Exam 2022 Numerical Methods

Søren Degn Abrahamsen
Soabr19

Note
All Code has been written by me unless stated in the specific exercise. The *utility.h* file has some functions which has been implemented in collaboration with Victor Dahl Herlev and Mads Fogh Høffer.

The code used to solve this assignment can be found in *main.cpp* and functions that are used in main.cpp can be found in *utility.h* and *helper_functions.h*.

Exercise 1

i)

This exercise is solved using *Cholesky.h* from NR.
Since no specification for a design matrix has been provided, I will assume that no design matrix needs to be implemented. The Diagonal elements of L can be seen below

```
Diagonal elements of L  MatrixDiag 6x6:
        3.06035         2.70952         3.25561         1.52257         2.66776         1.31079
```

ii)

The solution x to Ax=b, computed by Cholesky Decomposition, can be seen below

```
The solution x to Ax=b  Vector 6D:
      -0.320979        0.182261       -0.211888       -0.328873        0.159704       -0.184628
```

iii)

U and V will be square and have the same size.

Exercise 2

This exercise is solved using the *newt* function from *roots_multidim.h* with some extra code to print certain values.

i)

$$f_0(1,-1) = 14.635314$$
$$f_1(1,-1) = -13.550080$$

ii) & iii)

i	x1	x2	dk	C	e
1	0.704807	-0.717825	0.408364	inf	inf
2	0.440817	-0.605803	0.286775	1.71968	0.141426
3	0.303597	-0.506673	0.169281	2.05838	0.058985
4	0.219186	-0.462968	0.0950544	3.31708	0.0299709
5	0.124482	-0.475378	0.0955131	10.5711	0.0964371
6	0.0847747	-0.461469	0.0420733	4.61191	0.00816386
7	0.070061	-0.459548	0.0148386	8.38259	0.00184571
8	0.0623783	-0.457377	0.00798357	36.2587	0.00231104
9	0.061773	-0.457545	0.000628108	9.85461	3.88783e-06
10	0.0615344	-0.457527	0.000239172	606.238	3.46789e-05

As seen in the table above, in 10 iterations we obtain

$$u_A = 0.0615344$$
$$u_B = -0.457527$$

To achieve a proven accuracy of 10^{-4} we need to run 9 iterations. The convergence is calculated as

$$C = \frac{|d_k|}{|d_{k-1}|^2}$$

and the error is calculated as

$$e = C|d_k|^2$$

which is taken from the estimations of Newton's method in 1 dimension. As explained in lecture 6, the convergence can be set to 1000 because Newton's method converges too quickly to estimate the actual value of C. If the convergence is set to 1000 and the error is calculated as above, we obtain the following table.

i	x1	x2	dk	C	e
1	0.704807	−0.717825	0.408364	1000	166.761
2	0.440817	−0.605803	0.286775	1000	82.2401
3	0.303597	−0.506673	0.169281	1000	28.6561
4	0.219186	−0.462968	0.0950544	1000	9.03534
5	0.124482	−0.475378	0.0955131	1000	9.12275
6	0.0847747	−0.461469	0.0420733	1000	1.77017
7	0.070061	−0.459548	0.0148386	1000	0.220184
8	0.0623783	−0.457377	0.00798357	1000	0.0637374
9	0.061773	−0.457545	0.000628108	1000	0.000394519
10	0.0615344	−0.457527	0.000239172	1000	5.72034e−05

In the table we see that it takes 10 iterations to achieve the desired accuracy.

Exercise 3

i)

When rewriting the second order ODE's to first order, a variable change is introduced. Note, since the two second order ODE's are almost identical in symbols besides the subscript, the below is general for both of them.

$$u(t) = y(t)$$
$$v(t) = y'(t)$$

u and v are differentiated yielding

$$u'(t) = y'(t)$$
$$v'(t) = y''(t) = abg - aby(t) - ay'(t) + f\big(x(t)\big)$$

The first order ODE then is

$$u'(t) = v(t)$$
$$v'(t) = abg - abu(t) - av(t) + f\big(x(t)\big)$$

Now I state all 5 ODE's and initial conditions

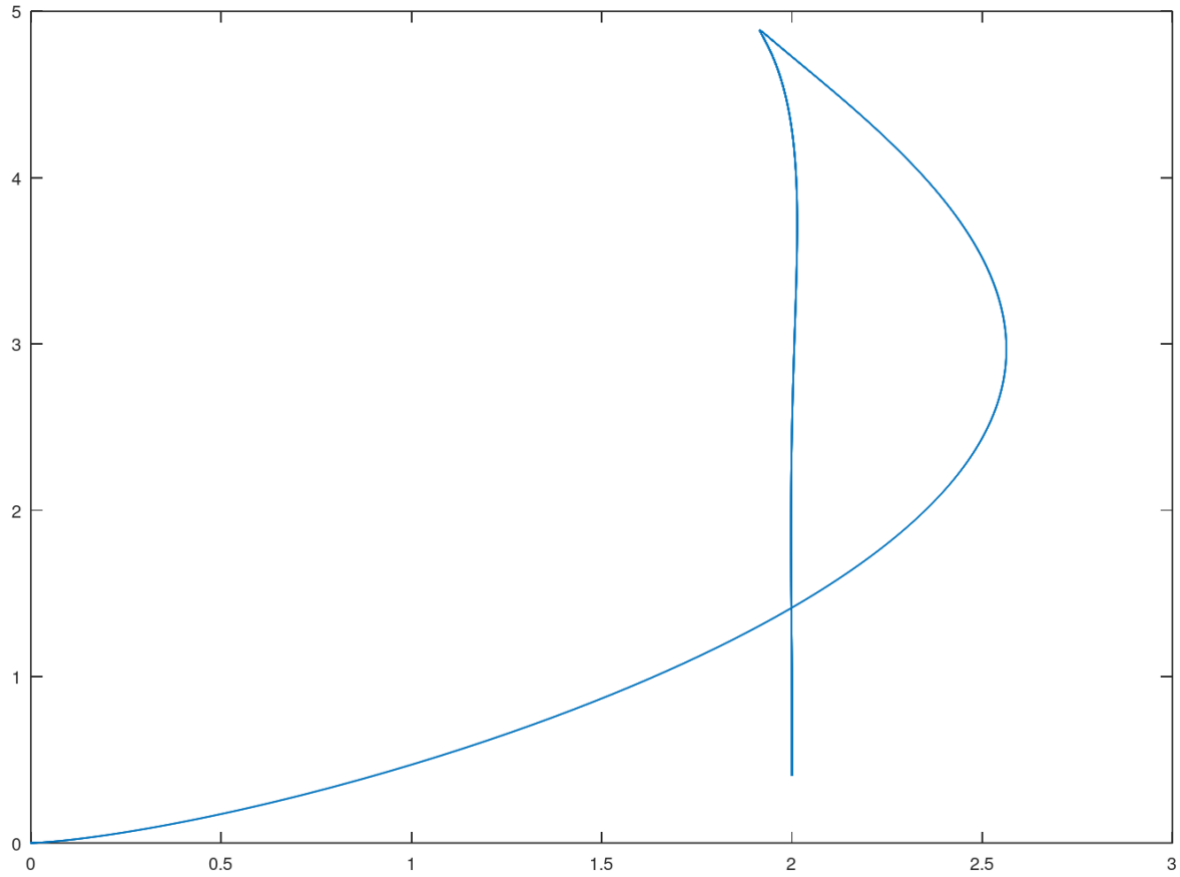
$$u_1'(t) = v(t)$$
$$v_1'(t) = a_1 b_1 g_1 - a_1 b_1 u_1(t) - a_1 v_1(t) + f_1\big(x(t)\big)$$
$$u_1(0) = Y_1 \quad v_1(0) = 0$$

$$u_2'(t) = v_2(t)$$
$$v_2'(t) = a_2 b_2 g_2 - a_2 b_2 u_2(t) - a_2 v_2(t) + f_2\big(x(t)\big)$$
$$u_2(0) = Y_1 \quad v_2(0) = 0$$

$$x'(t) = -x(t)$$
$$x(0) = 1$$

ii)

The plot is made using Octave – the command used to generate the plot is provided in the *main.cpp*.



The plot above shows the trajectory $\left(y_1(t), y_2(t)\right)$ where $0 \leq t \leq 20$. For clarification, the x-axis represents $y_1(t)$ and the y-axis represents $y_2(t)$.

iii)

MIDPOINT
Richardson for y(2)

i	A(hi)	A(hi-1)-A(hi)	Rich-alpk	Rich-error	f-computations
1	0.03125	inf	inf	inf	5
2	4.78012	-4.74887	inf	inf	10
3	4.53427	0.24585	4	-0.0819499	20
4	4.47269	0.0615871	4	-0.020529	40
5	4.45777	0.0149202	4	-0.00497342	80
6	4.45412	0.00364539	4	-0.00121513	160
7	4.45322	0.00089942	4	-0.000299807	320
8	4.453	0.000223288	4	-7.44293e-05	640

As seen in the table above, 8 iterations resulting in 640 subdivisions are needed to achieve an accuracy better than 10^{-4}.

Exercise 4

The self-implemented *FiniteDiff* method in *helper_functions.h* uses *tridag.h* from NR.

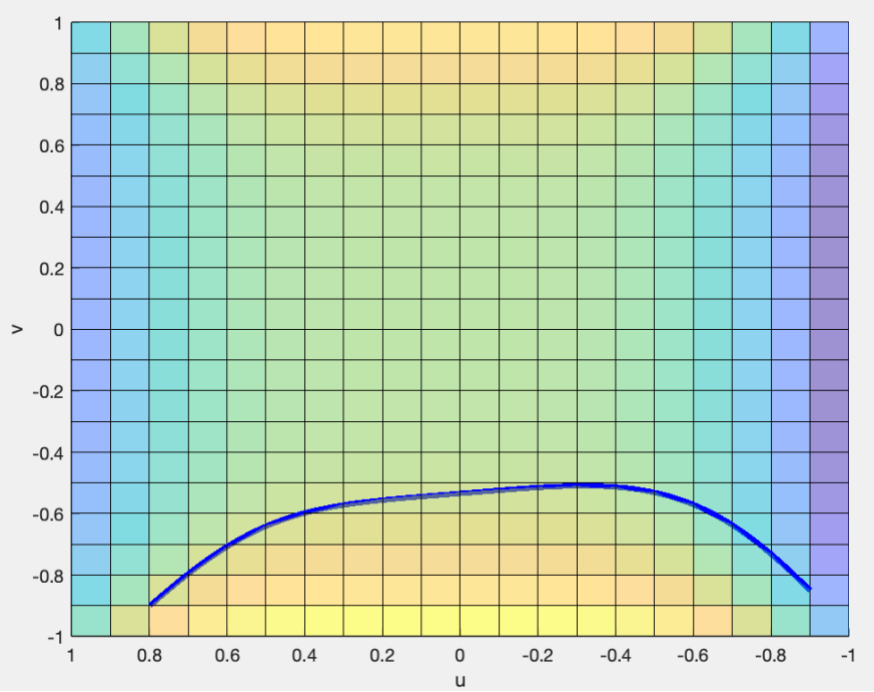
i)

i	A(hi)	A(hi-1)-A(hi)	Rich-alpk	Rich-error	f-computations
1	-0.875	inf	inf	inf	3
2	-0.868727	-0.00627293	inf	inf	5
3	-0.237484	-0.631243	0.00993742	-0.637579	9
4	-0.566814	0.32933	-1.91675	0.11291	17
5	-0.517244	-0.0495702	-6.64371	-0.0064851	33
6	-0.529062	0.0118181	-4.19443	0.00227515	65
7	-0.529191	0.000129158	91.5011	-1.42714e-06	129
8	-0.529171	-2.0371e-05	-6.3403	-2.77522e-06	257
9	-0.529165	-5.19735e-06	3.91949	1.78023e-06	513
10	-0.529164	-1.30465e-06	3.9837	4.3726e-07	1025

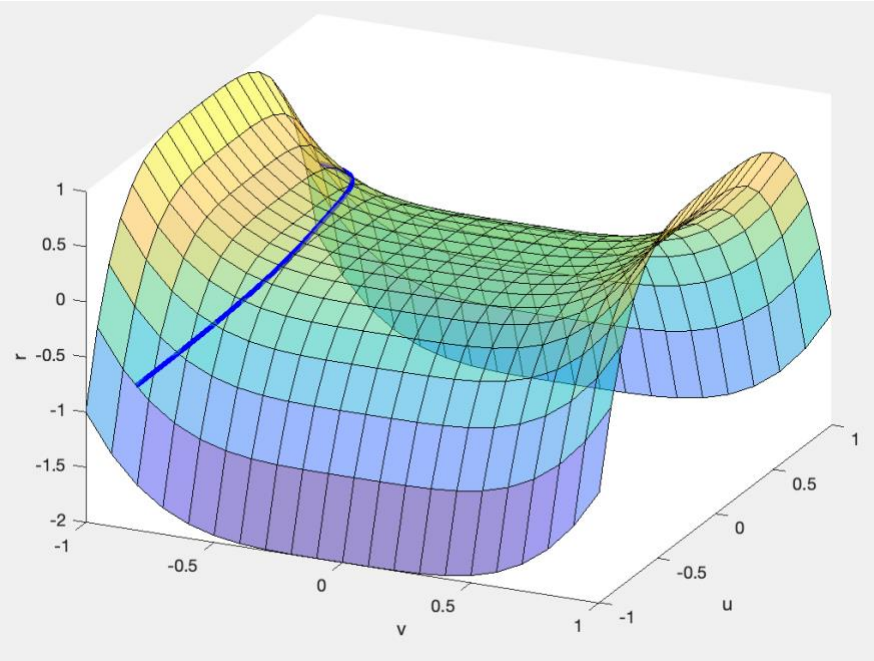
$V\left(\frac{u_A+u_B}{2}\right)$ is approximated to -0.529164 when the accuracy is better than 10^{-6}. The *f-computations* in the table shows the number of points used to estimate $V(u)$ satisfying the accuracy criterion. 1025 points where generated including the boundaries. If *N* is the number of subintervals between the boundaries, then $N = 1024$ when the accuracy was fulfilled.

ii)

The plots are made using Matlab – the file is provided in the folder as *Exercise4.m*



The plot above shows the $V(u)$ found in the last iteration of **i)** and the plot below shows the solution curve with the surface. The blue line is the solution curve.



Based on the plots and **i)** the result is deemed correct. The solution curve lays flat on the surface and the accuracy supports this.

Exercise 5

i)

For a single subdivision using the extended midpoint method, the integral can be estimated as described in the equation below

$$\int_a^b \frac{\cos(x^3)\, e^{-x}}{\sqrt{x}}\, dx = (b-a) \cdot \frac{\cos\left(\left(\frac{a+b}{2}\right)^3\right) e^{-\frac{a+b}{2}}}{\sqrt{\frac{a+b}{2}}}$$

ii)

EXTENDED MIDPOINT METHOD

i	A(hi)	A(hi-1)-A(hi)	Rich-alpk	Rich-error	f-computations
1	-0.0167948	inf	inf	inf	2
2	0.629306	-0.646101	inf	inf	4
3	1.01052	-0.38121	1.69487	0.548605	8
4	1.13552	-0.125003	3.0496	0.0609888	16
5	1.21473	-0.0792132	1.57806	0.137033	32
6	1.27796	-0.0632325	1.25273	0.2502	64
7	1.32279	-0.0448295	1.41051	0.109204	128
8	1.35432	-0.0315224	1.42215	0.0746713	256
9	1.37654	-0.0222198	1.41866	0.0530738	512
10	1.39222	-0.0156862	1.41652	0.0376599	1024
11	1.40331	-0.0110826	1.41539	0.0266798	2048
12	1.41114	-0.00783326	1.41481	0.018884	4096
13	1.41668	-0.00553777	1.41451	0.0133597	8192
14	1.42059	-0.00391538	1.41436	0.00944911	16384
15	1.42336	-0.00276844	1.41429	0.00668238	32768
16	1.42532	-0.00195753	1.41425	0.00472546	65536
17	1.4267	-0.00138416	1.41423	0.00334152	131072
18	1.42768	-0.000978746	1.41422	0.00236285	262144
19	1.42837	-0.000692075	1.41422	0.0016708	524288
20	1.42886	-0.00048937	1.41422	0.00118144	1.04858e+06
21	1.42921	-0.000346037	1.41421	0.000835404	2.09715e+06

The accuracy can be seen in the *Rich-error* column. Rich-error is calculated derived as explained in the course

$$Rich - error = \frac{A(h_2) - A(h_1)}{\alpha^k - 1}$$

where

$$\alpha^k = \frac{A(h_1) - A(h_2)}{A(h_2) - A(h_3)}$$

I'm having some trouble understanding what exactly needs to be stated in the *f-computation* column but in this case the number of subdivisions is given. For this case, $4.1943 \cdot 10^6$ is needed to achieve an accuracy better than 10^{-3}. This means, for the first row of f-computations, 2 integrals are estimated and added, in the next row 4 integrals are estimated and added together. To achieve a better accuracy than stated, 2 million sub-integrals are estimated between 0 and 4 and added together. This is a lot of subdivisions!

Iii)

The function has a singularity when $x = 0$, therefore the DE-rule is applied with the trapezoidal method. The results can be seen in the table below.

DE-RULE WITH TRAPEZOIDAL

i	A(hi)	A(hi-1)-A(hi)	Rich-alpk	Rich-error	f-computations
1	6.32826e-14	inf	inf	inf	1
2	-0.119745	0.119745	inf	inf	2
3	0.481777	-0.601522	-0.19907	-0.501657	4
4	1.55213	-1.07036	0.561983	-2.44364	8
5	1.45569	0.0964408	-11.0986	0.00797125	16
6	1.34653	0.109167	0.883421	0.936424	32
7	1.40737	-0.060848	-1.7941	-0.0217773	64
8	1.43004	-0.0226697	2.68411	0.0134609	128
9	1.43004	-1.07361e-11	2.11154e+09	5.08447e-21	256

The trapezoidal method is used with the new function, given by the DE-rule, and the integral is approximated on $x \in [-4.3, 4.3]$, resulting in only 256 subdivisions and 9 iterations. The Rich-alpk and Rich-error columns in the above table, should be disregarded as they provide no meaningful information when using the DE-rule.