# CS/SE 4X03 — Assignment 4

23 November, 2021

Due date: 6 December

**Instructions**

- If you write your solutions by hand, please ensure your handwriting is legible. We may subtract marks for hard-to-read solutions.
- Submit to Avenue a **PDF file** containing your solutions and the **required** MATLAB **files**.

  Assignments in other formats, e.g. IMG, PNG, **will not be marked**.

  Name your MATLAB files **exactly** as specified.
- Name your PDF file **Lastname-Firstname-studentnumber.pdf**.
- Submit **only what is required**.
- Do not submit zipped files. We will **ignore any compressed file** containing your files.
- We will **deduct marks** for not following the required naming conventions.

**Problem 1** [9 points] Study the slides about deep learning and sections 1 to 4 and 6 of `https://arxiv.org/abs/1801.05894`.

**[2 points]** Modify the `netbp.m` code so you can pass parameters as follows

```
function  cost = netbp2(neurons, data, labels, niter, lr, file)
%Trains a neural network with 4 layers.
%Layer 1 and layer 4 have 2 neurons, layer 2 has neurons(1) and
%layer 3 has neurons(2).
%data is a matrix with two rows.  data(:,i) contains the (x,y)
%coordinates of point i.
%labels is a matrix with two rows.  labels(:,i) is [1;0]
%if data(:,i) is in category A and [0;1] if it is in category B.
%lr is learning rate
%niter is maximum number of iterations
%file is a filename where the file is created by
%save(file,'W2','W3','W4','b2','b3','b4')
%cost is a vector storing the value of
%the cost function after each iteration; cost(j) is the cost at
%iteration j
```

**[2 points]** Implement the Matlab function

```
function category = classifypoints(file, points)
%Reads parameters from a file and classifies points into two
%categories, A or B. This file is created by netbp2.m.
%points is a matrix with two rows, where points(:,i) contains the
%(x,y) coordinates of point i.
%Returns vector category, where category(i) is 1 if
%points(1,i) >= points(2,i) and 0 otherwise.
```

**[4 points]** Run the function
`function main_nn(neurons, learning_rate, niter)`
with various parameters. Try to find parameters to this function such that in the plot `classified.eps` the grey region contains only $A$ training points, and the white region contains only $B$ training points. Note: your regions can be quite different from the regions in Figure 1.

**[1 point]** Summarize in 4 bullets what you have found useful/interesting in this problem.

**Submit**

- PDF: `netbp2.m`, `classifypoints.m`, the two plots produced by `main_nn.m`
- Avenue: `netbp2.m`, `classifypoints.m`

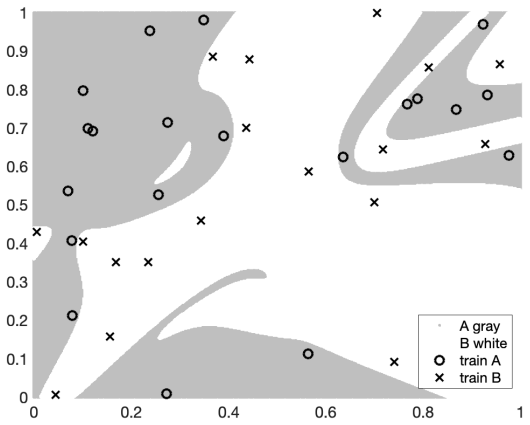


Figure 1: Classified points

**Problem 2** [7 points] Implement in Matlab the bisection method for finding a root of a scalar equation.

Consider the polynomial

$$\begin{aligned} f(x) &= (x-2)^9 \\ &= x^9 - 18x^8 + 144x^7 - 672x^6 + 2016x^5 - 4032x^4 + 5376x^3 - 4608x^2 + 2304x - 512. \end{aligned}$$

(a) [2 points] Write a Matlab script to evaluate this function at 161 equidistant points in the interval $[1.92, 2.08]$ using two methods:

- evaluate $(x-2)^9$ directly
- evaluate $x^9 - 18x^8 + 144x^7 - 672x^6 + 2016x^5 - 4032x^4 + 5376x^3 - 4608x^2 + 2304x - 512$ using Horner's method

Plot the results in two different plots. Explain the differences between the two plots.

(b) [2 points] Apply your bisection method to find a root starting with $[1.92, 2.08]$, tolerance $10^{-6}$, and using the Horner's evaluation.

Obviously, $r = 2$ is a root of multiplicity 9. Can you determine a value for $r$ such that $|r-2| < 10^{-6}$?

(c) [2 points] If you cannot find a root with accuracy of $10^{-6}$ can you explain why?

(d) [1 point] Apply Matlab's `fsolve` to find a root of

$$x^9 - 18x^8 + 144x^7 - 672x^6 + 2016x^5 - 4032x^4 + 5376x^3 - 4608x^2 + 2304x - 512$$

using initial guess 1.9 and of $(x-2)^9$ with initial guess 1.9.

Report your results.

**Submit**

- PDF: answers to (a-d)

**Problem 3** [8 points] Implement Newton's method for systems of equations.

Each of the following systems of nonlinear equations may present some difficulty in computing a solution. Use Matlab's `fsolve` and your own implementation of Newton's method to solve each of the systems from the given starting point.

In some cases, the nonlinear solver may fail to converge or may converge to a point other than a solution. When this happens, try to explain the reason for the observed behaviour.

Report for `fsolve` and your implementation of Newton's method and each of the systems below, the number of iterations needed to achieve accuracy of $10^{-6}$ (if achieved).

(a)

$$\begin{aligned} x_1 + x_2(x_2(5 - x_2) - 2) &= 13 \\ x_1 + x_2(x_2(1 + x_2) - 14) &= 29 \end{aligned}$$

starting from $x_1 = 15$, $x_2 = -2$.

(b)

$$\begin{aligned} x_1^2 + x_2^2 + x_3^2 &= 5 \\ x_1 + x_2 &= 1 \\ x_1 + x_3 &= 3 \end{aligned}$$

starting from $x_1 = (1 + \sqrt{3})/2$, $x_2 = (1 - \sqrt{3})/2$, $x_3 = \sqrt{3}$.

(c)

$$\begin{aligned} x_1 + 10x_2 &= 0 \\ \sqrt{5}(x_3 - x_4) &= 0 \\ (x_2 - x_3)^2 &= 0 \\ \sqrt{10}(x_1 - x_4)^2 &= 0 \end{aligned}$$

starting from $x_1 = 1$, $x_2 = 2$, $x_3 = 1$, $x_4 = 1$.

(d)

$$\begin{aligned} 10^4 x_1 x_2 &= 1 \\ e^{-x_1} + e^{-x_2} &= 1.0001 \end{aligned}$$

starting from $x_1 = 0$, $x_2 = 0$.

**Submit**

- PDF: answers to (a-d)
- Avenue: your MATLAB code. Name your main file `main_newton.m`. When executed, it should produce all required results.

**Problem 4** [4 points] Consider two bodies of masses $\mu = 0.012277471$ and $\hat{\mu} = 1 - \mu$ (Earth and Sun) in a planar motion, and a third body of negligible mass (moon) moving in the same plane. The motion is given by

$$\begin{aligned} u_1'' &= u_1 + 2u_2' - \hat{\mu}\frac{u_1 + \mu}{((u_1+\mu)^2 + u_2^2)^{3/2}} - \mu\frac{(u_1 - \hat{\mu})}{\left((u_1-\hat{\mu})^2 + u_2^2\right)^{3/2}} \\ u_2'' &= u_2 - 2u_1' - \hat{\mu}\frac{u_2}{((u_1+\mu)^2 + u_2^2)^{3/2}} - \mu\frac{u_2}{\left((u_1-\hat{\mu})^2 + u_2^2\right)^{3/2}}. \end{aligned} \tag{1}$$

The initial values are

$$\begin{aligned} u_1(0) &= 0.994, \\ u_1'(0) &= 0, \\ u_2(0) &= 0, \\ u_2'(0) &= -2.00158510637908252240537862224. \end{aligned} \tag{2}$$

Implement the classical Runge-Kutta method of order 4 and integrate this problem on [0,17.1] with uniform stepsize using 100, 1000, 10,000, and 20,000 steps. Plot the orbits, i.e. $u_2$ versus $u_1$ for each case. How many uniform steps are needed before the orbit appears to be qualitatively correct.

**Submit**

- PDF: the four plots.
- Avenue: all your MATLAB code. Name your main file `main_rk4.m`. When executed, it should produce the four plots.

**Problem 5** [6 points] Integrate (1-2) with MATLAB's ODE solvers ode23, ode45, ode78, ode89, and ode113 from 0 to 1000 with absolute and relative error tolerances of $10^{-10}$. Report for each solver in a table

| | | number of | | |
|---|---|---|---|---|
| solver | CPU time | steps | failed steps | function evaluations |
| ode23 | | | | |
| ode45 | | | | |
| ⋮ | | | | |

Which is the most efficient solver on this problem?

**Submit**

- PDF: the table
- Avenue: all your MATLAB code. Name your main file `main_solvers.m`; it should produce this table on the screen.

**Problem 6** Bonus, [10 points] Write a MATLAB script `main_bonus.m` that

(a) [5 points] uses `ode15s` and produces the plots in Figure II.1.2 from `https://archimede.dm.uniba.it/~testset/report/hires.pdf`.

(b) [3 points] for relative and absolute tolerances of $10^{-6}$, produces the table

| | | number of | | | | |
|---|---|---|---|---|---|---|
| solver | CPU time | steps | failed steps | function eval | LU decompositions | nonlinear solves |
| ode23s | | | | | | |
| ode15s | | | | | | |
| ode45 | | | | | | |

for the HIRES problem from this pdf (ode45 is an explicit method so it does not have nonlinear solves).

(c) [2 points] What conclusions can you make from this experiment?

**Submit**

- PDF: the table, conclusions
- Avenue: all your MATLAB code. `main_bonus.m` should produce this table on the screen.

**Problem 7** [10 points] For this problem use the file `nbody.dat` from Assignment 3. Write the body of the Matlab function

```
function T = findPeriod(t, x, y, z)
% FINDPERIOD finds the period of an object in a periodic
% motion.
% t is a vector with time instants.
% The position at time t(i) is x(i), y(i), z(i).
% If the data is not periodic, or if a period cannot be computed,
% then -1 is returned.
% The spacing between t(i) and t(i+1) is not necessarily
% uniform.
%
% The initial position is at time t(1) and it is given by
% x(1), y(1), z(1).
% This function finds (an approximation) for the period T such that
% at times t(1), t(1)+T, t(1)+2*T ...
```

```
% the object is at position x(1), y(1), z(1) again.
% Calculate T here
T = -1;
end
```

(a) [5 points] Describe your algorithm in words and pseudo code.

(b) [5 points] Run the script script `main_nbody.m`. Validate your computations against the data at

`http://www.windows2universe.org/our_solar_system/planets_table.html`.

You will receive full marks if your results are accurate up to the first digit after the decimal point. E.g. the period of Uranus is given as 84.01; full mark is if you obtain 84.0.

**Submit**

- PDF: the output of this script, `findPeriod.m`.
- Avenue: all your MATLAB code.

**Problem 8** [4 points] The Kermack-McKendrick model for the course of an epidemic in a closed population is given by the system of ODEs

$$\frac{dS}{dt} = -\beta SI \tag{3}$$

$$\frac{dI}{dt} = \beta SI - \gamma I \tag{4}$$

$$\frac{dR}{dt} = \gamma I, \tag{5}$$

where $t$ is time, and at time $t$,

- $S(t)$ is the number of people that are susceptible of being infected
- $I(t)$ is the number of infected people
- $R(t)$ is the number of people who are not infections (e.g. vaccinated or developed immunity)

The parameter $\beta > 0$ is infection rate and the parameter $\gamma > 0$ is recovery rate (see below).

This is an SIR (Susceptible, Infected, Recovered) model, and one of the fundamental models in epidemiology.

If we add the equations (3,4,5) we have

$$\frac{d}{dt}(S + I + R) = 0.$$

That is, the total population does not change over time. Suppose we have initial values, at time $t = 0$, $S(0) = S_0 > 0$, $I(0) = I_0 > 0$, and $R(0) = 0$. Then at any $t$,

$$S(t) + I(t) + R(t) = S_0 + I_0.$$

Let

$$R_0 = \frac{\beta}{\gamma}.$$

From (4), $I(t)$ will increase from its initial value when

$$I'(0) = \beta S(0)I(0) - \gamma I(0) = \gamma I_0(R_0 S_0 - 1) > 0.$$

That is, when $R_0 S_0 > 1$, and will decrease when $R_0 S_0 < 1$.

The constant $\gamma$ is the fraction of the population that will recover per day. For example, if for COVID-19 there are $d = 14$ days to recover, $\gamma = 1/d = 1/14$ of the infected population will recover per day.

Using Matlab's `ode45`, simulate (3,4,5) for $t \in [0, 200]$ with a population of size $n = 1000$ and initial conditions $I_0 = 1$, $S_0 = n - I_0$, $R_0 = 0$. Use $\gamma = 1/14$. Experiment with various values for $\beta$. For each of $R_0 S_0 \in \{0.9, 1, 3, 5\}$, plot $S(t), I(t), R(t)$ versus $t$.

**Submit**

- PDF: the four plots
- Avenue: all your MATLAB code. Name the main program `main_sir.m`