Numerical Mathematics:
Homework 1

Problem 1

Prove that for any $\epsilon > 0$, $n^{1+\epsilon} \notin \mathcal{O}(n \log n)$ (but $n \log n \in \mathcal{O}(n^{1+\epsilon})$). Hence, $n^{1+\epsilon}$ grows asymptotically faster than $n \log n$.

Solution. It suffice to show that $n^{\epsilon} \notin \mathcal{O}(\log n)$. This implies that for every $C > 0$, there exists a $N > 0$ such that:

$$n^{\epsilon} > C \log n.$$

By applying the substitution $n = \exp(\hat{n})$, we may rewrite this this statement to: for every $C > 0$, there exists a $\hat{N} > 0$ such that:

$$\exp(\hat{n}\epsilon) > C\hat{n}.$$

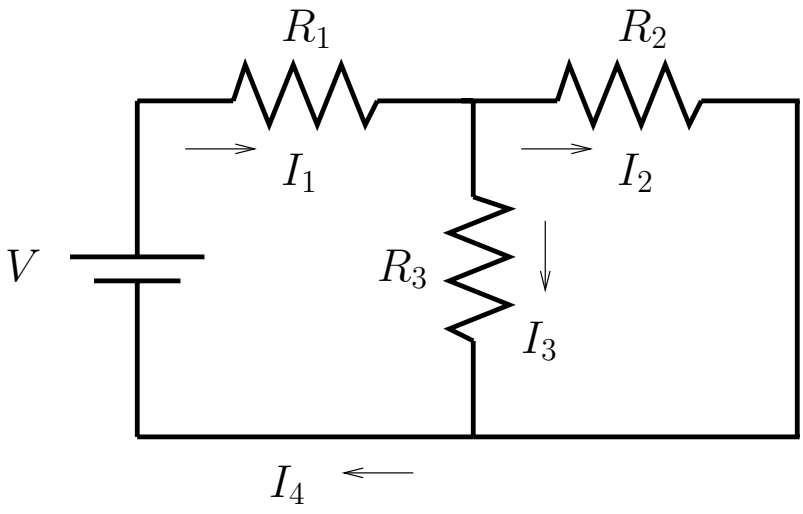
The latter statement is a well known fact. □

Problem 2

For the DC circuit shown here, Kirchhoff's Rules and Ohm's Law tell us the following relationships between the currents I_1, I_2, I_3, I_4, the resistances R_1, R_2, R_3, and the voltage V provided by the battery:

$$\begin{aligned} I_2 + I_3 &= I_1 \\ I_2 + I_3 &= I_4 \\ I_1 R_1 + I_3 R_3 &= V \\ I_1 R_1 + I_2 R_2 &= V \end{aligned}$$

Submit a function `find_current.m` which takes as inputs the values for V, R_1, R_2, and R_3 (in that order), and returns as output a column vector containing the values of the currents I_1, I_2, I_3, I_4 (again, in that same order). You are allowed to use the backslash operator here if you want!



Solution. Shown below is the code.

```
function I = find_current(V,R1,R2,R3)

% matrix for Kirchhoff's rules
A = [-1 1 1 0;
      0 1 1 -1;
      R1 0 R3 0;
      R1 R2 0 0];    % 2 pts

% righthand side for Kirchhoff's rules
b = [0;0;V;V]; % 2 pts

% solve for currents
I = A\b; % 1pts

end
%find_current(1,2,3,4)

ans =

0.2692
0.1538
0.1154
0.2692
%
```

□

Problem 3

In this problem, you are going to get practice in plotting functions. Consider the signal:

$$x = \sin(50\pi t) + \frac{1}{4}\sin(226\pi t) + (1/100)t^2$$

(a)
Build a function called `PlotSignal.m` which plots this curve on the interval $0 \leq t \leq 0.1$. Use an increment of $\Delta t = 0.001$. Create this signal by first allocating a vector of zeros for x and then use a for loop to fill in the vector. Plot the function ($x(t)$ in the vertical axis, t in the horizontal axis). Give the figure a title called 'signal', add labels in the x-axis ('t') and y-axis ('x(t)'). Note that `PlotSignal.m` doesn't have any inputs and outputs.

(b)
Do the same as in part IIa, but now vectorize the code. That is, use the element-by-element operation wherever it is necessary so that *you no longer need to use a for loop.* Call this new function `PlotSignal_vectorized.m`.

Solution.

(a)
Shown below is the code.

```
function [ ] = PlotSignal( )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

```

```
t= 0:0.001:0.1;
x = zeros(size(t));

for l=1:length(x)  % 1 pts
    x(l) = sin(50 *pi * t(l))  + 0.25 * sin(226 * pi *t(l)) + 0.01 * t(l)^2; % 1 pts
end


figure
plot(t,x) % 1pts
xlabel('t'); ylabel('x') % 1pts
title('Signal') % 1 pts



end
```

(b)
Shown below is the code.

```
function [ ] = PlotSignal_vectorized( )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

t= 0:0.001:0.1;
x = sin(50 *pi * t)  + 0.25 * sin(226 * pi *t) + 0.01 * t.^2; % 2pts


figure
plot(t,x) % 1 pts
xlabel('t'); ylabel('x') % 1 pts
title('Signal') % 1pts



end
```

□

Problem 4

Make a function calld `evaluate_pi.m` which plots the evaluates π using the formula:

$$\frac{\pi}{2} = 1 + \frac{1}{3} + \frac{1 \cdot 2}{3 \cdot 5} + \frac{1 \cdot 2 \cdot 3}{3 \cdot 5 \cdot 7} + \frac{1 \cdot 2 \cdot 3 \cdot 4}{3 \cdot 5 \cdot 7 \cdot 9}$$

The function must take as argument n, the number of terms. Code it efficiently!

Solution. Shown below is the code.

```
function [ out] = evaluate_pi( n )
%EVALUATE_PI Summary of this function goes here
%   Detailed explanation goes here

out = 1;

if n > 0

    a=1;
```

```
    for k=1:n
        a = a * k/(2*k+1)
        out = out + a;
    end


end


out = 2*out;

end
```

□

(extra) Problem 5

Apply Horner's method to evaluate a polynomial of the form:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

In other words, by rewriting the polynomial into the form:

$$p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + x(\cdots(a_{n-1} + a_n x)\cdots))))$$

we can introduce the variables:

$$\begin{aligned} b_n &:= a_n \\ b_{n-1} &:= a_{n-1} + b_n x \\ &\vdots \\ b_0 &= a_0 + b_1 x \end{aligned}$$

to evaluate the polynomial efficiently. Write a MATLAB function called `poly_eval_horner.m` which evaluates a polynomial of degree n using Horner's iteration scheme. Use the following structure: `function [pofx] = eval_poly_horner(x,a)` where `x` is a scalar input, `a` is a column (or row) vector of size $n+1$ with the coefficients $a_0, a_1, \ldots, a_n$ in it, and `pofx` is the function value $p(x)$.

Solution. Shown below is the code.

```
function [ pofx ] = eval_poly_horner( x, a )
%EVAL_POLY_HORNER Summary of this function goes here
%   Detailed explanation goes here

pofx = a(end);
for k=1:(length(a)-1)
    pofx = a(end-k) + pofx*x;
end


end
```

□