Danmarks Tekniske Universitet

02685 Assignment 02

Partial Differential Equations

Arturo Arranz Mateo (s160412)
Manuel Montoya Catala (s162706)

April 2017

Contents

1 Exercise 1: 2-point BVPs

In this exercise, we want to solve the nonlinear BVP (equation (2.91) in the book) with Dirichlet boundary conditions:

$$\begin{aligned} \epsilon u'' - u(u'-1) \quad 0 \le t \ge 1 \\ u(0) = \alpha \quad u(1) = \beta \end{aligned} \tag{1}$$

using a certain number of methods. For the single shooting method, it means that we will transform the BVP (1) into the following initial value problem (IVP)

$$\begin{aligned} \epsilon u'' - u(u'-1) \quad 0 \le t \ge 1 \\ u(0) = \alpha \quad u'(0) = \sigma \end{aligned} \tag{2}$$

where σ is unknown and must be chosen to satisfy the boundary condition $\mu_\sigma(1) = \beta$. You may use ode45 or any other Runge-Kutta solver (including the solvers you developed for Assignment 1) to numerically solve (2). For this exercise, we will take $\alpha = -1, \beta = 1.5$, and you may start considering $\epsilon = 0.01$.

1.1 Newton's method for solving nonlinear BVPs

1.1.1 Second-order accurate scheme

A possible approximation to (1) would be

$$0 = \frac{\epsilon}{h^2}(u_{i-1} - 2u_i + u_{i+1}) + u_i(\frac{1}{2h}(u_{i+1} - u_{i-1}) - 1) \tag{3}$$

if we Taylor expand each of the terms $u_{i\pm h}$ and substitute u_i for its real value $u(x_i)$

$$0 = \frac{\epsilon}{h^2}(h^2u''(x_i) + \frac{h^4}{12}u''''(x_i) + \mathcal{O}(h^6)) + u_i[\frac{1}{2h}(2hu'(x_i)\frac{h^3}{3}u'''(x_i) + \mathcal{O}(h^5)) - 1] \tag{4}$$

Then the LTE error would be

$$\begin{aligned} \tau_i &= f_i - f(x_i) \\ &= \frac{\epsilon h^2}{12}u''''(x_i) + \frac{h^2}{6}u(x_i)u''(x_i) + \mathcal{O}(h^3) \end{aligned} \tag{5}$$

This demonstrates that the scheme is $\mathcal{O}(h^2)$.

1.1.2 Rewrite (2) as an IVP in standard form, ie.

To do so, we simply rename the first derivative u' as v, so we have a new variable $v = u'$ and u'' can be therefore expressed as $u'' = v'$.

$$\begin{aligned} u' = v \quad 0 \le t \ge 1 \\ v' = \frac{u(v-1)}{\epsilon} \quad 0 \le t \ge 1 \\ u(0) = \alpha \quad v(0) = \sigma \end{aligned} \tag{6}$$

1.1.3 Solve the BVP (1) using bvp4c

```
options=bvpset('reltol',1e-2,'abstol',1e-2);
```

and set an initial guess on you grid by using (for instance)

```
solinit = bvpinit(linspace(0,1,10),[0 0]);
```

1.1.4 BVP using the Newton's method

If we define the function $G_i = G(u(x_i))$

$$G_i = \frac{\epsilon}{h^2}(u_{i-1} - 2u_i + u_{i+1}) + u_i(\frac{1}{2h}(u_{i+1} - u_{i-1}) - 1) \tag{7}$$

And its Jacobian as

$$DG = \begin{bmatrix} \frac{\partial G_1}{\partial u_0} \frac{\partial G_1}{\partial u_1} & \frac{\partial G_1}{\partial u_2} & \cdots \\ \frac{\partial G_2}{\partial u_1} & \frac{\partial G_2}{\partial u_2} & \frac{\partial G_2}{\partial u_3} \\ \vdots & & \ddots \end{bmatrix} \tag{8}$$

Then we can find the function u(x) by finding the roots of G_i. Newton's method, as explained in the LeVeque's book[2], has being used with initial guess (2.106). In the figure 1 the solution for $\epsilon = 0.05$ is shown.

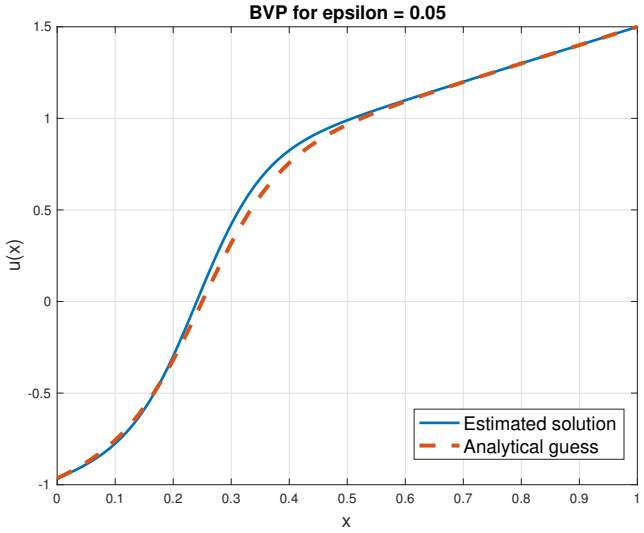
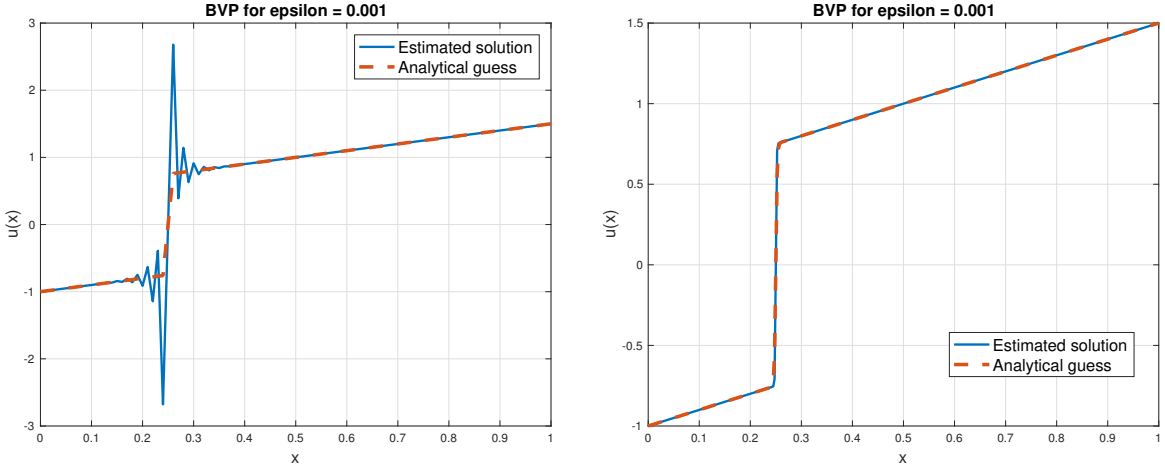


Figure 1: Estimated solution and initial guess

If test it for smaller values of epsilon maintaining the same number of points on the grid the LTE grows. This is because higher epsilon implies more abrupt changes hence higher derivatives, which the LTE depends of. This can be solved by augmenting the number of points on the grid, making h smaller. In figure 2 the empirical results are shown.



(a) 100 grid points

(b) 400 grid points

Figure 2: Estimated solution and initial guess $\epsilon = 0.001$

1.2 Single shooting

Single shooting methods aim to solve BVPs as IVPs. The purpose is to find the initial acceleration, $u'(0)$, from (6) for which $u_\sigma(1) - \beta = 0$.

1.2.1 Solve using a single shooting method.

Newton's method has being chosen for solving the shooting problem since it is a second order method, hence, it is more efficient [1]. If we define $G(\sigma) = u(1, \sigma) - \beta$ then Newtons method can be used to find the roots as:

$$\begin{aligned} \sigma^{[n+1]} &= \sigma^{[n]} - \frac{G(\sigma^{[n]})}{G'(\sigma^{[n]})} \\ &= \sigma^{[n]} - \frac{u(1, \sigma^{[n]}) - \beta}{\frac{\partial u(1, \sigma^{[n]})}{\partial \sigma}} \end{aligned} \tag{9}$$

However, we need to define $G'(\sigma)$. For doing so we need to know how our system change respect to σ i.e. deriving (6) using the chain rule:

$$\begin{aligned} \frac{u''(x)}{\partial \sigma} = \frac{\partial f(u(x,\sigma), u'(x,\sigma))}{\partial \sigma} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial \sigma} + \frac{\partial f}{\partial u'}\frac{\partial u'}{\partial \sigma} \\ \frac{\partial u(0,\sigma)}{\partial \sigma} = 0 \quad \frac{\partial u'(0,\sigma)}{\partial \sigma} = 1 \end{aligned} \tag{10}$$

if we define

$$\frac{\partial u}{\partial \sigma} = w(x) \tag{11}$$

then we obtain

$$\begin{aligned} w'' = \frac{1}{\epsilon}(1 - u')w - \frac{u}{\epsilon}w' \\ w(0) = 0 \quad w'(0) = 1 \end{aligned} \tag{12}$$

which can be solve with a standard numerical solver. Now it is possible to calculate newtons method as

$$\sigma^{[n+1]} = \sigma^{[n]} - \frac{u(1, \sigma^{[n]}) - \beta}{w(1)} \tag{13}$$

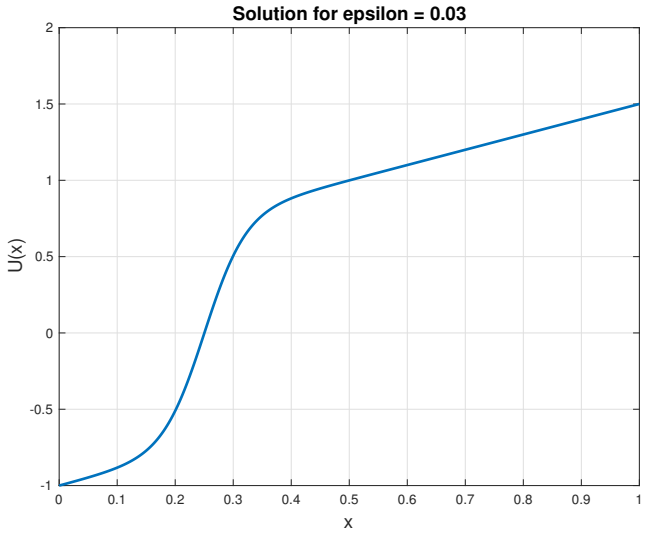


Figure 3: Estimated solution

For $\epsilon = 0.05$ Newton's method converges to 3. The initial value $u'(0)$ that corresponds $u(1) = 1.5$ is $\sigma = 1.2303$.

1.2.2 Sensitivity analysis

The sensitivity of the solution at the final time $t = 1$ with respect to σ has being calculated, and it is shown in the figure 4

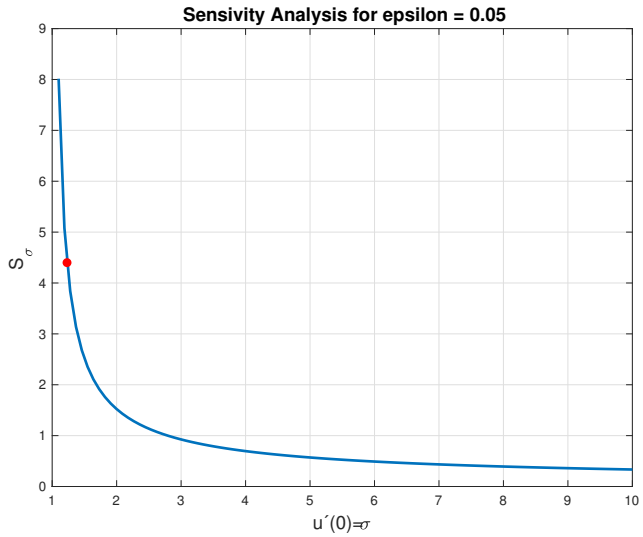


Figure 4: Sensivity of the solution

The red dot corresponds to $\sigma = 1.2303$ which is the solution for $\epsilon = 0.05$. As we can see the solution is very sensitive which implies that a small change will converge far away from $u(1) = 1.5$

1.2.3 Try the single shooting with different values of ϵ

If epsilon decrease the sensitivity of the system increases, as the numercial results in figure 5 indicate. This will imply that Newtons method will have convergence problems.

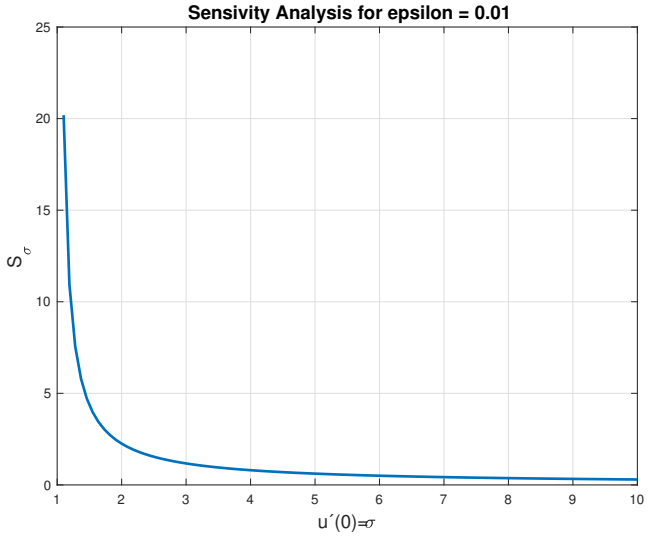
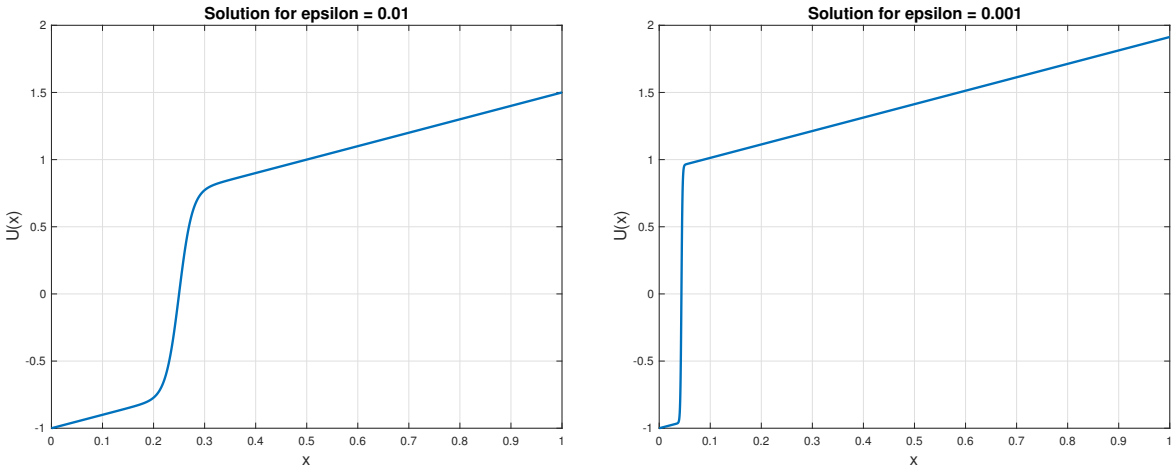


Figure 5: Sensivity for $\epsilon = 0.01$

For the soltuion in figure 6a it tooks 382 iterations to converge, while for 6b it did not converge after reaching the maximum number of iterations. One way to solve this problem would be by providing a better initial guess.



(a) $\epsilon = 0.001$ (b) $\epsilon = 0.001$

Figure 6: Estimated solution and initial guess $\epsilon = 0.001$

2 Exercise 2: 9-point Laplacian

2.1 Introduction

Given the *Poisson problem*

$$f = u_{xx} + u_{yy} \tag{14}$$

We can approximate the function $u(x, y)$ with a 5-points or 9-points stencil scheme. Both schemes have the same accuracy, but the latter has a very convenient expression for the LTE , which the higher order term depends on $\nabla^2 f$. Since the function *f(x,y)* is usually known, we can incorporate this information to our solver for achieving $\mathcal{O}(h^4)$ accuracy. Assuming same spacing in each dimension, h, the expression for the 9-points stencil scheme is given by

$$\nabla_9^2 u_{ij} = \frac{1}{6h^2}[4u_{i-1,j} + 4u_{i+1,j} + 4u_{i,j-1} + 4u_{i,j+1} + u_{i-1,j-1} + u_{i+1,j-1} + u_{i-1,j+1} + u_{i+1,j+1} + 20u_{i,j}] \tag{15}$$

and after Taylor expanding and subtracting the real value $u(x_i, y_i)$ the LTE is given by

$$\tau_{ij} = \frac{1}{12}h^2(u_{xxxx} + 2u_{xxyy} + u_{yyyy}) + \mathcal{O}(h^4) \tag{16}$$

Then the fourth order accurate scheme is given by

$$\nabla_9^2 u_{ij} = f(x_i, y_j) + \frac{h^2}{12}\nabla^2 f(x_i, y_j) \tag{17}$$

note that $f(x_i, y_j)$ means the real value while f_{ij} is an approximation. The same applies to the function u(x,y).

2.2 Implementation

For solving u(x,y) it is necessary to solve the linear system AU = F

$$A = \frac{1}{6h^2}\begin{bmatrix} T & S & & & \\ S & T & S & & \\ & S & T & S & \\ & & \ddots & \ddots & \ddots \\ & & & S & T \end{bmatrix} \tag{18}$$

where

$$T = \begin{bmatrix} 20 & 4 & & & \\ 4 & 20 & 4 & & \\ & 4 & 20 & 4 & \\ & & \ddots & \ddots & \ddots \\ & & & 4 & 20 \end{bmatrix} \qquad S = \begin{bmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & 1 & 4 & 1 & \\ & & \ddots & \ddots & \ddots \\ & & & 1 & 4 \end{bmatrix} \tag{19}$$

$$U = \begin{bmatrix} u^{[1]} \\ u^{[2]} \\ \vdots \\ u^{[m]} \end{bmatrix} \qquad where, u^{[j]} = \begin{bmatrix} u_{1j} \\ u_{2j} \\ \vdots \\ u_{mj} \end{bmatrix} \tag{20}$$

Since most of elements of the arrays are 0, sparse storage has being used.

2.3 Results

The first equation to solve is

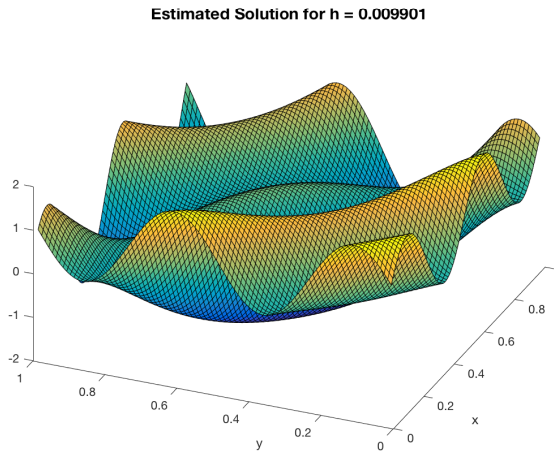
$$u_{0,exact}(x, y) = sin(4\pi(x + y)) + cos(4\pi xy) \tag{21}$$

then we can calculate f as in (14)

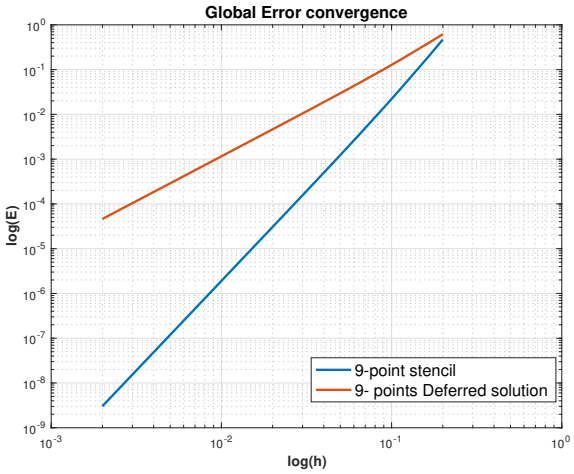
$$f(x, y) = (4\pi)^2[-2sin(4\pi(x + y)) - cos(4\pi xy)(x^2 + y^2)] \tag{22}$$

and the expression for the *deferred corrections*

$$\nabla^2 f(x,y) = (4\pi)^2[64\pi^2 sin(4\pi(x+y)) - 4cos(4\pi xy) + 32\pi xysin(4\pi xy) + 16\pi^2 cos(4\pi xy)(x^2+y^2)^2] \quad (23)$$



(a) Estimated solution



(b) Global error

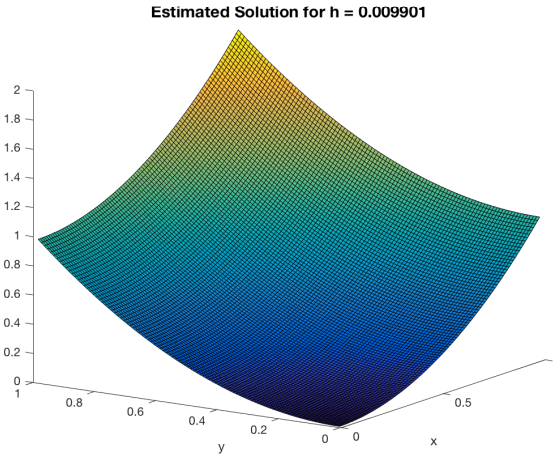
Figure 7: Equation (21)

As **??** shows, the 9-points stencil global error converges with a rate $\mathcal{O}(h^2)$ while with the deferred corrections it converges with $\mathcal{O}(h^4)$ rate.

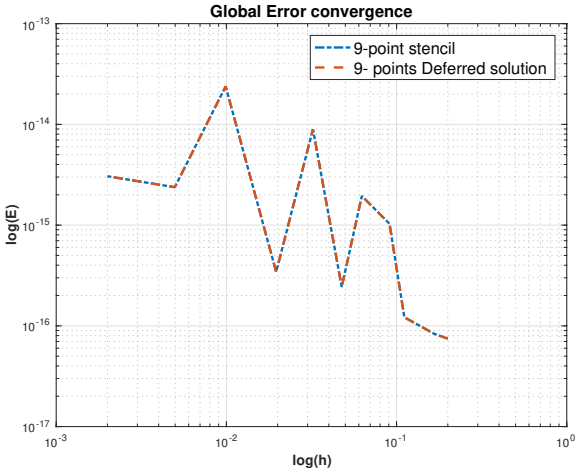
Now turn to a second equation

$$\begin{aligned} u_{1,exact} &= x^2 + y^2 \\ f(x,y) &= 4 \\ \nabla^2 f(x,y) &= 0 \end{aligned} \quad (24)$$

If we look at the expression of the LTE (16). We see that the higher term depends on the fourth derivative of u(x,y). In this case the fourth derivative is zero, hence including the correcting terms will not make it more accurate. *Actually, our solution is can not being improve.* The variance shown in **??** are probably because machine precision and if we look at the magnitude, we can assume that the error is 0.



(a) Estimated solution



(b) Global error

Figure 8: Equation (24)

To solve the derivatives of the $u_{2,exact}(x,y)$ equation:

$$u_{2,exact}(x,y) = sin(2\pi|x-y|^{2.5}) \quad (25)$$

We will use the Chain Rule in this regard, we will call $z = |x-y|$ and $\phi = 2\pi z^{2.5}$. Knowing the properties of the derivative of the absolute value function $\frac{d|x|}{dx} = \frac{|x|}{x} = sign(x)$ and the chain, product and quotient rules and moving the terms around we have:

$$\begin{aligned} u_{2,xx} &= -\frac{5\pi(x-y)^2}{2z^{1.5}}\Big(10\pi z^{2.5} sin(\phi) - 3cos(\phi)\Big) \\ u_{2,yy} &= -\frac{5\pi(y-x)^2}{2z^{1.5}}\Big(10\pi z^{2.5} sin(\phi) - 3cos(\phi)\Big) \end{aligned} \tag{26}$$

Due to the fact that $(x-y)^2 = (y-x)^2$ we have that the function $f(x,y)$ is:

$$f(x,y) = u_{2,xx} + u_{2,yy} = -\frac{10\pi(x-y)^2}{2z^{1.5}}\Big(10\pi z^{2.5} sin(\phi) - 3cos(\phi)\Big) \tag{27}$$

To compute the 4th derivatives we would have to follow the same process. We have used an automatic solver to get the result since the number of terms became very big. In this case we also have that $u_{2,yyyy} = u_{2,xxxx}$

$$\begin{aligned} \nabla^2 f(x,y) =& 2 - \frac{5\pi}{8z^{\frac{13}{2}}}((z(80\pi z^{\frac{13}{2}} + 380\pi(x-y)^2 z^{\frac{9}{2}} - 70\pi(x-y)^4 z^{\frac{5}{2}}) \\ & - 1000\pi^3(x-y)^{10} z^{\frac{5}{2}})\sin(\phi) + 3(x-y)^4 z + 1800\pi^2(x-y)^{10})\cos(\phi)) \end{aligned} \tag{28}$$

For some reason the solver is unable to calculate the estimated solution, so we are unable to make error analysis. In the figure 9 the true solution is displayed.

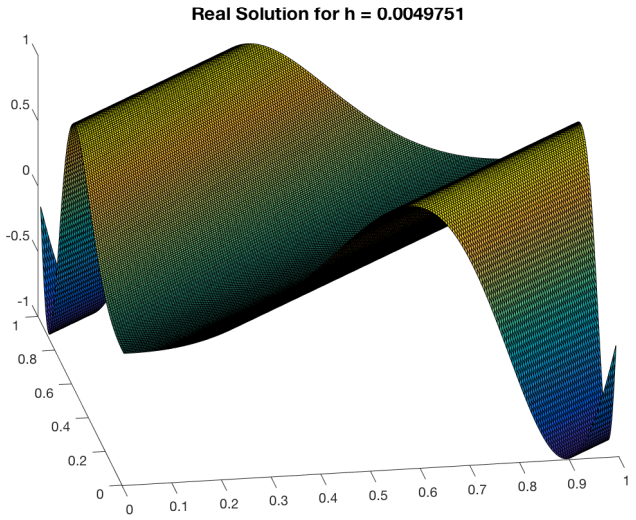


Figure 9: Solution for $u_{2,exact}$

References

[1] Gabriella Sebestyen *Numerical Solution of Two-Point Boundary Value Problems*

[2] Randall J. LeVeque *Finite Difference Methods for Ordinary and Partial Differential Equations*

A Code files

A.1 Problem 1

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Exercise 1.1: Newton?s method for solving nonlinear BVPs  %%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add subscripts
addpath('functions');
%% Ex 1.1.3 BVP
epsilon = 0.05;
twoode = @(t,x,epsilon) [x(2); -x(1)*(x(2)-1)/epsilon];
twobc = @(xa,xb,epsilon) [xa(1)+1; xb(1)-1.5];
options=bvpset('reltol',1e-2,'abstol',1e-2);
solinit = bvpinit(linspace(0,1,10),[-1 1.5]);
sol = bvp4c(twoode,twobc,solinit,options,epsilon);

figure(1)
hold on
    plot(sol.yp(2,:))
    grid on
hold off

%% Ex 1.1.4: BVP Finite Differences Method for non-linear equations

close all
%System parameters
m = 400;                              %number of points on grid
h = 1/(m);
x = linspace(0,1,m+1);                  %x gird
epsilon = 0.001;
alpha = -1;                           %u(0)=alpha
beta = 1.5;                           %u(1)=beta
params = [alpha beta epsilon];

%Analytic guess of solution
a = 0;
b = 1;
w0 = 1/2*(a-b+beta-alpha);
x_hat = 1/2*(a+b-alpha-beta);
u_guess = x-x_hat+w0*tanh(w0*(x-x_hat)/(2*epsilon));

%Newtons method
U = newtonODE(@functions11,u_guess',h,10e-8,epsilon);

% Plotting
lw = 2;
plot(x,U,'LineWidth',lw)
hold on
    plot(x,u_guess,'--','LineWidth',lw+1)
%     plot(x(2:n-1),x(2:n-1)+alpha-a,'-.','LineWidth',lw)
%     plot(x(2:n-1),x(2:n-1)+beta-b,'-.','LineWidth',lw)
    leg = legend('Estimated solution','Analytical guess');
    set(leg,'FontSize',14);
```

```
    ep = num2str(epsilon);
    title(['BVP for epsilon = ' ep],'FontSize',14);
    ylabel('u(x)','FontSize',14)
    xlabel('x','FontSize',14)
    grid on
hold off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%   Exercise 1.2: Shooting Methods  %%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add paths
addpath('functions');

%% Ex 1.2.1 Shooting methods through Newtons root finding

close all
%System paramters
alpha = -1;          % u(0)
beta = 1.5;         % u(1)
sigma0 = 1.001;       % u'(0) guess
epsilon = 0.001;
sigma1 = sigma0;
%Solver parameters
options = odeset('RelTol',1e-6);
tspan = [0 1];
%Newton method paramters
tol = 10e-5;
err = 1000;
itmax = 1000;
it = 0;
%Newtons method begins
while(err>tol && it<itmax)
    sigma = sigma1;
    Y0 = [alpha sigma 0 1];
    [t,y] = ode45(@(x,u) func12Newton(x,u,epsilon), tspan, Y0, options);
    G = y(end,1)-beta;
    dG = y(end,3);
    sigma1 = sigma - G/dG;
    err = norm(y(end,1)-beta);
    it = it+1;
end

figure(1)
plot(t,y(:,1),'LineWidth',lw)
hold on
    ep = num2str(epsilon);
    title(['Solution for epsilon = ' ep],'FontSize',14);
    ylabel('U(x)','FontSize',14)
    xlabel('x','FontSize',14)
    grid
hold off


%% Ex 1.2.2 Sensivity Analysis

close all
%Memory allocation
n = 100;
uList = zeros(n,1);                 % u(1) for different sigmas
wList = zeros(n,1);                 % du(1)/dsigma for different sigmas
%System parameters
```

```
alpa = -1;                          % u(0)
beta = 1.5;                         % u(1)
sigmaList = linspace(1.1,10,n);     % u'(0) guess list
epsilon = 0.001;
%Solver parametes
options = odeset('RelTol',1e-6);
tspan = [0 1];

for i = 1:n
    Y0 = [alpa sigmaList(i) 0 1];
    [t,y] = ode45(@(x,u) func12Newton(x,u,epsilon), tspan, Y0, options);
    uList(i) = y(end,1);
    wList(i) = y(end,3);
end

%Plotting
lw = 2;
figure(1)
plot(sigmaList,wList,'LineWidth',lw)
hold on
    ep = num2str(epsilon);
    title(['Sensivity Analysis for epsilon = ' ep],'FontSize',14);
    ylabel('S_\sigma','FontSize',14)
    xlabel('u(0)=\sigma','FontSize',14)
    grid
hold off

figure(2)
plot(sigmaList,uList,'LineWidth',lw)
hold on
    grid
    title(['u(1,\sigma) vs Sigma for epsilon = ' ep],'FontSize',14);
    ylabel('u(1,\sigma)','FontSize',14)
    xlabel('u(0)=\sigma','FontSize',14)
hold off

%% Ex 1.2.1 With psudo-secant method

close all
alpa = -1;      % u(0)
beta = 1.5;     % u(1)
sigma = 1.1;    % u'(0) guess
epsilon = 0.001;
Y0 = [alpha sigma];
options = odeset('RelTol',1e-6);
tspan = [0 1];
tol = 10e-6;

error = 1000;
c = [1.1 1.3];
while(error > tol)
    [t,y] = ode45(@(x,u) func12(x,u,epsilon), tspan, [alpha c(1)], options);
    c1 = y(end,1)-beta;
    [t,y] = ode45(@(x,u) func12(x,u,epsilon), tspan, [alpha c(2)], options);
    c2 = y(end,1)-beta;
    l = 0.5*(c(1)+c(2));
    [t,y] = ode45(@(x,u) func12(x,u,epsilon), tspan, [alpha l], options);
    c3 = y(end,1)-beta;
    if(c3 < 0)
        c(1) = l;
    else
```

```
        c(2) = l;
    end
    error = norm(c1-c2);
end
[t,y] = ode45(@(x,u) func12(x,u,epsilon), tspan, [alpha c(2)], options);
plot(t,y(:,1))

function [ Uk ] = newtonODE(functions,U0,h,tol,epsilon)
%Newton root finding algorithm
%   Find the roots of non-linear set of functions G(u)
% INPUT:
%    functions  : function handle for G and J
%    h :    step size
%    U0   :  vector of length m with intial guess for every u(x)
%    tol :  tolerance for the solution
% OUTPUT:
%    Uk  : estimated root
    Uk = U0;
    [G,J] = functions(Uk,h,epsilon);
    maxIt = 100;
    it = 0;
    while((it < maxIt) && (norm(G,'inf') > tol))
        it = it +1;
        Uk(2:end-1) = Uk(2:end-1) - J\G;
        [G,J] = functions(Uk,h,epsilon);
    end
end

function f = func12(x,u,epsilon)
%DEPPREY Summary of this function goes here
%   Detailed explanation goes here

    % Syntax: xdot = PreyPredator(t,x,a,b)
    f = zeros(2,1);
    f(1) = u(2);
    f(2) = (u(1)-u(1)*u(2))/epsilon;
end

function f = func12Newton(x,u,epsilon)
%DEPPREY Summary of this function goes here
%   Detailed explanation goes here

    % Syntax: xdot = PreyPredator(t,x,a,b)
    f = zeros(4,1);
    f(1) = u(2);
    f(2) = (u(1)-u(1)*u(2))/epsilon;
    f(3) = u(4);
    f(4) = (u(1)-1)*u(3)/epsilon - u(1)*u(4)/epsilon;
end

function [G,J] = functions11(u,h,epsilon)
%%% Caluclate the multivariate non-linear funcation G(u) and its Jacobian
% J(u)

G = epsilon/h^2*( u(1:end-2) - 2*u(2:end-1) + ...
    u(3:end))+ u(2:end-1).*((u(3:end) - u(1:end-2))/(2*h) - 1);

J = 1/h^2 * (diag(-2*epsilon + 0.5*h*(u(3:end) - u(1:end-2))-h^2) ...
    + diag(epsilon - 0.5*h*u(3:end-1),-1) + ...
    diag(epsilon + 0.5*h*u(2:end-2),1));

end
```

A.2 Problem 2

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%   Exercise 2: 9-points Laplacian  %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Add paths
addpath('functions');
%% Parametes
close all
m = 500;
span = [0 1];
h = abs(span(2)-span(1))/(m+1);
x = linspace(span(1)+h,span(2)-h,m);
y = linspace(span(1)+h,span(2)-h,m);
%RHS and boundary functions

sol31 = @(x,y) sin(4*pi*(x+y))+cos(4*pi*x.*y);
func31 = @(x,y) (4*pi)^2*(-2*sin(4*pi*(x+y))-cos(4*pi*x*y)*(x^2+y^2));
lapla31 = @(x,y) (4*pi)^2*(64*pi^2*sin(4*pi*(x+y))-4*cos(4*pi*x*y)+ ...
    32*pi*x*y*sin(4*pi*x*y)+16*pi^2*cos(4*pi*x*y)*(x^2+y^2)^2);
functions1 = {func31,sol31,lapla31};

sol32 = @(x,y) x.^2 + y.^2;
func32 = @(x,y) 4;
lapla32 = @(x,y) 0;
functions2 = {func32,sol32,lapla32};

sol33 = @(x,y) sin(2*pi*abs(x-y).^(2.5));
func33 = @(x,y) 0;
lapla33 = @(x,y) 0;
functions3 = {func33,sol33,lapla33};

% sol33 = @(x,y) sin(2*pi*(abs(x-y).^(2.5)));
% func33 = @(x,y)  2 * -5*pi*(x^2-2*y*x+y^2)* ...
%     (10*pi*abs(x-y)^(5/2)*sin(2*pi*abs(x-y)^(5/2))...
%     -3*cos(2*pi*abs(x-y)^(5/2)))/(2*abs(x-y)^(3/2));
% lapla33 = @(x,y) 2 * -5*pi*((abs(x-y)*...
%     (80*pi*abs(x-y)^(13/2)+(380*pi*x^2-760*pi*y*x+380*pi*y^2)...
%     *abs(x-y)^(9/2)+(-70*pi*x^4+280*pi*y*x^3-420*pi*y^2*x^2+....
%     280*pi*y^3*x-70*pi*y^4)*abs(x-y)^(5/2))+...
%     (-1000*pi^3*x^10+10000*pi^3*y*x^9- ...
%     45000*pi^3*y^2*x^8+120000*pi^3*y^3*...
%     x^7-210000*pi^3*y^4*x^6+252000*pi^3*y^5*x^5-...
%     210000*pi^3*y^6*x^4+120000*pi^3*y^7*x^3-45000*...
%     pi^3*y^8*x^2+10000*pi^3*y^9*x-1000*pi^3*y^10)*...
%     abs(x-y)^(5/2))*sin(2*pi*abs(x-y)^(5/2))+...
%     ((3*x^4-12*y*x^3+18*y^2*x^2-12*y^3*x+3*y^4)...
%     *abs(x-y)+1800*pi^2*x^10-18000*pi^2*y*x^9+...
%     81000*pi^2*y^2*x^8-216000*pi^2*y^3*x^7+378000*...
%     pi^2*y^4*x^6-453600*pi^2*y^5*x^5+378000*pi^2*y^6*...
%     x^4-216000*pi^2*y^7*x^3+81000*pi^2*y^8*x^2-18000 ...
% *pi^2*y^9*x+1800*pi^2*y^10)*cos(2*pi*abs(x-y)^(5/2)))/(8*abs(x-y)^(13/2));
% functions3 = {func33,sol33,lapla33};

%% 9-points Laplacian on function 1
close all
calculateError = 0;             %calculate and plot Error convergence
plotting = 1;                   %show plots
corrected = 0;                  %deferred corrections
funcs = functions3;
%Create scheme and RHS
```

```
A = poisson9(m);
F = form_RHS9(m,funcs{1},funcs{2},funcs{3},x,y,corrected);
%Get estimated solution
Uvec = A\F;
Uhat = vecToMatrix(m,Uvec);
%Get real solution
[X,Y]=meshgrid(x,y);
Usol = funcs{2}(X,Y);
E = h*norm(Usol-Uhat); %Global error

%Global error convergence
if(calculateError==1)
    mList = [4,5,8,10,15,20,30,50,100,200,500];
    [Elist9,hList] = globalError(mList,funcs{1},funcs{2},...
        funcs{3},span,corrected);
end
%Global error convergence withouf deferred correction
if(calculateError==1 && corrected ==1)
    [Elist9_deferred,hList] = globalError(mList,funcs{1}, ...
        funcs{2},funcs{3},span,0);
end
%Plotting
if(plotting==1)
    Elist = Elist9;
    ex3plot
end

%%  5-points stencil
close all
calculateError = 1;             %calculate and plot Error convergence
plotting = 1;                   %show plots
funcs = functions3;

%Create scheme and RHS
A = poisson5(m);
F = form_RHS5(m,funcs{1},funcs{2},x,y);
%Get estimated solution
Uvec = A\F;
Uhat = vecToMatrix(m,Uvec);
%Get real solution
[X,Y]=meshgrid(x,y);
Usol = funcs{2}(X,Y);
E = h*norm(Usol-Uhat); %Global error

%Global error convergence
if(calculateError==1)
    mList = [4,5,8,10,15,20,30,50,100,200,500];
    [Elist5,hList] = globalError5(mList,funcs{1},funcs{2},span);
end
%Plotting
if(plotting==1)
    Elist = Elist5;
    ex3plot
end

function F = form_RHS9(m,f,bound,lapla,x,y,referred)
%Return right-hand side of a 2D multidimensional linear system
%AU = F where A m^2xm^2 matrix and F = m^2x1 vector
%f(i,j) = f(x(i),y(j)) = F(m*i+j)
    h = 1/(m+1);
    %Dirichlet BC
```

```
    x0 = x(1)-h; xend = x(end)+h;
    y0 = y(1)-h; yend = y(end)+h;
    u0j = bound(x0,[y0 y yend]);        %BC vector along u(x(0),y(j))
    uendj = bound(xend,[y0 y yend]);    %BC vector along u(x(end),y(j))
    ui0 = bound([x0 x xend],y0);        %BC vector along u(x(i),y(0))
    uiend = bound([x0 x xend],yend);    %BC vector along u(x(i),y(end))

    F = zeros(m*m,1);
    for j = 1:m
        for i =1:m
            F((j-1)*m+i) = feval(f,x(i),y(j));
        end
        %Dirilecht BC in x;
        F((j-1)*m+1) = F((j-1)*m+1) - 4*u0j(j+1)/(6*h^2) - u0j(j+2)/(6*h^2) ...
            - u0j(j)/(6*h^2);
        F(j*m) = F(j*m) - 4*uendj(j+1)/(6*h^2) - uendj(j)/(6*h^2) - ...
            uendj(j+2)/(6*h^2);
    end
    % Dirilecht BC in y
    % boundary conditions along y(0)
    for i = 1:m
        F(i) = F(i) - 4*ui0(i+1)/(6*h^2) - ui0(i)/(6*h^2) - ui0(i+2)/(6*h^2);
    end
    % corners u(0,0) and u(1,0) have beaing added twice. We substract once
    F(m) = F(m) + ui0(end)/(6*h^2);
    F(1) = F(1) + ui0(1)/(6*h^2);

    % boundary conditions along y(end)
    for i = 1:m
        F(m*(m-1)+i) = F(m*(m-1)+i) -4*uiend(i+1)/(6*h^2) -uiend(i)/(6*h^2) ...
            -uiend(i+2)/(6*h^2);
    end
    % corners u(1,1) and u(0,1) have beaing added twice. We substract once
    F(m*m) = F(m*m) + uiend(end)/(6*h^2);
    F(m*(m-1)+1) = F(m*(m-1)+1) + uiend(1)/(6*h^2);

    if(referred==1)
        for j = 1:m
            for i =1:m
                F((j-1)*m+i) = F((j-1)*m+i)+(h^2/12)*feval(lapla,x(i),y(j));
            end
        end
    end
end

function A = poisson9(m)
%This function return the 9-points stencil shceme for the discrete
% Poisson equation
    e = ones(m,1);
    S = spdiags([e e e], [-1 0 1], m, m);
    C = spdiags([e 4*e e], [-1 0 1], m, m);
    B = spdiags([3*e -24*e 3*e], [-1 0 1], m, m);
    I = speye(m);
    A = kron(S,C)+kron(I,B);
    A = (m+1)^2*A./6;
end

function F = form_RHS5(m,f,bound,x,y)
%Return right-hand side of a 2D multidimensional linear system
%AU = F where A m^2xm^2 matrix and F = m^2x1 vector
%f(i,j) = f(x(i),y(j)) = F(m*i+j)
    h = 1/(m+1);
```

```
    x0 = x(1)-h; xend = x(end)+h;
    y0 = y(1)-h; yend = y(end)+h;
    F = zeros(m*m,1);
    for j = 1:m
        for i =1:m
            F((j-1)*m+i) = feval(f,x(i),y(j));
        end
        %Dirilecht BC in x
        F((j-1)*m+1) =F((j-1)*m+1) -feval(bound,x0,y(j))/h^2;
        F(j*m) = F(j*m) -feval(bound,xend,y(j))/h^2;
    end
    %Dirilecht BC in y
    for i = 1:m
        F(i) = F(i) - feval(bound,x(i),y0)/h^2;
    end
    for i = 1:m
        F(m*(m-1)+i) = F(m*(m-1)+i) - feval(bound,x(i),yend)/h^2;
    end
end

function A = poisson5(m)
    e = ones(m,1);
    S = spdiags([e -2*e e], [-1 0 1], m, m);
    I = speye(m);
    A = kron(I,S)+kron(S,I);
    A=(m+1)^2*A;
end

function [EList,hList] = globalError(mList,func,sol,lapla,span,corrected)
%GLOBAL_ERROR this function return the global error for each mList
%INPUTS
% mList: list of number of points
% func: function handle of RHS  AU = F
% sol:  function handle for the ture U solution
% x:    space discretization vector
% y:    space discretization vector
% span: grid size
%OUTPUS:
% Elist: global error for each mList
% hlist: step size for each mList

    n = length(mList);
    EList = zeros(n,1);
    hList = zeros(n,1);
    for i = 1:n
       m = mList(i);
       h =  abs(span(1)-span(2))/(m+1);
       x = linspace(span(1)+h,span(2)-h,m);
       y = linspace(span(1)+h,span(2)-h,m);
       %Create scheme
       A = poisson9(m);
       F = form_RHS9(m,func,sol,lapla,x,y,corrected);
       %Get solution
       Uvec = A\F;
       Uhat = vecToMatrix(m,Uvec);

       %true solution
       [X,Y]=meshgrid(x,y);
       Usol = sol(X,Y);

       %Global error
       EList(i) = h*norm(Usol-Uhat);
```

```
        hList(i) = h;
    end
end

function [EList,hList] = globalError5(mList,func,sol,span)
%GLOBAL_ERROR this function return the global error for each mList
%INPUTS
% mList: list of number of points
% func: function handle of RHS  AU = F
% sol:  function handle for the ture U solution
% x:    space discretization vector
% y:    space discretization vector
% span: grid size
%OUTPUS:
% Elist: global error for each mList
% hlist: step size for each mList

    n = length(mList);
    EList = zeros(n,1);
    hList = zeros(n,1);
    for i = 1:n
        m = mList(i);
        h =  abs(span(1)-span(2))/(m+1);
        x = linspace(span(1)+h,span(2)-h,m);
        y = linspace(span(1)+h,span(2)-h,m);
        %Create scheme
        A = poisson5(m);
        F = form_RHS5(m,func,sol,x,y);
        %Get solution
        Uvec = A\F;
        Uhat = vecToMatrix(m,Uvec);

        %true solution
        [X,Y]=meshgrid(x,y);
        Usol = sol(X,Y);

        %Global error
        EList(i) = h*norm(Usol-Uhat);
        hList(i) = h;
    end
end

function U = vecToMatrix(m,U0)
    U = zeros(m,m);
    for i = 1:m
        U(i,:) = U0((i-1)*m+1:m*i);
    end
end
```