Topic 0: Introductory Computer Exercises

C1. *But... that's not quite right!!* In Matlab, set

```
x=1/7, y=2/7, z=4/7
```

Now compute $x + y + z$ by writing `r=(x+z)+y`. Check your answer is correct by computing `r-1`. What do you find?

Normally, Matlab displays a small number of decimal places. You can display more using the command `format long`. Do this for `x`, `y`, `z` and `r`.

You can print an *exact* binary representation of each number by using the `flt2bin` function in the file `flt2bin.m` on EleUM. Print the binary representation of `x`, `y`, `z` and `r`. Explain your result for `(x+z)+y`.

C2. *Becos, becos, becos* Type any number in your calculator. Now repeatedly press the `cos` button. (Make sure you are in *radians* mode. What happens? Which equation are you solving?

C3. *Find roots like a Babylonian* The equation $x^2 = a$ is equivalent to $2x^2 = a + x^2$, which is in turn equivalent (for $x \neq 0$) to

$$x = \frac{a}{2x} + \frac{x}{2}.$$

Set `a=2` and `x` to any positive number. Now repeatedly execute

```
x = a/(2*x) + x/2 .
```

What happens?

The equation $x^2 = a$ is also equivalent to $x = a/x$. What happens if you change `x` and repeatedly execute

```
x = a / x?
```

C4. *Polymath* When measuring how a a quantity y depends on x, we often measure data points (x_i, y_i) for $i = 1, \ldots, n$. For example, the data

x	0.0	0.1	0.3	0.6	1.0
y	0.96	0.99	0.67	0.28	0.11

can be represented in Matlab by

```
X=[0.0,0.1,0.3,0.6,1.0]
Y=[0.96,0.99,0.67,0.28,0.11]
```

Plot the data using the command `plot(X,Y)`. How well do you think this plot reflects the real relationship between x and y?

A better fit can be obtained using *polynomial interpolation*. A polynomial function p is computed such that $p(x_1) = y_1, \ldots, p(x_n) = w_i$. In order to fit n data points, a polynomial of degree $n - 1$ is needed. i.e. The highest term is in x^{n-1}. The polynomial interpolation to the 5 data points (x_i, y_i) can be computed using

```
p = polyfit(X,Y,5-1) .
```

Note: Matlab stores a polynomial of degree d as an array of coefficents $c = [c_1, c_2, \ldots, c_{d+1}]$ representing

$$p(x) = c_1 x^d + c_2 x^{d-1} + \cdots + c_{d-1} x^2 + c_d x + c_{d+1}.$$

Plot the polynomial p over the interval from $a = -0.1$ to $b = 1.1$ with horizontal resolution $h = 0.01$, using

```
XX=-0.1:0.01:1.1;
plot(XX,polyval(p,XX));
```

Also plot p and the original data together using `plot(X,Y, XX,polyval(p,XX))`.

The polynomial describing the polynomial p can be evaluated using Matlab command `polyval`. Evaluate the polynomial at the points 0.2 and 0.3 using

`polyval(p,[0.2,0.3])`.

Compare your computed values with with the given values at the interpolation nodes.

C5. *Back to basis* Fix `n=6`, and set

`X=-n:1:n, Y=zeros(1,2*n+1); Y(n+1)=1.`

Compute and plot the polynomial interpolating the $2n+1$ points (x_i, y_i) over the interval $[-n, n]$. Comment on what you see.

C5. *A patchwork of polynomials* A better way of interpolating a function is to use a piecewise-polynomial or *spline*.

Use the Matlab command `s=spline(X,Y)` to compute the spline s interpolating the data of Question 4. Examine the data structure `s`, especially the fields `s.breaks` and `s.coefs`.

Plot the spline s over the interval $[-0.2, +1.2]$ using `plot(XX,ppval(s,XX))`.

Use the Matlab command `ppval(s,x)` to evaluate the piecewise-polynomial spline s over the interval $[0, 1]$ with spacing 0.1.

Repeat your calculusations for the interpolation date of Question 5.

C6. *Lining-up* Fitting data exactly is often a bad idea as it leads to *overfitting*; capturing the *noise* in the data rather than the *trend*. A better way is to find a function *approximating* the data.

The Matlab command `polyfit` can also be used to approximate data. For example, the command

`polyfit(X,Y,1)`

gives a linear (degree-1) approximation to the data. Use this command to compute a linear approximation to the data of Question 4. Plot the resulting function.

C7. *Making waves* In signal processing (and also in your ear) a periodic signal is decomposed as a sum of sine and cosine functions. This decomposition is called a *Fourier series*, and the terms of the series are the *harmonics* of the signal.

For example, the square wave

$$f(x) = \begin{cases} +1 \text{ if } \lfloor x/\pi \rfloor \text{ is even,} \\ -1 \text{ if } \lfloor x/\pi \rfloor \text{ is odd,} \end{cases}$$

has Fourier series

$$f(x) = \frac{4}{\pi} \sum_{k=0}^{\infty} \frac{1}{2k+1} \sin\big((2k+1)x\big).$$

Use Matlab to plot the square wave

`f = @(x) 1 - 2*mod(floor(x/pi),2).`

and on the same graph, the first few terms of the Fourier series

`fs = @(x) (4/pi) * ( sin(x) + sin(3*x)/3 + sin(5*x)/5 ).`

over the interval from -10 to $+10$. What happens if you take more (or fewer) terms?

C8. *Spot the difference* Recall from Calculus that the derivative of a function is defined as

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} .$$

We can therefore estimate the derivative by

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

for h very small.

Estimate the derivative of the function $f(x) = 1/x$ at $x = 1$ using different values of h. You can set up your calculuation using

```
f=@(x)1/x, x=1
h=0.1
fp=(f(x+h)-f(x))/h
```

Compare your answer with the exact value $f'(1) = -1$. Which value of h gives the best approximation? You can try both positive and negative values of h.

Another approximation to the derivative is given by

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} .$$

Repeat your calculations with this approximation.

C9. *Adding it all up* Recall from Calculus that the integral of a function is defined as a limit of Riemann sums.

$$\int_a^b f(x)\,dx = \lim_{||P|| \to 0} \sum_{i=1}^{n} f(c_i)(p_i - p_{i-1})$$

where P is a partition $a = p_0 < p_1 < p_2 < \cdots < p_{n-1} < p_n = b$, and $p_{i-1} \le c_i \le p_i$ for all i.

We can compute an approximation to an integral by taking a Riemann sum with n equally-spaced nodes, $p_i = a + ih$ where $h = (b-a)/n$. Note that then $p_i - p_{i-1} = h$ for all i. We can also take c_i to be the midpoint of p_{i-1} and p_i, so $c_i = a + (i - \frac{1}{2})h$.

In Matlab, we can define the points c_i by

```
h=(b-a)/n, C=a+h*([1:n]-0.5)
```

and the approximation to the integral by

```
I = h*sum(f(C))
```

Estimate the integral of $f(x) = 1/x$ and $f(x) = e^x = \exp(x)$ over the interval $[1, 3]$ using $n = 8$ subintervals. Compare your estimates with the exact answer.

Note: To allow "vectorised" operations in Matlab, define $f(x) = 1/x$ by `f = @(x) 1./x`.

C10. *Growth and decay* The Matlab command

```
s=ode45(f,[a,b],y0)
```

solves the differential equation $dy/dx = f(x, y)$ on the interval $x \in [a, b]$ with initial condition $y(a) = y_0$. The result consists of approximate values $y_i \approx y(x_i)$ stored in arrays `s.x` and `s.y`. A faster but less accurate solver is given by `ode23`.

Define the function $f(x, y) = e^{-x} - y/2$ using the command

```
f = @(x,y) exp(-x) - y/2 .
```

Use the Matlab commands `ode23` and `ode45` to compute $y(x)$ for $x \in [0, 6]$ for the ordinary differential equation:

$$dy/dx = e^{-x} - y/2; \quad y(0) = 0 .$$

Compare with the actual solution $y(x) = 2(e^{-x/2} - e^{-x})$.

C11. *Time is ticking* Matlab can also be used to solve differential equations in more than one variable. An example is the forced pendulum

$$\frac{dx}{dt} = v; \quad \frac{dv}{dt} = \frac{1}{m}\big(A\cos\omega t - kx - \delta v\big)$$

Setting $y_1 = x$ and $y_2 = v$, we can express this in Matlab as

```
f = @(t,y) [y(2); (A*cos(omega*t) - k*y(1) - delta*y(2)) / m] .
```

Compute the solution s for $m = \delta = k = A = \omega = 1$ starting at $x = 1$ and $v = 0$ over the interval $T = [0, 4\pi]$ by setting `T=[0,4*pi], y0=[1;0]` and using `ode45(f,T,y0)` .