Computer Assignment 2

1. Consider the function $f(x) = \frac{\sin x}{x} - \cos x$. What do you observe when you type `f(1e-8)` in `Matlab`? Explain your result. Re-write $f(x)$ for x near 0 so `f(1e-8)` will give you a more satisfactory result.

   Solution:

   `f(1e-8) = 0` in MATLAB. This is due to a cancellation of error. To remediate this we can write a Taylor expansion of $f(x)$. We get

   $$f(x) \approx \left(\frac{1}{2!} - \frac{1}{3!}\right) x^2 + \left(\frac{1}{4!} - \frac{1}{5!}\right) x^4 + \left(\frac{1}{6!} - \frac{1}{7!}\right) x^6.$$

   Evaluating the above approximation at `(1e-8)` gives `= -5.5000e-16`.

2. The continued fraction representation of the number 2 is given by

   $$2 = 1 + \cfrac{2}{1 + \cfrac{2}{1 + \cfrac{2}{1 + \cfrac{2}{\ddots}}}}. \tag{1}$$

   What is the minimum number of fractions that you need in `MATLAB` to write the number 2 as a continued fraction?

   Solution:

   ```
   r = 1+2/1;
   n=1;
   while r~=2
       r = 1+2/r;
       n = n+1;
   end
   ```

   The iteration stops at `n = 53` so 53 fractions are needed.

3. Implement a code that read the excel file 'LDIstudents.xls' (in the 'assesments / computer assignments' folder in blackboard) and store the information of each student in a `MATLAB structure` that you will call `Student`. For each group, calculate the average mark as well as the standard deviation. Compute the correlation between the marks of all students and the 'Assistance to lectures', 'Assistance to tutorials', and the 'Height'.

   Solution:

   ```
   [numeric, text, ~] = xlsread('LDIstudents.xls');

   Student.Id = numeric(:, 1);
   Student.Group = cell2mat(text(2:size(text,1), 2));
   Student.Height = numeric(:, 3);
   ```

```
Student.Assistance2Lectures = numeric(:, 4);
Student.Assistance2Tutorials = numeric(:, 5);
Student.Mark = numeric(:, 6);

% mark and standard deviation
Groups  = unique(Student.Group);
for k = 1:length(Groups)
    averageMark.Groups(k) = mean(Student.Mark(Student.
       Group == Groups(k)));
    deviationMark.Groups(k) = std(Student.Mark(Student.
       Group == Groups(k)));
end
% Correlations
corr_Marks_Lectures = corr(Student.Mark, Student.
   Assistance2Lectures);
corr_Marks_Tutorials = corr(Student.Mark, Student.
   Assistance2Tutorials);
corr_Marks_Height = corr(Student.Mark, Student.Height);
```

We obtain:

```
averageMark.Groups(1) = 84.4423
averageMark.Groups(2) = 86.5385
deviationMark.Groups(1) = 10.0165
deviationMark.Groups(2) = 9.7487
corr_Marks_Lectures =  0.7244
corr_Marks_Tutorials =  0.6148
corr_Marks_Height = 0.0349
```

4. Write a function that takes as an input a `double` array of positive numbers and outputs its geometric mean. What is the complexity of your algorithm? Using `rand`, `randn`, and `randi`, to corroborate that for a given array a of positive numbers,

$$\text{mean}(a) \geq \text{geometric\_mean}(a).$$

Find a way to plot your results.

Solution:

```
function gmean = geomean(a)
gmean = 1;
 for p = a
     gmean = gmean*p;
 end
 gmean = gmean^(1/length(a));
end
```

For an array of length `n` there are `n` iterations with $O(1)$ operations at each iteration. Hence the complexity of the above function is $O(n)$.

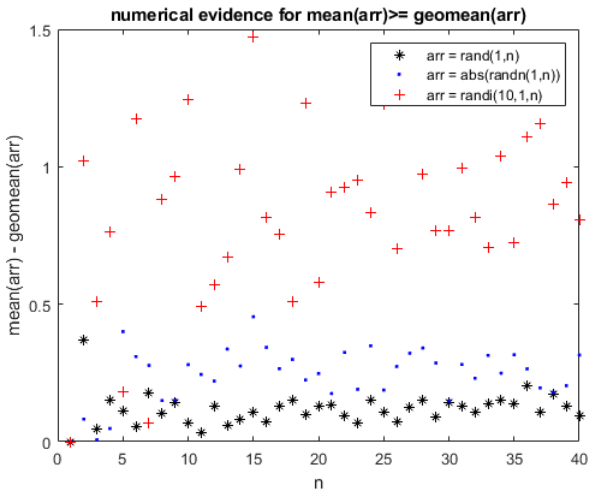

Figure 1: Q4

The following is just one of many ways one could plot the inequality.

```
N = 40;
for n=1:N
    arr = rand(1,n);
    plot(n,mean(arr)-geomean(arr), '*','Color','k'),
       hold on,
    arr = abs(randn(1,n));
    plot(n,mean(arr)-geomean(arr), '.','Color','blue'),
        hold on,
    arr = randi(10,1,n);
    plot(n,mean(arr)-geomean(arr), '+','Color','red'),
       hold on,
end
xlabel('n'); ylabel('mean(arr) - geomean(arr)')
legend('arr = rand(1,n)', 'arr = abs(randn(1,n))', 'arr
    = randi(10,1,n)')
title('numerical evidence for mean(arr)>= geomean(arr)'
   )
```

5. Consider the following function which depends on both spatial variables (x and y) and a time variable t.

$$G_\omega(x,y,t) = \frac{e^{i\omega\sqrt{x^2+y^2}}}{4\pi\sqrt{x^2+y^2}} e^{-i\omega t}.$$

Here i is the imaginary unit. This function represents a 2D version of point source of an acoustic wave with frequency ω. Using `getfram(gcf)`, create a movie of `real(`$G_\omega(x,y,t)$`)` (the real part of G_ω), for x and y between -2π and 2π, and t starting from 0 to any time of your choice. You are free to choose any value for the frequency ω as well.

Consider using `zlim`, `caxis` and an appropriate `shading` to make your movie looks better.

Solution:

```
omega = 5;
[X,Y] = meshgrid(linspace(-2*pi,2*pi,100),linspace(-2*
   pi,2*pi,100));
L = sqrt(X.^2 + Y.^2);
G = @(t) exp(1i*omega*L)./(4*pi*L).*exp(-1i*omega*t);
k = 1;
for t = linspace(0,10,100)
    Z = real(G(t));
    surf(X,Y,Z); shading interp; clim([-.1 .1]);
    xlabel('x'); ylabel('y'); zlabel('z');
    title('Movie of a point source')
    axis equal;
    zlim([-3,3]);
    colorbar;
    F(k) = getframe(gcf);
    k  = k+1;
end
f = figure;
movie(f, F,10)
```

6. Write a recursive function that reverse the order of a 1D input array. Example: `recursiveReverseArray([1 2 3 4 5])` should output `[5 4 3 2 1]`.

   Solution:

```
function out = recursiveReverseArray(arr)
if length(arr) == 1
    out = arr;
else
out = [arr(end) recursiveReverseArray(arr(1:(end-1)))];
end
```

7. The Legendre polynomials satisfy the following recurrence relation.

$$(2n+1)xP_n(x) = (n+1)P_{n+1}(x) + nP_{n-1}(x),$$

   with $P_0(x) = 1$ and $P_1(x) = x$. Show that $P_n(x) = O(x^n)$ for $x \to \infty$.

   Hint: use inductive/recursive thinking.

   Solution:

   Note that if $f(x) = O(x^n)$, then $xf(x) = O(x^{n+1})$. Then, it also holds that if $f(x) = O(x^n)$ and $g(x) = O(x^{n-1})$, then $xf(x) + g(x) = O(x^{n+1})$.

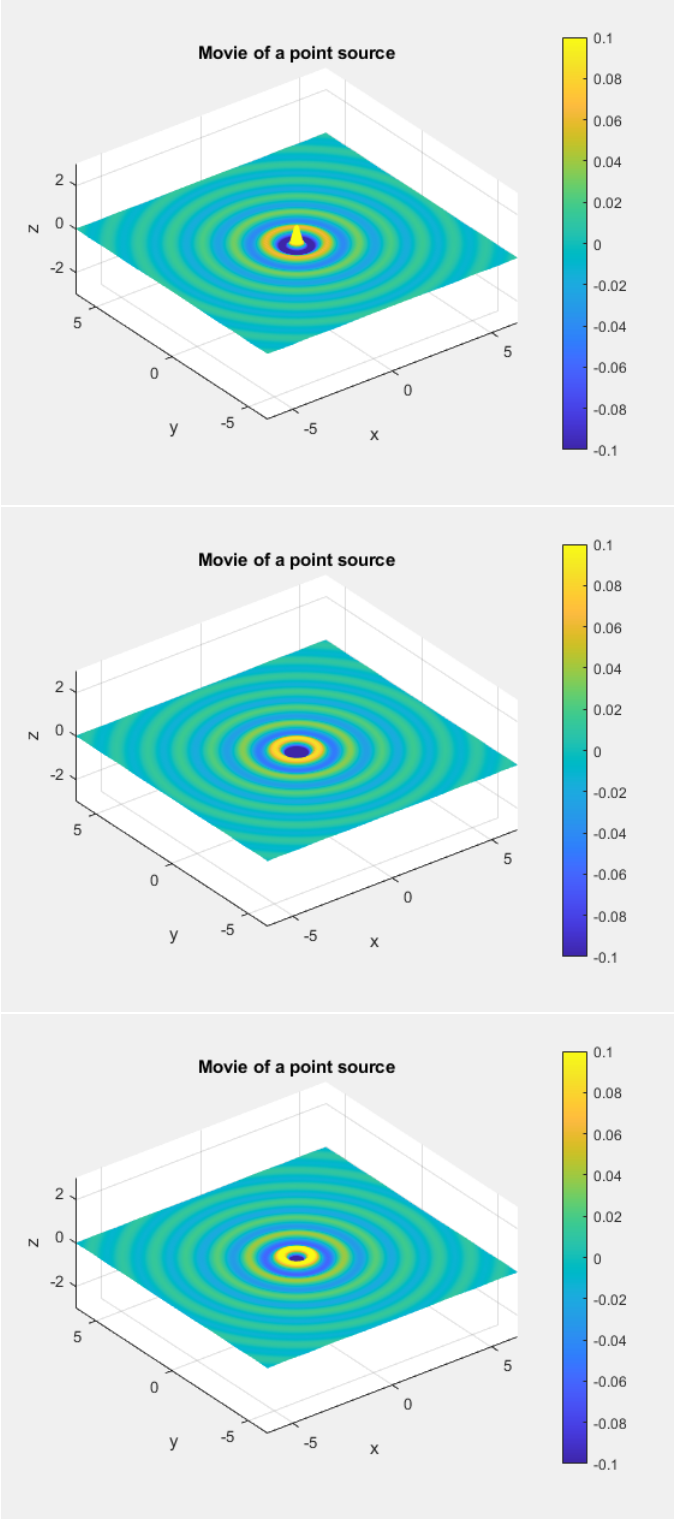

Figure 2: screenshots for Q4

From the above we have that if $P_n = O(x^n)$ and $P_{n-1} = O(x^{n-1})$, then $P_{n+1} = O(x^{n+1})$. The fact that $P_0 = O(1)$ and $P_1 = O(x)$ shows, by induction, that $P_n = O(x^n)$ for any $n \geq 0$.

8. Write a function that takes as an input a 1D `double` array and returns the index of the smallest element in the array. What is the complexity of your algorithm?

   Solution:

```
function idx = myMin(arr)
min = arr(1);
idx = 1;
for i = 1:length(arr)
    if arr(i)<=min
        idx = i;
        min = arr(i);
    end
end
```

   For an array of length `n` there are `n` iterations with $O(1)$ operations at each iteration. Hence the complexity of the above function is $O(n)$.

9. The following is the implementation of the 'Selection Sort' algorithm seen in lectures

```
function out = selectionSort(arr)
n = length(arr);
for i = 1:n
    % Find the minimum element in the unsorted array
    [~,idx]= min(arr(i:end));
    min_idx = idx + i-1;
    % Swap the minimum element with the first
    if min_idx ~= i
        aux = arr(i);
        arr(i) = arr(min_idx);
        arr(min_idx) = aux;
    end
end
out = arr;
end
```

   Replace the `min` built-in `MATLAB` function on the code above with your own implementation. What is the complexity of the Selection Sort algorithm using your own `min` function? Plot the average complexity for random inputs of varying length.

   Solution:

   For an array of length `n` there are `n` iterations with $O(n)$ operations at each iteration. Hence the complexity of the above function is $O(n^2)$.

   The code for plotting the average runtime is:

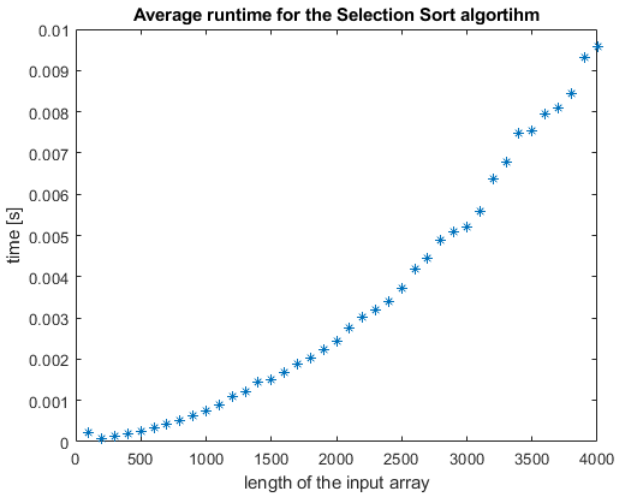

Figure 3: Q9

```
N = 40; K = 100;
T = zeros(N,K);
for k=1:K
    for n = 1:N
        arr = rand(1,n*100);
        tic,
        mySelectionSort(arr);
        T(n,k) = toc;
    end
end
T = 1/K*sum(T,2);
%%
plot([1:N]*100, T, '*'), hold on,
xlabel('length of the input array'); ylabel('time [s]')
title('average runtime for the Selection Sort algortihm
   ')
```

10. Re-write the following function to make it more efficient. Showcase your result in a plot.

```
function R = myFunction(M)
n = size(M,1);
for i= 1:n
    for j = 1:n
        R(i,j) = M(i,j) + M(j,i);
    end
end
end
```

Solution:

The function simply outputs `M+M'`:

```
function R = myEfficientFunction(M)
R = M + M';
end
```

For plotting:

```
N = 100; K = 100;
T1 = zeros(N,K);
T2 = zeros(N,K);
for k=1:K
    for n = 1:N
        M = rand(n);
        tic,
        myFunction(M);
        T1(n,k) = toc;
        tic,
        myEfficientFunction(M);
        T2(n,k) = toc;
    end
end
T1 = 1/K*sum(T1,2);
T2 = 1/K*sum(T2,2);

plot(1:N, T1, '*'), hold on, plot(1:N, T2, '.')
legend('inefficient  function', 'efficient function')
xlabel('length of the input matrix'); ylabel('time [s]
   in logarithmic scale');
set(gca, 'YScale', 'log')
title('average runtime')
end
```

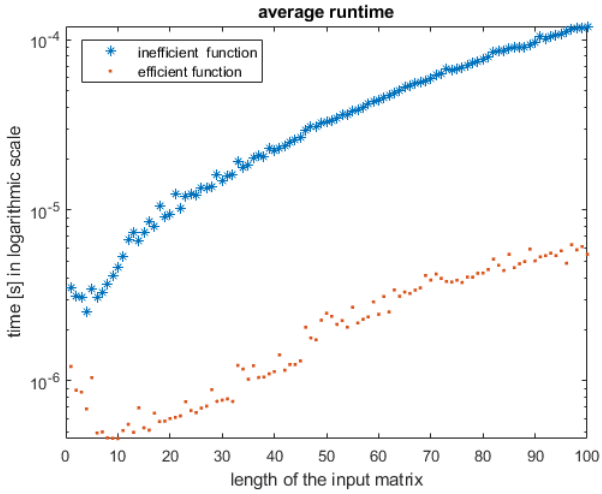
average runtime
inefficient function
efficient function
time [s] in logarithmic scale
length of the input matrix
10^-4
10^-5
10^-6
0
10
20
30
40
50
60
70
80
90
100

Figure 4: Q10