Homework 1 - Solutions
MA/CS 375, Fall 2005

1. How many numbers belong to the set $\mathbb{F}(2, 2, -1, 4)$? What is the value of ϵ_M for such set?

   Solution: The parameters are $\beta = 2$, $t = 2$, $L = -1$, and $U = 4$. The maximum and minimum representable values in this set are:

   $$x_{\min} = \beta^{L-1} = \frac{1}{4}, \quad x_{\max} = \beta^U(1 - \beta^{-t}) = 12$$

   Machine epsilon for this set is

   $$\epsilon_M = \beta^{1-t} = \frac{1}{2}$$

   There are t digits in the mantissa $[\ a_1 \quad a_2 \quad a_3 \quad \cdots \quad a_t\ ]$. Each of these t bits take β possible values expect the first bit which takes $\beta - 1$ because $a_1 = 0$ is reserved. There is a factor of 2 for the sign bit to account for whether the number is positive or negative. So the range of possible values for the mantissa are $2(\beta - 1)\beta^{t-1}$. The range of possible exponent values is $U - L + 1$, so the total number of values in $\mathbb{F}$ are

   $$N(\beta, t, L, U) = 2(\beta - 1)\beta^{t-1}(U - L + 1)$$

   In the case of this problem the $N = 24$.

2. Use MATLAB to construct an upper (respectively lower) triangular matrix of dimension 8 with -4 on the diagonal and 1 on the upper (respectively lower) diagonal. Interchange the second and fifth rows and then the second and sixth columns. Compute the determinants of each of the matrices you have computed. Do you notice anything?

   Solution: Diagonal matrices are typically created using the `diag()` function, however, this matrix can be created using a single line of MATLAB code with the built-in function `toeplitz()`. A toeplitz matrix is any matrix which contains the same value in every element along a given diagonal. The upper and lower triangular matrices are obtained using:

   ```
   >>U=toeplitz([-4 zeros(1,7)],[-4 1 zeros(1,6)]); U=L';
   ```

   Since these matrices are bidiagonal, the determinants can be computed simply by taking the product of the diagonal entries. This is because when one applies the expansion of minors approach, at each step there is only one submatrix which has a nonzero determinant. The determinant of both `L` and `U` is $(-4)^8 = 65536$. MATLAB gives the same result. To interchange rows 2 and 5, we can use an approach such as the following:

```
>>tempu=U(2,:); U(2,:)=U(5,:); U(5,:)=tempu;
>>templ=L(2,:); L(2,:)=L(5,:); L(5,:)=templ;
```

Although more work is done, this can be also done in 1 command.

```
>>U=U([1 5 3 4 2 6 7 8],:);
```

To interchange the columns 2 and 6,

```
>>tempu=U(:,2); U(:,2)=U(:,6); U(:,6)=tempu;
>>templ=L(:,2); L(:,2)=L(:,6); L(:,6)=templ;
```

This gives the new matrices (zeros not shown):

$$U = \begin{bmatrix} -4 & & & & & & 1 & \\ & 1 & & & -4 & & & \\ & & -4 & 1 & & & & \\ & & & -4 & 1 & & & \\ & & 1 & & & -4 & & \\ & -4 & & & & & 1 & \\ & & & & & & -4 & 1 \\ & & & & & & & -4 \end{bmatrix}, \quad L = \begin{bmatrix} -4 & & & & & & & \\ & & & & 1 & -4 & & \\ & & & -4 & & & 1 & \\ & & & 1 & -4 & & & \\ 1 & & & & & & -4 & \\ & -4 & & & & 1 & & \\ & 1 & & & & & -4 & \\ & & & & & & 1 & -4 \end{bmatrix}$$

These matrices are no longer bidiagonal, so the simple determinant formula used before is not available, but this poses no difficulty to MATLAB:

```
>>det(U)

ans=65536

>>det(L)

ans=65536
```

This demonstrates two important properties. The transpose of a square matrix has the same determinant as the original matrix and interchanging two rows or two columns of a matrix does not alter its determinant.

3. Check the linear independence of the following vectors in $\mathbb{R}^5$:

$$v_1 = \begin{pmatrix} -1 \\ 1 \\ 1 \\ -1 \\ 0 \end{pmatrix}, \; v_2 = \begin{pmatrix} -2 \\ 2 \\ 3 \\ 3 \\ 0 \end{pmatrix}, \; v_3 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \; v_4 = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 0 \end{pmatrix}, \; v_5 = \begin{pmatrix} -1 \\ 1 \\ 1 \\ -1 \\ -1 \end{pmatrix}.$$

Solution: Using MATLAB, define a matrix `V=[v1,v2,v3,v4,v5]`. Then compute the determinant numerically using `det(V)`. MATLAB returns the value 4. Since the determinant is nonzero, the vectors are linearly independent. That is to say that any given vector of the set can not be constructed using a linear combination of the remaining vectors. *Note:* You can compute the number of linearly independent vectors in a matrix V using the MATLAB command `rank(V)`.

4. The Stirling numbers of the second kind are defined by the formula:

$$S_n^{(m)} = \frac{1}{m!} \sum_{k=0}^{m} (-1)^{(m-k)} \begin{pmatrix} m \\ k \end{pmatrix} k^n$$

(a) Write a matlab function "stirling2.m" that accepts the arguments n and m and returns $S_n^{(m)}$

```
function S = stirling_2(n,m)
if n > m
    term = -(-1)^m*m;
    S    = term;
    factorial = 1;
    for k = 2:m-1
        term = -term*(k/(k-1))^n*(m-k+1)/k;
        S = S+term;
        factorial = factorial*k;
    end
    factorial = factorial*m;
    S = S+m^n;
    S = S/factorial;
else
    if n == m
        S = 1
```

```
    else
        S = 0;
    end
end
```

(b) Use your function to compute $S_n^{(m)}$ for $m = 5, 10, 15$ and $n = 5, 10, 15, 20, 25$. The answers must be as accurate as possible. You must write a driver script that calls the function and prints your answers (with as many digits as feasible–use *format long g*) in a table of the form:

Table 1: Stirling numbers

n \ m	5	10	15
5	1	0	0
10	42525	1	0
15	210766920	12662650.0000002	1
20	749206090500	5917584964655.04	452329200.001216
25	2.43668497411075e+15	1.20316339217539e+18	4.2993946553481e+15

5. Consider the following algorithm to compute the volume of the unit sphere. Compute n random triples (x_k, y_k, z_k) with each coordinate uniformly distributed in the interval $[0, 1]$. Compute the number of these, p, lying inside the unit sphere. For n large we expect:

$$\frac{p}{n} \approx \frac{\frac{\text{Volume of Sphere}}{8}}{\text{Volume of Unit Cube}} = \frac{\text{Volume of Sphere}}{8},$$

since the points are constrained to lie in the octant $0 \leq x, y, z \leq 1$ and $1/8$ of the unit sphere lies in that octant. Carry out the computation for increasing n until the results apparently converge to two decimal digits. Using the fact that the volume is $\frac{4}{3}\pi$, use your results to approximate π. How accurate is your approximation?

Solution: Again this computation can be performed with only a single line of MATLAB after setting n.

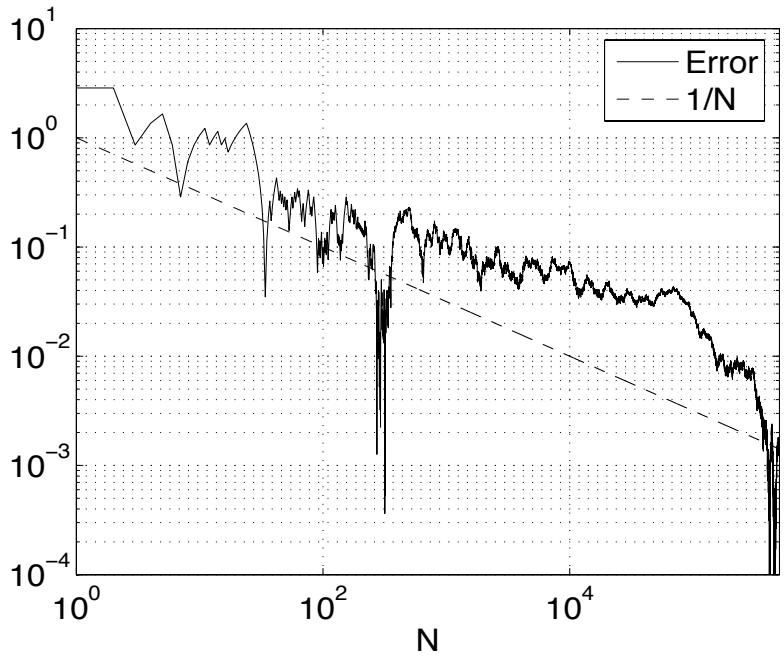
```
>>PI=6*sum(sum(rand(3,n).^2)<1)/n;
```

We can determine an error function for increasing n also in one line.

```
>>err=abs(pi-6*cumsum(sum(rand(3,n).^2)<1)./(1:n));
```

A good way to demonstrate the order of convergence is on a log-log plot.

```
>>loglog(1:n,err);
```



6. The roots of the quadratic

$$ax^2 + bx + c = 0$$

can be found by the well known formula

$$x_{\pm} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

We can also approximate the roots using fixed-point iterations, as follows:

(a) divide through by x to get the equivalent equation (assuming $x \neq 0$):

$$ax + b + \frac{c}{x} = 0$$

(b) Method 1, **forward iteration**: consider the sequence

$$\begin{aligned} x_1 &= x_{init} \\ x_{n+1} &= -\frac{b}{a} - \frac{c}{a}\frac{1}{x_n} \ , \ n = 1, 2, 3, \ldots \end{aligned}$$

(c) Method 2, **backward iteration**: consider the sequence

$$\begin{aligned} x_1 &= x_{init} \\ x_n &= -\frac{b}{a} - \frac{c}{a}\frac{1}{x_{n+1}} \\ \text{i.e.} \quad x_{n+1} &= -\frac{c}{b + a x_n} \ , \ n = 1, 2, 3, \ldots \end{aligned}$$

Write matlab functions **forward.m** and **backward.m** implementing the above iterations, that accept as inputs x_{init}, a, b, c, tol and produce an answer x^* that differs from one of the roots by no more than tol. (You can accomplish that, e.g., by terminating the iteration if $|x_{n+1} - x_n| \leq \text{tol}/2$). Demonstrate the performance of your function for the values:

$$x_{init} = 1 \ , \ a = 1 \ , \ b = c = -1 \ , \ \text{tol} = 10^{-10}$$

How many iterations are required to reach the desired accuracy? Note that the two functions will reach different roots, $x^*_{forward}$ and $x^*_{backward}$. Verify directly that these are within the desired tolerance from one of the roots of the quadratic. Which root is found by each method? Does the answer change if you vary x_{init}? Now, use $x^*_{forward}$ as an initial value for **backward.m** with $\text{tol} = 10^{-10}$. What value do you end up with? Does the answer change if you also run **backward.m** again with $x_{init} = x^*_{forward}$ as before but with $\text{tol} = 10^{-13}$. Why?

Solutions:

```
function x = forward(xinit, a, b, c, tol)
x = xinit;
xold = 0;
i=0;
while abs(xold-x) > tol/2
    i=i+1;
    xold = x;
    x = - (1/a)*(b + c/xold)
end
```

```
i
exact1 = (-b+sqrt(b^2-4*a*c))/(2*a)
exact2 = (-b-sqrt(b^2-4*a*c))/(2*a)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function x = backward(xinit, a, b, c, tol)
x = xinit;
xold = 0;
i=0;
while abs(xold-x) > tol/2
    i=i+1;
    xold = x;
    x = - c/(b + a*xold)
end
i
exact1 = (-b+sqrt(b^2-4*a*c))/(2*a)
exact2 = (-b-sqrt(b^2-4*a*c))/(2*a)
```

Using the values

$$x_{init} = 1 \ , \ a = 1 \ , \ b = c = -1 \ , \ \text{tol} = 10^{-10}$$

forward.m converges within tolerance in **26** iterations to the positive root (1.6180339887383), whereas **backward.m** converges within tolerance in **28** iterations to the negative root (-0.618033988738303).

When **backward.m** is run with $x^*_{forward}$ as an initial guess and a tolerance of 10^{-10}, the program runs for one iteration and returns a new solution, 1.61803398878025 which is within tolerance of an exact solution. However, when the process is repeated and the tolerance requested is 10^{-13}, it requires **61** iterations to converge. This is because $x^*_{forward}$ is not a stable solution for the backward scheme and any error will grow with repeated iterations. Eventually the solution moves far enough away from the starting point to be attracted to the other root.

Explaination of grading scheme:

Problem 1 (16 pts):
Correct ϵ_M (6 pts)
Correct $N(\beta, t, L, U)$ (6 pts)
Explain how you determined N. (4 pts)
No credit for simply copying out of the book.

Problem 2 (17 pts):
Correctly constructing U and L (5 pts)
Correct determinants (4 pts)
Correct interchange (4 pts)
Correct observation (4 pts)
Use of `toeplitz()` (+2 extra credit pts)

Problem 3 (16 pts):
Correct result on linear independence and reasoning (16 pts)

Problem 4 (17 pts):
Working `stirling.m` with correct table (12 pts)
Driver script (5 pts)
Using `nchoosek()` (-5 pts)

Problem 5 (17 pts):
Code (10 pts)
Some results (7 pts)
"One line solution" (+2 extra credit pts)

Problem 6 (17 pts):
Working `forward.m` (5 pts)
Working `backward.m` (5 pts)
Results and discussion (7 pts)