Numerical Methods (NUM101) — Coursework 2

This coursework consists of one part. it is worth 17% of the credits for this unit. The maximum number of marks for this coursework is 17.

Deadline	hand-in or upload?
31 Mar (Thu)	10:30am upload to Victory Assignment `CW2`
31 Mar (Thu)	hand-in of printouts at CAM office

Instructions and rules

- Material to be handed in to CAM: printouts of all program codes.
- Material to be uploaded to Victory: Matlab script `cw2.m`, function file `WeightedRF.m`, and, if you need them, function files of other functions that you call inside `cw2.m` or `WeightedRF.m`.
- Marks will be awarded for your work only if both, the electronic upload and the printed hand-in from the CAM office, are present and identical. The uploaded code will be tested by me and receive a score that is provisional until I mark the printout.
- Scripts and functions that generate Matlab errors receive provisionally 0 marks. The student then has to demonstrate in the lab session that the error is minor to justify partial credit.
- Credit:

  100% code performs computation correctly and efficiently, is well structured and commented;

  ≥80% code performs computation correctly and efficiently

  ≥60% code performs computation correctly but has problems[1];

  ≥40% code does not perform computations correctly but could be made to work with minor corrections;

  ≥20% the intentions behind the code are discernible with some effort.

- This is **individual** coursework. In cases where **two students collaborate** they must declare this at the top of their script and make a separate mark on their printout. These submissions are subject to particularly strict scrutiny for problematic constructions[2].
- For questions, clarifications and further help contact:

  Jan Sieber (`jan.sieber@port.ac.uk`, office LG.146).

[1] for example, the function works correctly most of the time but fails for some valid arguments

[2] more examples of problematic constructions (look also for warnings in the Matlab editor):

- hard-coded ‘magic’ numbers spread throughout the code,
- functions that should be general but only work for this example,
- one part of the code is a repetition of another part,
- stray brackets, misleading variable names or variable usages (say, using `x(:)` if `x` is scalar),
- arrays grow inside a loop,
- a variable is defined but not used.

Question 1: Finding roots of functions with weighted Regula Falsi

Background The standard *Regula Falsi* is a method to find roots of a function in an interval $[a,b]$ similar to the bisection method. Assume that $f(a)f(b)<0$. Regula Falsi determines a new point x_m as the intersection of the straight line connecting $(a,f(a))$ and $(b,f(b))$ with the x-axis:

$$x_m = \frac{f(a)\,b - f(b)\,a}{f(a)-f(b)}. \tag{1}$$

How it continues depends on the sign of $f(x_m)$:

(Case a) $f(x_m)$ has the same sign as $f(a)$: the new lower bound a is x_m, b stays the same;

(Case b) $f(x_m)$ has the same sign as $f(b)$: the new upper bound b is x_m, a stays the same;

(Case 0) $f(x_m)=0$: x_m is a root $\Rightarrow$ stop with $a=b=x_m$.

It turns out that this method is *worse* than simple bisection. An improvement is the *weighted Regula Falsi*, which works better than bisection most of the time. The weighted Regula Falsi follows exactly the same procedure above but it modifies the equation (1) for x_m slightly:

$$x_m = \frac{w_a\,b\,f(a) - w_b\,a\,f(b)}{w_a f(a) - w_b f(b)}, \tag{2}$$

where the weights w_a and w_b are set initially equal to 1. In later stages they are set like this:

- if the previous $k \geq 2$ iterations all were of (Case a) then $w_b = 2^{1-k}$ and $w_a = 1$
- if the previous $k \geq 2$ iterations all were of (Case b) then $w_a = 2^{1-k}$ and $w_b = 1$;
- otherwise $w_a = w_b = 1$

This means, for example, if one has moved up the lower boundary a $k \geq 2$ times in a row without changing the upper boundary b then one downweighs the function value $f(b)$ by a factor of $1/2^{k-1}$ (see also `wikipedia`, but note that `wikipedia`'s description is slightly misleading).

Tasks

(a) Write a function `WeightedRF` (in a function file `WeightedRF.m`) that finds the root of a function f using the above weighted Regula Falsi method. The first line of the file `WeightedRF.m` should look like this:

```
function [a,b]=WeightedRF(f,aini,bini,maxit,tol)
```

The meaning of the inputs has to be:

- `f`: a function of a single variable that you can call inside `WeightedRF`. The procedure is supposed to find a root of `f`;
- `aini`: initial lower bound for the root of `f`;
- `bini`: initial upper bound for the root of `f`; the root of `f` is to be found between `aini` and `bini`;

- `maxit` (positive integer): the iteration should stop as soon as the number of iterations exceeds `maxit`
- `tol` (small positive real number): tolerance; the iteration should stop if the difference between the current bounds is less than `tol`.

The meaning of the outputs has to be:

- `a`: a column vector of all lower bounds generated by the iteration;
- `b`: a column vector of all upper bounds generated by the iteration.

This means that the return values `a` and `b` are vectors with at most `maxit` elements, `a(1)` sbould be `aini`, and `b(1)` should be `bini`. If the iteration stops after `i<maxit` iterations are reached then `a` and `b` should contain `i` elements. [10 marks]

(b) Write a script `cw2.m` that tests `WeightedRF` on two functions, f_1 and f_2. The function f_1 is given by

$$f_1(x) = \exp(2x/p) + x$$

where $p = 0.\text{xxxyyy}$ and xxxyyy is your student ID. For example, if your student ID is 314159 then $p = 0.314159$. For the other inputs choose `aini=-1`, `bini=0`, `maxit=50`, `tol=1e-14`.

The function f_2 you have to choose such that $f_2(0) < 0$, $f_2(1) > 0$, f_2 is continuously differentiable (that is, its derivative is continuous), strictly monotone increasing in the interval $[0, 1]$ (that is, if $x < y$ then $f_2(x) < f_2(y)$), and has its root at p: $f_2(p) = 0$ ($p = 0.\text{xxxyyy}$ again). Moreover, you have to choose f_2 such that `WeightedRF` *fails* to reduce the the interval length below `tol=1e-6` for any choice of `maxit` less than 200 (the other inputs are `aini=0` and `bini=1`).

For each of the two functions plot the error $|a - b|$ on a logarithmic scale, for example using the command `semilogy` (remember, that `a` and `b` are vectors):

```
semilogy(abs(a-b)+eps,'.-');
grid on
```

Note that the `+eps` adds the tiny `eps` ($\approx 10^{-16}$), which is a built-in constant in Matlab (look up its help).

!!! A test script without a function f_2 satisfying the above conditions earns only 3 points**!!!** [7 marks]

Total for Question 1: 17 marks

Hints and further instructions

- The electronic upload must contain the files `WeightedRF.m` (a function file) and `cw2.m` (a matlab script).
- I will test your `WeightedRF` also with other inputs to see if it behaves sensibly when it fails to converge (say, setting `maxit=4`, but `tol=1e-14`), or trying a different function. You can rely on `aini<bini` and on `f(aini)*f(bini)<0` in my test inputs.
- While writing `WeightedRF` choose a simple test input, for example

  ```
  [a,b]=WeightedRF(@(x)exp(x)+x,-1,0,10,1e-14)
  ```

  (this should find the root of $f(x) = \exp(x) + x$ in $[-1, 0]$). You may start with the standard Regula Falsi first and then introduce the weights w_a and w_b.

- A convenient way to define a function inside a script (without creating a separate `m`-file) is, for example `f=@(x)exp(x)+x;` which defines $f(x) = \exp(x) + x$. Look at Matlab's help for anonymous functions.
- If you want to number your functions inside `cw2.m` you should use curly braces. So, for example, `f{1}=@(x)exp(2*x/p)+x;`, and then pass on `f{1}` as the first input into `WeightedRF`.
- One criterion for *efficient* computation is that your function `WeightedRF` does not call `f` more often than necessary: if you perform `i` iterations you should call `f` only `i+1` times.
- Matlab's `break` and `continue` statements may be useful when you want to jump out of a loop early. Look at their help page.