MAT 425: Homework 9 (04/06)

Ex 1. We consider the following boundary-value problem (BVP):

$$\begin{cases} -x^2y'' - 2xy' + 2y = -4x^2 \\ y(0) = 0\ ,\ y(1) = 0. \end{cases}$$

a) Let $\varphi \in C_0^2([0,1])$ a 'test function'. Multiplying the ODE by φ and integrating in x yields:

$$\int_0^1 (-x^2y(x)')'\varphi(x) + 2y(x)\varphi(x)\,dx = \int_0^1 -4x^2\varphi(x)\,dx.$$ 1pt

Integrating by parts and using that $\varphi(0) = \varphi(1) = 0$, we find: 1pt

$$\int_0^1 -x^2y(x)'\varphi'(x) + 2y(x)\varphi(x)\,dx = \int_0^1 -4x^2\varphi(x)\,dx.$$

Thus, $\mathcal{A}(y,\varphi) = L(\varphi)$ with

$$\begin{aligned} \mathcal{A}(y,\varphi) &= \int_0^1 -x^2y(x)'\varphi'(x) + 2y(x)\varphi(x)\,\mathrm{d}x, \\ L(\varphi) &= \int_0^1 -4x^2\varphi(x)\,\mathrm{d}x. \end{aligned}$$ .5pt .5pt

b) The coefficients of the function y satisfy: $A\mathbf{c} = \mathbf{b}$. Thus, for any i, we have: 1pt

$$\begin{aligned} \Sigma_j a_{ij}c_j = b_i \quad &\Rightarrow \quad \Sigma_j \mathcal{A}(\phi_i,\phi_j)c_j = L(\phi_i) \\ &\Rightarrow \quad \mathcal{A}(\phi_i, y) = L(\phi_i) \end{aligned}$$

by linearity. Moreover, since $\mathcal{A}(u,v) = \mathcal{A}(v,u)$, we also deduce $\mathcal{A}(y,\phi_i) = L(\phi_i)$. Now taking any test function $\varphi = \Sigma_i\alpha_i\phi_i$:

$$\begin{aligned} \mathcal{A}(y,\varphi) &= \mathcal{A}(y, \Sigma_i\alpha_i\phi_i) = \Sigma_i\alpha_i\,\mathcal{A}(y,\phi_i) \\ &= \Sigma_i\alpha_i\,L(\phi_i) = L(\Sigma_i\alpha_i\phi_i) = L(\varphi). \end{aligned}$$ 1pt

c) The code is given in page 4 and the solution with $N = 10$ is given in figure 1. 2+2pts

d) To find the accuracy, one has to take care of the accuracy of the integral used to estimate the coefficients of A and b. Otherwise, the error stops decreasing (and actually increases) as N get larger. Thus, taking for the evaluation of the integral $\Delta x = \frac{1}{\mathbf{10}\cdot N}$, we obtain the figure 2 using the L^∞ norm. However, the estimation of the slope is only $c \approx .93$. Thus, the method, as it is implemented, is only close to 1st order accurate. 1pt

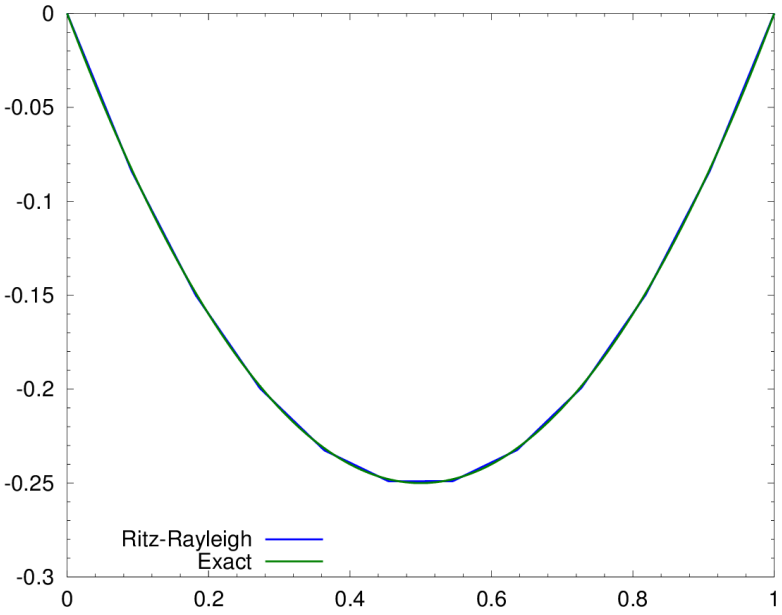

Figure 1: Solution of the BVP using the Ritz-Rayleigh method with $N = 10$ basis functions.

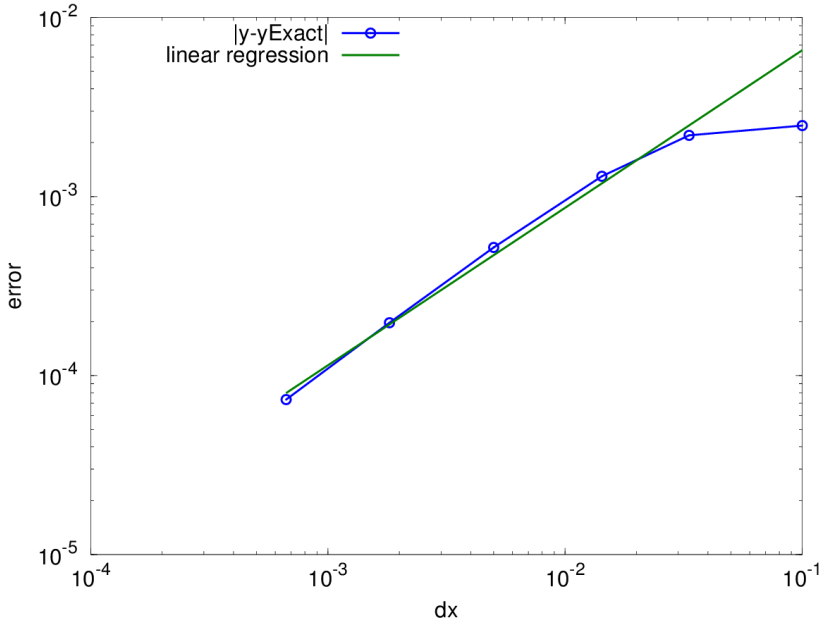

Figure 2: Accuracy of the Ritz-Rayleigh method using several basis functions. The method is close to 1st order accurate (the slope is $c \approx .93$).

Ex 2. The code is given in page 5. 1pt

We plot the solution for $\Delta x = \Delta y = \frac{1}{10}$ in figure 3. The exact solution of the elliptic PDE is known, thus we estimate the error of the numerical solution using the L^∞ in the following way: 1pt

$$\text{Error} = \|u_{exact} - u_{num}\|_\infty \approx \max_{i,j} |u_{exact}(x_i, y_j) - u_{i,j}|.$$

We find that the error is: $\text{Error} = 8.88 \cdot 10^{-16}$. 1pt

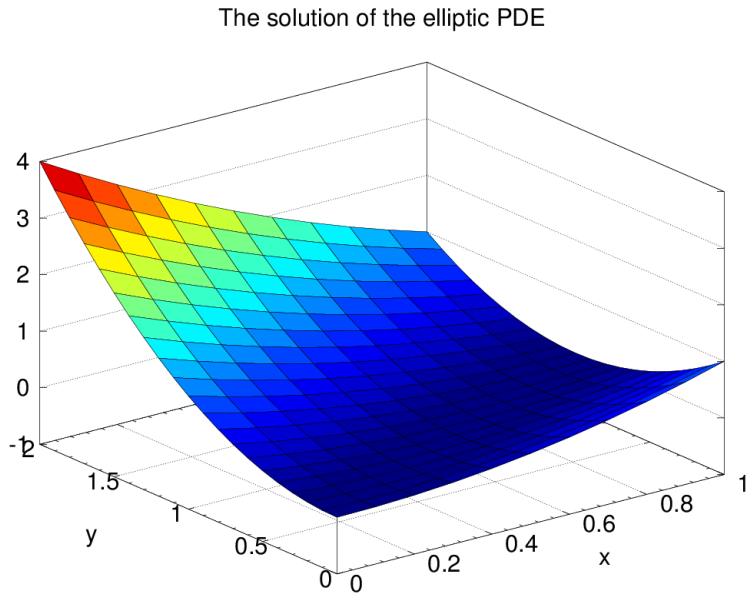

Figure 3: The solution of the elliptic PDE with $\Delta x = \Delta y = \frac{1}{10}$.

```
%% Solve the BVP using Ritz-Rayleigh method
%%   -(py')' + q(x) y = f(x)
%%   y(0)=y(1)=0
%%---- The Boundary Value Probem (BVP)
p = @(x) x;
q = @(x) 4+0*x;
f = @(x) 4*x.^2-8*x+1;
%% --- numerical parameters
N = 10;
dx = 1/(N+1);
%%--- estimatation matrix
A = zeros(N,N); b = zeros(N,1);
intX = linspace(0,1,1001);
%%-- construction of 'b'
for i=1:N
  %% phi_i and phi_i'
  [phi_i,dphi_i] = RR_phi(intX,i,N);
  %% estimation of L(phi_i)
  b(i) = trapz(intX,f(intX).*phi_i);
end
%%-- construction of 'A'
for i=1:N
  [phi_i,dphi_i] = RR_phi(intX,i,N);
  for j=(i-1):(i+1)
    if (j>=1 && j<=N)
      [phi_j,dphi_j] = RR_phi(intX,j,N);
      A(i,j) = trapz(intX, p(intX).*dphi_i.*dphi_j + q(intX).*phi_i.*phi_j);
    end
  end
end
%%-------------------------------%%
%%---    Resolution system    ---%%
%%-------------------------------%%
c = A\b;
%%-- construction solution
ySol = zeros(1,length(intX));
for i=1:N
  [phi_i,dphi_i] = RR_phi(intX,i,N);
  ySol = ySol + c(i)*phi_i;
end
%%-- plot
yExact = @(x) x.*(x-1);
plot(intX,ySol,intX,yExact(intX))
legend('Ritz-Rayleigh','Exact','location','southwest')
```

```
%% Solve the elliptic PDE
%%     u_xx + u_yy  =  f    in [a,b]x[c,d]
%%     u = g                at the frontier
%% the functions
f = @(x,y) 4;
g = @(x,y) (x-y)^2;
%% domain [a,b]x[c,d]
a = 0; b = 1;
c = 0; d = 2;
%% numerical parameters
n = 10; dx = (b-a)/n;
m = 20; dy = (d-c)/m;
%%--------------------------------%%
%%-----          FDM           -----%%
%%--------------------------------%%
lMax = (n-1)*(m-1);
A    = zeros(lMax,lMax);
vecB = zeros(lMax,1);
ij_to_l = @(i,j) i + (m-1-j)*(n-1);
for i=1:(n-1)
  for j=1:(m-1)
    %% init
    l    = ij_to_l(i,j);
    xi   = a+i*dx;
    yj   = c+j*dy;
    l_ip = ij_to_l(i+1,j);
    l_im = ij_to_l(i-1,j);
    l_jp = ij_to_l(i,j+1);
    l_jm = ij_to_l(i,j-1);
    %%---- the matrix A: stencil with 5 points
    %% diagonal
    A(l,l)  = -(2+2*dx^2/dy^2);
    vecB(l) = dx^2*f(xi,yj);
    %% extra diago: !! Danger border !!
    if (i!=1)
      A(l_im,l) = 1;
    else % x_i-1=a (left)
      vecB(l) = vecB(l) - g(a,yj);
    end
    if (i!=(n-1))
      A(l_ip,l) = 1;
    else % x_i+1=b (right)
      vecB(l) = vecB(l) - g(b,yj);
    end
```

```
    if (j!=1)
      A(l,l_jm) = 1;
    else % y_i-1=c (down)
      vecB(l) = vecB(l) - dx^2/dy^2*g(xi,c);
    end
    if (j!=(m-1))
      A(l,l_jp) = 1;
    else % y_i+1=d (up)
      vecB(l) = vecB(l) - dx^2/dy^2*g(xi,d);
    end
  end
end
%%------------------------%%
%%---     Solution     ---%%
%%------------------------%%
vecU = A\vecB;
%%--reconstruction matrix solution
matU = zeros(n+1,m+1);
for i=0:n
  for j=0:m
    if (i==0 || i==n || j==0 || j==m)
      %% at the border
      xi = a+i*dx;
      yj = c+j*dy;
      matU(i+1,j+1) = g(xi,yj);
    else
      %% inside
      l = ij_to_l(i,j);
      matU(i+1,j+1) = vecU(l);
    end
  end
end
%%-- plot
x = linspace(a,b,n+1);
y = linspace(c,d,m+1);
[X,Y] = meshgrid(x,y);
surf(X,Y,matU')
xlabel('x'); ylabel('y')
title('The solution of the elliptic PDE')
%%-- the error
zSol = @(x,y) (x-y).^2;
error = max(max( abs(matU'-zSol(X,Y)) ));
```

```python
## Solve the BVP using Finite-Element Method
##   -(py')' + q(x) y = f(x)
##   y(0)=y(1)=0
import numpy as np
import matplotlib.pyplot as plt
from Basis_Phi import *
##---- The Boundary Value Probem (BVP)
def p(x):
    return x**2
def q(x):
    return 2
def f(x):
    return -4*x**2
##--- numerical parameters
##-- vector space V
N = 19
dx = 1.0/(N+1)
##--- estimation matrix
A = np.zeros((N,N))
b = np.zeros(N)
##--
##-- construction of the matrix 'A' and vector 'b'
##--
intX = np.linspace(0,1,10000)               # to estimate the integral
for i in range(N):
    # a) phi_i and phi_i'
    phi_i,dphi_i = Basis_Phi(intX,i+1,N)
    # b) b_i = L(phi_i)
    b[i] = np.trapz(intX,f(intX)*phi_i)
    # c) estimation of A(phi_i,phi_j)
    for j in range(N):
        # phi_j and phi_j'
        phi_j,dphi_j = Basis_Phi(intX,j+1,N)
        # a_ij = A(phi_i,phi_j)
        A[i,j] = np.trapz(intX, p(intX)*dphi_i*dphi_j + q(intX)*phi_i*phi_j)
##------------------------------##
##---    Resolution system    ---##
##------------------------------##
C_tp = np.linalg.lstsq(A,b)
C = C_tp[0].reshape(-1)
##-- construction solution
ySol = np.zeros(len(intX))
for i in range(N):
    # a) phi_i and phi_i'
```

```
    phi_i,dphi_i = Basis_Phi(intX,i+1,N)
    # b) solution
    ySol += C[i]*phi_i
##-- plot
def yExact(x):
    return x*(x-1)
plt.clf()
plt.ion()
plt.plot(intX,yExact(intX))
plt.plot(intX,ySol,'r--',linewidth=4.0)
plt.xlabel(r'$x$',fontsize=25)
plt.ylabel(r'$y$',fontsize=25)
plt.legend(['Exact','Numerical Solution'],loc=2)
plt.show()
```

```
''' Solve the elliptic PDE
    dx^2 u + dy^2 u  =  f    on [a,b]x[c,d]
        u = g                on the border
'''
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
## the functions
def f(x,y):
    return 4
def g(x,y):
    return (x-y)**2
## domain [a,b]x[c,d]
a = 0.0
b = 1.0
c = 0.0
d = 2.0
## numerical parameters
n = 50
m = 40
dx = (b-a)/n
dy = (d-c)/m
##--------------------------------##
##-----         FDM          -----##
##--------------------------------##
lMax = (n-1)*(m-1)
A    = np.zeros((lMax,lMax))
vecB = np.zeros(lMax)
def ij_to_l(i,j):
    return i + (m-1-j)*(n-1)
for i in range(1,n):
    for j in range(1,m):
        ## init
        l    = ij_to_l(i,j)
        xi   = a+i*dx
        yj   = c+j*dy
        l_ip = ij_to_l(i+1,j)
        l_im = ij_to_l(i-1,j)
        l_jp = ij_to_l(i,j+1)
        l_jm = ij_to_l(i,j-1)
        ## the matrix A: stencil with 5 points
        A[l-1,l-1]  = -(2+2*dx**2/dy**2)
        vecB[l-1] = dx**2*f(xi,yj)
        ## extra diago: !! the border !!
```

```
        if (i!=1):
            A[l-1,l_im-1] = 1
        else: # x_i-1=a (left)
            vecB[l-1] -= g(a,yj)
        if (i!=(n-1)):
            A[l-1,l_ip-1] = 1
        else: # x_i+1=b (right)
            vecB[l-1] -= g(b,yj)
        if (j!=1):
            A[l-1,l_jm-1] = dx**2/dy**2
        else: # y_i-1=c (down)
            vecB[l-1] -= dx**2/dy**2*g(xi,c)
        if (j!=(m-1)):
            A[l-1,l_jp-1] = dx**2/dy**2
        else: # y_i+1=d (up)
            vecB[l-1] -= dx**2/dy**2*g(xi,d)
##------------------------##
##---     Solution     ---##
##------------------------##
vecU_tp = np.linalg.lstsq(A,vecB)
vecU = vecU_tp[0].reshape(-1)
##--reconstruction matrix solution
matU = np.zeros((n+1,m+1))
for i in range(n+1):
  for j in range(m+1):
    if ((i==0) or (i==n) or (j==0) or (j==m)):
      ## at the border
      xi = a+i*dx
      yj = c+j*dy
      matU[i,j] = g(xi,yj)
    else:
      ## inside
      l = ij_to_l(i,j)
      matU[i,j] = vecU[l-1]
##-- plot
x = np.linspace(a,b,n+1)
y = np.linspace(c,d,m+1)
X,Y = np.meshgrid(x,y)
fig = plt.figure()
ax = fig.gca(projection='3d')
surf = ax.plot_surface(X, Y, np.transpose(matU))
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.show()
```