MATH2019 (2020-2021)
Introduction to Scientific Computation
Coursework 1 Feedback
Hand-in date: 29 Oct 2020

Coursework details

- Number of submitted scripts: 236
- Maximum mark possible: 40
- Average mark: 31.2 (=78.0%)

Solution m-files

Solution m-files can be found on the next few pages. Obviously, many alternatives are possible.

Feedback

An individual CW1_MARKS_REPORT for your submission can be found in Moodle (CW1 Submission -> Feedback files), both in html and pdf format. This report contains details on the obtained marks and comments for each output generated by your functions.

Please have a careful look at your report, and let me know if you notice some unusual marking of your outputs.

It is very well possible that the automated marking script needs further refining.

Note: In your feedback files on Moodle, there is also a file CW1_MARKS_MANUAL.csv, which only contains the marks for the manually-marked parts. (Only useful if you wish to generate your CW1_MARKS_REPORT yourself; see below.)

General feedback on performance of the class

Generally, the overall student's performance was very good.

The mean was 31.2 out of 40 marks. There were many very high marks (>35), and some marks <20.

Q2 (bisection_func): mean = 82%.

Well done by most.

Q3 (bisectionStop_func): mean = 73%

Well done by many. Some students did not have the proper length of the output vector.

Q4 (fpiter_func): mean = 82%

Well done by most.

Q5 (newton_func): mean = 85%

Well done by most.

Q6 (optParamFPiter_func): mean = 63%

This was the most difficult question, but still well done by many.

Automated marking algorithm

Apologies for the delayed return of your marks and feedback.

This was the first time that we have been using automated marking (for 85% of the submission). There were still some parts of the submissions (15%) that were manually marked by the Teaching Assistants (those involving the function descriptions).

The automation was the cause of significant delay: It has been much underestimated how much fine-tuning the automation needs, in particular:

- in the awarding of marks for a very large variety of near-correct answers (instead of zero marks),
- and in applying a penalty-reduction for carry-forward mistakes.

Precise tests carried out on your functions, and generating your own report

If you wish to know which tests were done to obtain the outputs, how these outputs were marked, or how the report was generated, please download the Coursework 1 Marking Details Folder on Moodle. This folder contains the following files

- GenerateCW1outputsForMarking.m – A function m-file that contains all the tests that were run. Please simply open this file in Matlab to see all test details. (This file is very similar to GenerateYourOutputsCW1.m .)
- CW1_TRUEOUTPUTS_FOR_MARKING.mat – A mat-file that contains the true outputs considered to be correct. (This output comes from the solution m-files; see on the next pages.)
- MarkCW1outputs.m – A function m-file that compares your outputs with the true outputs, assigns marks per output and generates the CW1 Marks Report.

To generate your own CW1_MARKS_REPORT, place the above files, and the file CW1_MARKS_MANUAL.csv (download from feedback files in CW1 submission) in the same folder as your written functions, and run the following:

```
[output,cmdW] = GenerateCW1outputsForMarking();
load('CW1_TRUEOUTPUTS_FOR_MARKING.mat','trueOutput');
[Marks,MarksReport] =
MarkCW1outputs(output,cmdW,trueOutput);
```

Once the above has run, have a look at the content of:

```
Marks.obtainable
Marks.obtained
Marks.comment
MarksReport
```

and the files:

CW1_MARKS_ALL.csv
html/CW1_MARKS_REPORT.pdf
html/CW1_MARKS_REPORT.html

MATH2019 (2020-2021)
Introduction to Scientific Computation
Coursework 1 Solution

Contents

1 m-file: bisection_func.m (Question 2)

```
function p_vec = bisection_func(f,a,b,Nmax)
%BISECTION_FUNC   Bisection Method
%   p_vec = BISECTION_FUNC(f,a,b,Nmax)
%    computes the vector of approximations p_vec
%    obtained by the bisection method
%    applied to function f,
%    using Nmax iterations,
%    and interval [a,b].

for i = 1:Nmax
    p = (a+b)/2;
    if f(p)*f(a)>0
        a = p;
    else
        b = p;
    end
    p_vec(i,1) = p;
end
```

2 m-file: bisectionStop_func.m (Question 3)

```
function p_vec = bisectionStop_func(f,a,b,Nmax,TOL)
%BISECTIONSTOP_FUNC   Bisection Method
%   p_vec = BISECTIONSTOP_FUNC(f,a,b,Nmax,TOL)
%    computes the vector of approximations p_vec
%    obtained by the bisection method
%    applied to function f,
%    using a maximum of Nmax iterations,
%    and TOL as a tolerance,
%    and interval [a,b].

% Write your code here
for i = 1:Nmax
    p = (a+b)/2;
    if f(p)*f(a)>0
        a = p;
    else
        b = p;
    end
    p_vec(i,1) = p;
    if i>1 && abs(p_vec(i)-p_vec(i-1))  < TOL
        break
    end
end
```

3 m-file: fpiter_func.m (Question 4)

```
function p_vec = fpiter_func(f,c,p0,Nmax)
%FPITER_FUNC   Fixed-Point Iteration
%   p_vec = fpiter_func(f,c,p0,Nmax)
%    computes the vector of approximations p_vec
%    obtained by fixed-point iteration
%    applied to function g(x) = x-c*f(x),
%    using Nmax iterations,
%    and initial guess p0.

g= @(x) x - c*f(x);
p = p0;
for i = 1:Nmax
  p = g(p);
  p_vec(i,1) = p;
end
```

4 m-file: newton_func.m (Question 5)

```
function p_vec = newton_func(f,dfdx,p0,Nmax)
%NEWTON_FUNC   Newton's Method
%   p_vec = newton_func(f,dfdx,p0,Nmax)
%    computes the vector of approximations p_vec
%    obtained by Newton's method
%    applied to function f
%    with derivative dfdx,
%    using Nmax iterations,
%    and initial guess p0.

for i = 1:Nmax
  p = p0 - f(p0)/dfdx(p0);
  p_vec(i,1) = p;
  p0 = p;
end
```

5 m-file: optParamFPiter_func.m (Question 6)

```
function [c_opt,N_opt] = optParamFPiter_func(f,p0,p,TOL,Nmax)
%OPTPARAMFPITER_FUNC   Optimal-Parameter Fixed-Point Iteration
%   [c_opt,N_opt] = optParamFPiter_func(f,p0,p,TOL,Nmax)
%    computes the optimal value c=c_opt
%    and corresponding number of iterations N_opt
%    for the fastest fixed-point iteration
%    applied to function g(x) = x-c*f(x),
%    to reach stopping criterion determined by TOL,
%    using initial guess p0,
%    and Nmax iterations,
%
%    Note: This function computes all possible values of c_opt,
%          instead of just one.

cR = (0.001:0.001:1)';
NR = zeros(size(cR));
for i=1:length(cR)
    c = cR(i);
    p_vec = fpiter_func(f,c,p0,Nmax);
    e = abs(p-p_vec);
    e(end+1) = 0.99*TOL;
    NR(i) = find(e<TOL,1);
end

% Following code sufficient to find one optimum
if 0
    [N_opt,imin] = min(NR);
    c_opt = cR(imin);
end

% Following code to find all optima
if 1
    N_opt = min(NR);
    imin = find(NR == N_opt);
    c_opt = cR(imin);
end
```