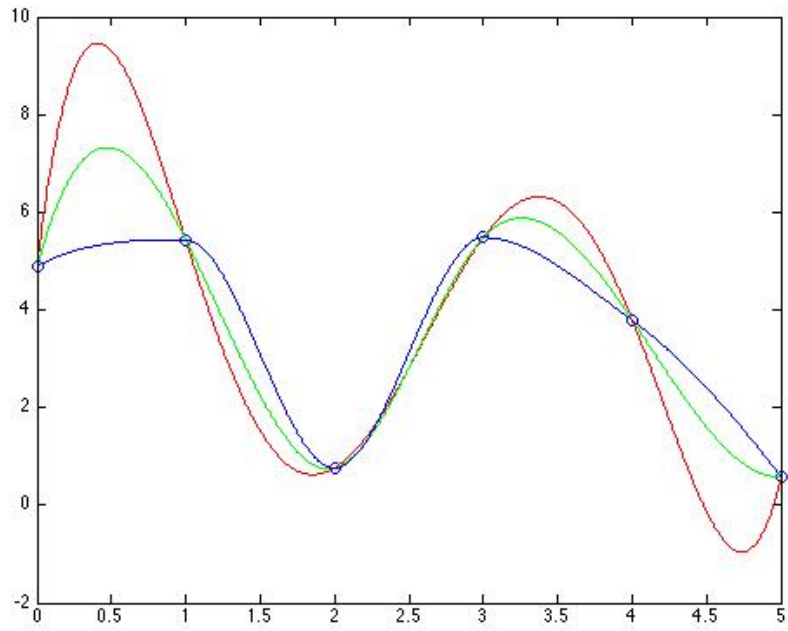
which is easy to solve by hand: $a = \frac{-1}{3}$, $b = \frac{4}{3}$.

(b) The graph at the end of this exam shows the result of applying three interpolation methods (piecewise cubic Hermite interpolation, cubic spline interpolation, and polynomial interpolation) to six randomly chosen points in the plane. Identify which curve is associated with which interpolation method. Explain.

Solution. The blue line is piecewise cubic Hermite interpolation, since it is the only one that is shape-preserving (direction changes in the data correspond to local minima and maxima). Polynomial interpolation typically overshoots the tabulated points much more than spline interpolation, so the red line is polynomial interpolation, and the green line is the cubic spline.

(c) Write a sequence of MATLAB commands that plots the cubic splines interpolating $y = e^x$ at five equally spaced points in the interval between 0 and 1.

Solution. The last line can be replaced by `plot(x,y)` to give a completely correct answer to the question. The solution below displays the five interpolation points as well.



```
>> u=linspace(0,1,5);  
>> v=exp(u);  
>> x=linspace(0,1,500);  
>> y=interp1(u,v,'splines');  
>> plot(u,v,'o',x,y);
```