University of Helsinki / Department of Mathematics and Statistics
SCIENTIFIC COMPUTING
Exercise 09 / Solutions

1. On the www-page there are program `hlp091.m` and `myf1d.m`. They were written in order to compute the line integral of a function $f(x,y) = c(1) * x^2 + c(2) * y^2 + c(3) * x * y + c(4) * x$ along the polygonal segment with vertices $(0,0),(2,0),(2,1)$. In this special case also the exact value of the integral was computed and compared to the numerical value.

(a) Run this program and study how it works. Use it to compute the line integral of $\sin(xy) + \exp(x - y)$ along the same path.

(b) Generalize this program to the three dimensional case.

Solution:

```
% FILE d091.m begins.
% FILE: h091.m
% uses myf1d.m
% Written after hlp091.m
function h091
n=2; % Part (a):
mypatha=[0 0 ; 2 0 ; 2 1]; % integration path
myfa=inline('sin(x(:,1).*x(:,2)) + exp(x(:,1)-x(:,2))','x','c');
c=rand(1,4);
vasta= mylineint(myfa, mypatha,c)

n=3; % Part (b):
mypathb=[0 0 0 ; 2 0 0 ; 2 1 0; 2 1 1]; % integration path

myfb=inline('sin(x(:,1).*x(:,2)) + exp(x(:,1)-x(:,2))+x(:,3)','x','c');

vastb= mylineint(myfb,mypathb,c)

% line integral
function y = mylineint(funfcn ,intpath,c)
[d1,d2]=size(intpath);
s=0;
for j=1:(d1-1)
    a=intpath(j,:);   b=intpath(j+1,:);
    s=s+quad('myf1d',0,norm(a-b),[],[],a,b,funfcn,c);
end
y = s;
%FILE d091.m ends.
```

```
function y = myf1d(x,a,b,myf,c)
    % x is real value in [0,norm(a-b)]
    % a is starting point in R^n
    % b is ending point in R^n
    % tmp is point x in R^n
    [d1,d2] = size(x);
    % d1 is assumed to be 1
    for j=1:d2
       tmp(j,:) = a+(b-a)*(x(j)/norm(a-b));
    end
    y = myf(tmp,c);
```

2. Let a be an $n \times n$ matrix with singular values $s_1 \geq s_2 \geq ... \geq s_n$, and eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$, $|\lambda_i| \geq |\lambda_{i+1}|$. Show that for all $k = 1, ..., n$, $|\lambda_1 \cdots \lambda_k| \leq s_1 \cdots s_k$.

Solution: The values of $s_1 \cdots s_k - |\lambda_1 \cdots \lambda_k|$ are printed Observe that all the printed numbers are positive, in concert with the assertion, except some numbers smaller than machine epsilon. Of course such small numbers are no counterexamples, but just reflect the round-off errors of floating point arithmetic.

```
% FILE d092.m begins.
% USES: d091f.m
for p=1:10
m= 2+ fix(2*(0.1 +rand));
a= rand(m,m)-0.5;
decreig=d091f(a);
[u,s,v]=svd(a);
dia=diag(s);
for pp=1:m
  t1=abs(prod(decreig(1:pp)));
  t2=prod(dia(1:pp));
  fprintf('%12.3e\n',t2-t1);
end
end
% FILE d092.m ends.
```

Output:

```
   4.490e-02
   4.254e-03
  -5.551e-17
   1.846e-01
   2.776e-17
   1.489e-01
   1.248e-01
   2.220e-16
   2.046e-01
  -2.776e-17
   2.671e-01
   7.104e-02
   0.000e+00
   7.775e-02
   6.959e-02
   3.816e-17
   1.712e-01
   1.388e-17
   3.901e-01
   7.547e-02
   1.648e-17
   2.709e-01
   1.729e-02
  -4.770e-18
   2.881e-01
   2.168e-01
   2.637e-16
```

3. The isoperimetric inequality says that if a curve of length L encloses a plane region with area A, then $A \le \pi(L/(2*\pi))^2$ [this upper bound is the area of a disk with boundary of length L]. In other words, the isoperimetric ratio $4*\pi*A/L^2 \le 1$.

(a) Familiarize yourself with the program myisoper.m on the www-page, which prompts the user to enter points in the plane and then plots a spline curve through the points and computes the isoperimetric ratio. Try to find examples of curves with as small isoperimetric ratio as possible.

(b) Simplify the program by excluding the use of splines and using polygons instead. For several sets of the same vertices, compute the ratios for both the spline and polygonal curves. What is your observation?

Solution: The program below is stored in the file d093.m. The command d093 executes the first function d093tst in the file. It may be noticed that the polygonal curve has smaller isoperimetric ratio than the spline curve. If we consider rectangles we see that the isoperimetric ratio tends to 0 when the ratio of of the sides tends to ∞.

```
% FILE: d093.m begins
function w=d093tst()

x=[0  3  2  1  0];
y=[0  0  3  2  1];
mypts=[x' y'];

myisoper(1, mypts);  % 1 yields spline curve
print -dps d093a.ps
myisoper(0, mypts);  % 0 yields polygonal curve
print -dps d093b.ps
% end of d093tst

function w=myisoper(curtype, mypts)
% if curtype = 0 make polygonal curve
% if curtype = 1 make spline curve
% USES: myplot.m, mygetpts.m
%clear;
close all;
if (nargin==1)
disp('Do not make x(n+1)=x(1)')
pts=mygetpts([-2 2 -2 2],1);
else
  pts=mypts;
end
clf
x=pts(:,1)';
y=pts(:,2)';
disp('Vertices of polygon:')
disp(pts)
n=length(x);    % Vertices of polygon
x(n+1)=x(1);    % Make the polygon closed
y(n+1)=y(1);
t(1) =0;        % Parameter values: length of polygon
for i=1:n,
  t(i+1)=t(i)+sqrt((x(i)-x(i+1))^2 + (y(i)-y(i+1))^2);
end;
t
for i=1:n,      % Refined division td
  for j=1:10,
    td(10*(i-1)+j) = t(i)*(9-j+1)/9.0 +t(i+1)*(j-1)/9.0;
  end;
end;
```

```
td
if (curtype ~=0)
xstart= (x(2)-x(n))/sqrt((x(2)-x(n))^2 + (y(2)-y(n))^2)
xstop= xstart
ystart= (y(2)-y(n))/sqrt((x(2)-x(n))^2 + (y(2)-y(n))^2);
ystop= ystart;
[xstart x xstop]
t
x
xd= spline(t,[xstart x xstop],td);  % Spline values at td
yd= spline(t,[ystart y ystop],td);
else
xd=x; yd=y;
end
myplot(x,y,'k.',xd,yd,'MarkerSize',30),
axis equal; grid on;
m=length(xd);
a2=polyarea(xd,yd);

disp(['polyarea = ' num2str(a2)])
patch(xd,yd,'y')
xlabel(['Area of shaded region = ' num2str(a2) ' (MYSPL)'])
le=sum( ((xd(1:m-1)-xd(2:m)).^2+(yd(1:m-1)-yd(2:m)).^2).^(0.5) );
ylabel(['Length of boundary = ' num2str(le)]);
myr=4*pi*a2/(le*le);
title(['Isoperimetric ratio = '  num2str(myr, '%8.4f') ]);
widemarg(gcf)
fprintf('A/(L*L)= %12.4e\n',myr )
w=myr;
% end of myisoper

function plt=myplot(varargin)
%  Purpose:  to label and plot in preferred font size etc.
axes('FontSize',[20],'FontWeight','bold');
hold on;
plt0=plot(varargin{:});
set(plt0,'LineWidth',2);
hold off
% end of  myplot

function w=mygetpts(bds,option)
% MYGETPTS plots a grid and prompts the user to
%        enter by mouse click points (x,y)
%        with bds(1) <x <bds(2), bds(3)<y<bds(4)
% nrx = number of grid subdivisions in x-direction
% nry = number of grid subdivisions in y-direction

if (nargin ==1)
  option=1;
end;

% First plot grid
nrx=10; nry=10;
figure(1)
hold on;
axis([bds(1)-0.1 bds(2)+0.1 bds(3)-0.1 bds(4)+0.1]);
axis('equal')
for i=1:(nrx+1)
  x=(bds(2)- bds(1))*(i-1)/nrx+ bds(1);
  line([ x x],[ bds(3 )  bds(4)]);
end;
for j=1:(nry+1)
  y=(bds(4)- bds(3))*(j-1)/nry+ bds(3);
  line([bds(1) bds(2)],[ y y]);
end
n=0;
% Get points with mouse:
disp('Left mouse button picks points.')
disp('Right mouse button picks last point.')
but = 1;
while (but == 1)
  [xi,yi,but] = ginput(1);
  plot(xi,yi,'go')
  n = n+1;
  x(n,1) = xi;
  y(n,1) = yi;
end
% Plot the points
pts=[x y];
plot(pts(:,1),pts(:,2),'ko','LineWidth',1.5)
plot(pts(:,1),pts(:,2),'b-','LineWidth',1.5)
% Prompt for correction:
[d1,d2]=size(pts);
if (option)
  yn=input('Enter 1/0 to edit/not to edit points: ')
else
```

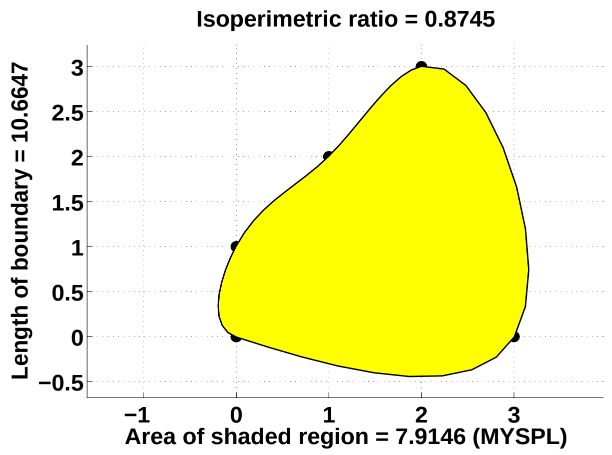
```
  yn=0;
end;
while ((yn==1)| (isempty(yn)==1))
  disp('Click with mouse the point to be edited. ')
  pointed=ginput(1);
  %find index of pointed point:
  index=1; tst=10;
  for i=1:d1
    tmp=norm(pointed-pts(i,:));
    if (tmp<tst)
      index=i; tst=tmp;
    end;
  end
  disp('Enter a new point: ');
  new=ginput(1);
  pts(index,:)=new;
% plot the grid again
  hold off;
  clf;
  axis([bds(1)-0.1 bds(2)+0.1 bds(3)-0.1 bds(4)+0.1]);
  axis('equal')
  for i=1:(nrx+1)
    x=(bds(2)- bds(1))*(i-1)/nrx+ bds(1);
    line([ x x],[ bds(3 )  bds(4)]);
  end;
  for j=1:(nry+1)
    y=(bds(4)- bds(3))*(j-1)/nry+ bds(3);
    line([bds(1) bds(2)],[ y y]);
  end
  hold on;
% plot the corrected points
  plot(pts(:,1),pts(:,2),'ko','LineWidth',1.5)
  plot(pts(:,1),pts(:,2),'b-','LineWidth',1.5)
  if (option)
    yn=input('Enter 1/0 to edit/not to edit points: ');
  else
    yn=0;
  end;
end;
% End of correction loop
hold on;
axis([bds(1)-0.1 bds(2)+0.1 bds(3)-0.1 bds(4)+0.1]);
axis('equal')
```

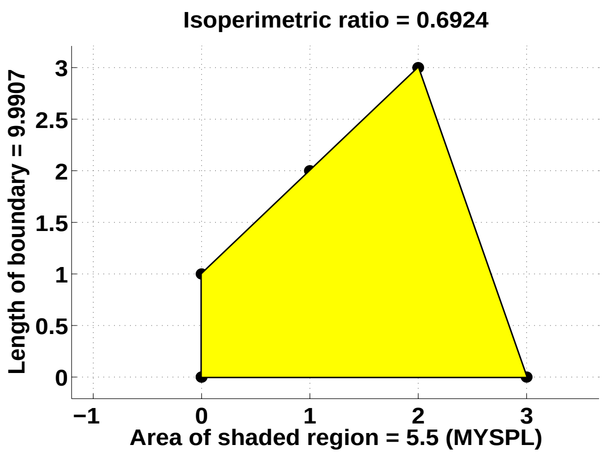
```
for i=1:(nrx+1)
  x=(bds(2)- bds(1))*(i-1)/nrx+ bds(1);
  line([ x x],[ bds(3 )  bds(4)]);
end;
for j=1:(nry+1)
  y=(bds(4)- bds(3))*(j-1)/nry+ bds(3);
  line([bds(1) bds(2)],[ y y]);
end
plot(pts(:,1),pts(:,2),'ko','LineWidth',1.5)
plot(pts(:,1),pts(:,2),'b-','LineWidth',1.5)
% Save points
if (option)
  ok=input('Enter 1/0 to save/not save points: ');
else
  ok=1;
end;
if (ok==1)
  fid=fopen('mygetpts.dat','w');
  fprintf(fid, '%12.6f %12.6f \n',pts');
  fclose(fid);
  disp('Points saved in the file mygetpts.dat');
end;
w=pts;
% FILE d093.m ends.
```

Output:

```
Vertices of polygon:
     0     0
     3     0
     2     3
     1     2
     0     1


t =

         0    3.0000    6.1623    7.5765    8.9907    9.9907


td =

  Columns 1 through 7
```

```
         0    0.3333    0.6667    1.0000    1.3333    1.6667    2.0000

  Columns 8 through 14

    2.3333    2.6667    3.0000    3.0000    3.3514    3.7027    4.0541

  Columns 15 through 21

    4.4055    4.7568    5.1082    5.4595    5.8109    6.1623    6.1623

  Columns 22 through 28

    6.3194    6.4765    6.6337    6.7908    6.9480    7.1051    7.2622

  Columns 29 through 35

    7.4194    7.5765    7.5765    7.7336    7.8908    8.0479    8.2050

  Columns 36 through 42

    8.3622    8.5193    8.6764    8.8336    8.9907    8.9907    9.1018

  Columns 43 through 49

    9.2129    9.3240    9.4351    9.5463    9.6574    9.7685    9.8796

  Column 50

    9.9907


xstart =

    0.9487


xstop =

    0.9487


ans =
  Columns 1 through 7

    0.9487         0    3.0000    2.0000    1.0000         0         0

  Column 8

    0.9487


t =

         0    3.0000    6.1623    7.5765    8.9907    9.9907


x =

     0     3     2     1     0     0

polyarea = 7.9146
A/(L*L)=   8.7446e-01
Vertices of polygon:
     0     0
     3     0
     2     3
     1     2
     0     1


t =

         0    3.0000    6.1623    7.5765    8.9907    9.9907


td =

  Columns 1 through 7

         0    0.3333    0.6667    1.0000    1.3333    1.6667    2.0000

  Columns 8 through 14

    2.3333    2.6667    3.0000    3.0000    3.3514    3.7027    4.0541
  Columns 15 through 21

    4.4055    4.7568    5.1082    5.4595    5.8109    6.1623    6.1623

  Columns 22 through 28

    6.3194    6.4765    6.6337    6.7908    6.9480    7.1051    7.2622

  Columns 29 through 35

    7.4194    7.5765    7.5765    7.7336    7.8908    8.0479    8.2050

  Columns 36 through 42

    8.3622    8.5193    8.6764    8.8336    8.9907    8.9907    9.1018

  Columns 43 through 49

    9.2129    9.3240    9.4351    9.5463    9.6574    9.7685    9.8796

  Column 50

    9.9907

polyarea = 5.5
A/(L*L)=   6.9244e-01
```





4. Consider the solution of the nonlinear system $f(x) = 0$ with

$$f_1(x) = x_1 + 3\log|x_1| - x_2^2, \quad f_2(x) = 2x_1^2 - x_1x_2 - 5x_1 + 1\,.$$

(a) Use numjaco.m to check that

$$J_f(x) = \begin{bmatrix} 1+3/x_1 & -2x_2 \\ 4x_1 - x_2 - 5 & -x_1 \end{bmatrix}.$$

Do not mix numjaco with the built-in function numjac!

(b) Show that $x^{(0)} = (2,2)^T$ is not a good starting point for the usual Newton iteration.

(c) Show also that $x^{(0)} = (2,2)^T$ is a good starting point for the damped Newton iteration. In the damped Newton method the correction vector is divided by 2 until $|f(x_{new})| \le |f(x_{old})|$:

```
for j=1:nstep
%   h= myjac(x)\myf(x);      ANALYTIC
   % N.B. myjac takes a column vector x
   h= numjaco('myf',2,x',2)\myf(x);   %NUMERIC
   % N.B. numjaco takes a row vector x
   count=0;   % counts the number of dampings
   while ( (norm(myf(x-h)) >= norm(myf(x))))
     h=0.5*h;
     count=count+1;
   end
   x=x-h;
end
```

Solution: This program uses the files numjaco.m (see d075) and myf.m. First the correctness of the Jacobian is checked with tstjaco and it appears that the analytic Jacobian agrees quite well with the numeric one. Then the program dampedNewton is executed both with damping (isdamp=1) and without damping (isdamp=0). In the first case a fast convergence is observed but not in the second one. In dampedNewton one can choose, by manually commenting/uncommenting the code, either the numeric or analytic Jacobian.

```
% FILE: d094.m begins
function z=d094
% USES: myjac.m,  numjaco.m, myf.m, tstjaco
% WARNING: In the current MATLAB, there is a function with
% the name  numjac. Do not mix it with numjaco !!!
% The code of dampedNewton has two options for computing
% Jacobian:  analytic (myjac) and numeric (numjaco)
path(path,'../d07/')  % numjaco.m will be in the search path

tstjaco;     % Verify
x=[2; 2];
isdamp=1;    % Damping is used if isdamp=1

x=dampedNewton(x,15,isdamp);
print -dps d094a.ps
plotnorm(x);
print -dps d094b.ps
isdamp=1;    % Damping is not used if isdamp=0
x=[2; 2];
x=dampedNewton(x,15,isdamp);
% end of d094

function t=tstjaco()
fprintf('Numerical test for numjaco and myjac:\n')
t=-1;
for j=1:10
  x=5*(rand(1,2)-0.5);
  numJ=numjaco('myf',2,x,2);
  t1=norm(myjac(x)-numJ);
  fprintf('t1=  %12.3e \n',t1)
  t=max([t, t1]);
end
fprintf('Max. error= %12.3e\n',t);
% end of  tstjaco
```

```
function w=myjac(x)
w=[   1+3/x(1) ,   -2*x(2);
     4*x(1)-x(2)-5 ,   -x(1)];
% end of myjac

function z= dampedNewton(x,nstep, isdamp)

fprintf(' n       x(n)          y(n)        damp     norm(f(x(n),y(n))) \n')
res=[];
for j=1:nstep
%   h= myjac(x)\myf(x);
   % N.B. myjac takes a column vector x
   h= numjaco('myf',2,x',2)\myf(x);
   % N.B. numjaco takes a row vector x
   count=0;   % counts the number of dampings
   while ((isdamp==1)&& (norm(myf(x-h)) >= norm(myf(x))))
     h=0.5*h;
     count=count+1;
   end
   x=x-h;
   fprintf('%3d',j)
   fprintf('  % 14.8f', x)
   t=norm(myf(x));
   fprintf('%3d  %12.3e\n', count,t)
   res=[res; x' t]
end
plot(res(:,1), res(:,2),'LineWidth',2)
title('Damped Newton route','FontSize',[20],'FontWeight','bold');
grid on
z=x;
% end of dampedNewton

function w=plotnorm(x)
% x is 2 x 1 vector
xvec=(x(1)-0.2):0.02:(x(1)+0.2);
yvec=(x(2)-0.2):0.02:(x(2)+0.2);
[xx,yy]=meshgrid(xvec,yvec);
myx=xx(:);
myy=yy(:);
m=length(myx);
for j=1:m
  myz(j)=norm(myf([myx(j), myy(j)]));
end
```

```
zz=reshape(myz, size(xx));
figure
axes('FontSize',[20],'FontWeight','bold');
surf(xvec, yvec, zz)
% end of plotnorm
% FILE d094.m ends.
```

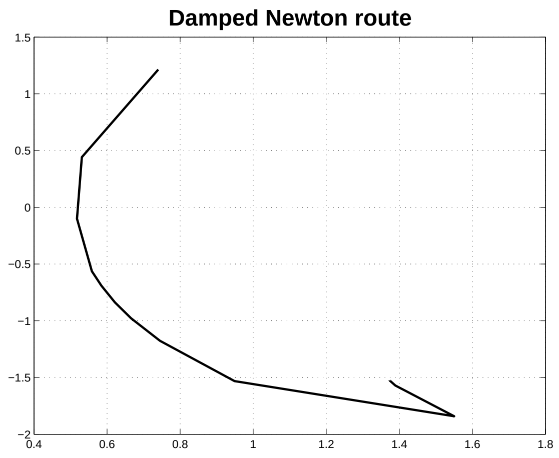
Note that the function myf is stored as in a separate file myf.m.

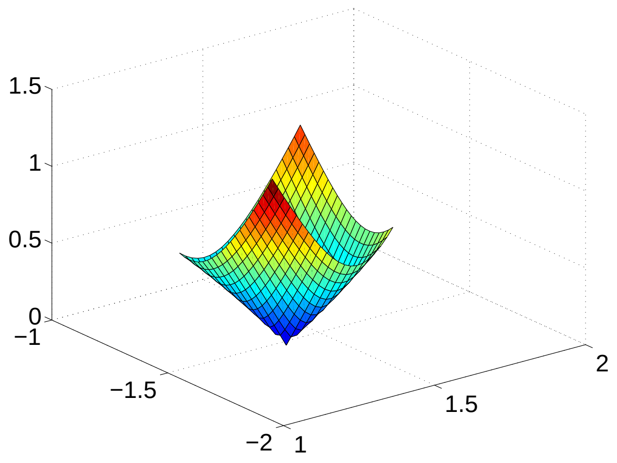
```
function w= myf(x)
w(1,1) =[ x(1)+3*log( abs(x(1)))-x(2)^2];
w(2,1)= [ 2*x(1)^2-x(1)*x(2)-5*x(1)+1];
% FILE: myf.m
```

Output:

```
Numerical test for numjaco and myjac:
t1=     5.986e-09
t1=     6.897e-10
t1=     2.937e-09
t1=     1.901e-08
t1=     2.745e-06
t1=     6.914e-10
t1=     1.787e-07
t1=     1.747e-08
t1=     3.886e-09
t1=     4.534e-09
Max. error=    2.745e-06
 n       x(n)           y(n)        damp     norm(f(x(n),y(n)))
  1      0.74006981      1.21378491  4     2.991e+00

res =

    0.7401    1.2138    2.9906

  2      0.53102378      0.44158542  1     2.049e+00

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
  3      0.51783407     -0.10010967  2     1.776e+00

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756

  4      0.55848382     -0.56378762  3     1.732e+00

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320

  5      0.58470263     -0.69106229  6     1.720e+00

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195

  6      0.62157808     -0.83764454  6     1.713e+00

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126

  7      0.66576127     -0.97725638  6     1.705e+00

res =
    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126
    0.6658   -0.9773    1.7047

  8      0.74487822     -1.17600054  5     1.692e+00

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126
    0.6658   -0.9773    1.7047
    0.7449   -1.1760    1.6915

  9      0.94893941     -1.53131739  3     1.629e+00

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126
    0.6658   -0.9773    1.7047
    0.7449   -1.1760    1.6915
    0.9489   -1.5313    1.6289

 10      1.55016083     -1.84108754  0     1.050e+00

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126
    0.6658   -0.9773    1.7047
    0.7449   -1.1760    1.6915
    0.9489   -1.5313    1.6289
    1.5502   -1.8411    1.0496

 11      1.38921906     -1.57038447  0     1.316e-01

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126
    0.6658   -0.9773    1.7047
    0.7449   -1.1760    1.6915
    0.9489   -1.5313    1.6289
    1.5502   -1.8411    1.0496
    1.3892   -1.5704    0.1316

 12      1.37353855     -1.52574400  0     2.489e-03

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126
    0.6658   -0.9773    1.7047
    0.7449   -1.1760    1.6915
    0.9489   -1.5313    1.6289
    1.5502   -1.8411    1.0496
    1.3892   -1.5704    0.1316
    1.3735   -1.5257    0.0025

 13      1.37347829     -1.52496497  0     6.122e-07

res =
    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126
    0.6658   -0.9773    1.7047
    0.7449   -1.1760    1.6915
    0.9489   -1.5313    1.6289
    1.5502   -1.8411    1.0496
    1.3892   -1.5704    0.1316
    1.3735   -1.5257    0.0025
    1.3735   -1.5250    0.0000

 14      1.37347835     -1.52496484  0     2.176e-14

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126
    0.6658   -0.9773    1.7047
    0.7449   -1.1760    1.6915
    0.9489   -1.5313    1.6289
    1.5502   -1.8411    1.0496
    1.3892   -1.5704    0.1316
    1.3735   -1.5257    0.0025
    1.3735   -1.5250    0.0000
    1.3735   -1.5250    0.0000

 15      1.37347835     -1.52496484  0     4.441e-16

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126
    0.6658   -0.9773    1.7047
    0.7449   -1.1760    1.6915
    0.9489   -1.5313    1.6289
    1.5502   -1.8411    1.0496
    1.3892   -1.5704    0.1316
    1.3735   -1.5257    0.0025
    1.3735   -1.5250    0.0000
    1.3735   -1.5250    0.0000
    1.3735   -1.5250    0.0000

 n       x(n)          y(n)        damp     norm(f(x(n),y(n)))
  1      0.74006981      1.21378491  4     2.991e+00

res =

    0.7401    1.2138    2.9906

  2      0.53102378      0.44158542  1     2.049e+00

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493

  3      0.51783407     -0.10010967  2     1.776e+00

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756

  4      0.55848382     -0.56378762  3     1.732e+00

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320

  5      0.58470263     -0.69106229  6     1.720e+00
```

```
res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195

  6      0.62157808     -0.83764454  6     1.713e+00

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126

  7      0.66576127     -0.97725638  6     1.705e+00

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126
    0.6658   -0.9773    1.7047

  8      0.74487822     -1.17600054  5     1.692e+00

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126
    0.6658   -0.9773    1.7047
    0.7449   -1.1760    1.6915

  9      0.94893941     -1.53131739  3     1.629e+00

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126
    0.6658   -0.9773    1.7047
    0.7449   -1.1760    1.6915
    0.9489   -1.5313    1.6289

 10      1.55016083     -1.84108754  0     1.050e+00

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126
    0.6658   -0.9773    1.7047
    0.7449   -1.1760    1.6915
    0.9489   -1.5313    1.6289
    1.5502   -1.8411    1.0496

 11      1.38921906     -1.57038447  0     1.316e-01

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126
    0.6658   -0.9773    1.7047
    0.7449   -1.1760    1.6915
    0.9489   -1.5313    1.6289
    1.5502   -1.8411    1.0496
    1.3892   -1.5704    0.1316

 12      1.37353855     -1.52574400  0     2.489e-03

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126
    0.6658   -0.9773    1.7047
    0.7449   -1.1760    1.6915
    0.9489   -1.5313    1.6289
    1.5502   -1.8411    1.0496
    1.3892   -1.5704    0.1316
    1.3735   -1.5257    0.0025

 13      1.37347829     -1.52496497  0     6.122e-07

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126
    0.6658   -0.9773    1.7047
    0.7449   -1.1760    1.6915
    0.9489   -1.5313    1.6289
    1.5502   -1.8411    1.0496
    1.3892   -1.5704    0.1316
    1.3735   -1.5257    0.0025
    1.3735   -1.5250    0.0000

 14      1.37347835     -1.52496484  0     2.176e-14

res =
    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126
    0.6658   -0.9773    1.7047
    0.7449   -1.1760    1.6915
    0.9489   -1.5313    1.6289
    1.5502   -1.8411    1.0496
    1.3892   -1.5704    0.1316
    1.3735   -1.5257    0.0025
    1.3735   -1.5250    0.0000
    1.3735   -1.5250    0.0000

  15      1.37347835     -1.52496484  0     4.441e-16

res =

    0.7401    1.2138    2.9906
    0.5310    0.4416    2.0493
    0.5178   -0.1001    1.7756
    0.5585   -0.5638    1.7320
    0.5847   -0.6911    1.7195
    0.6216   -0.8376    1.7126
    0.6658   -0.9773    1.7047
    0.7449   -1.1760    1.6915
    0.9489   -1.5313    1.6289
    1.5502   -1.8411    1.0496
    1.3892   -1.5704    0.1316
    1.3735   -1.5257    0.0025
    1.3735   -1.5250    0.0000
    1.3735   -1.5250    0.0000
    1.3735   -1.5250    0.0000
```





5. Jacobi's method for solving a linear $n \times n$-system $ax = b$ is based on the iteration

$$x_j^{(k+1)} = \left(b_j - \sum_{i=1, i\neq j}^{n} a_{ji} x_i^{(k)}\right) \Big/ a_{jj} \qquad (j = 1, \ldots, n)\,,$$

where the initial value $x^{(0)}$ is chosen arbitrarily. The iteration converges if a is diagonally dominant, i.e. if it is true that $w(a,j) > 0$ for all $j = 1, \ldots, n$, where

$$w(a,j) \equiv |a_{jj}| - \sum_{i=1, i\neq j}^{n} |a_{ji}|\,.$$

Write a MATLAB-program which generates a diagonally dominant linear system, solves it using the Jacobi method, and compares the result to the solution obtained by using the slash-operator $a\backslash b$. How does increasing the absolute values of the diagonal entries effect the accuracy of the results?

Solution:

```
% FILE d095.m begins.
% Jacobi iteration for a linear system a*x = b
%
% a = L+D+U is n by n
% D = diagonal; L = lower diagonal; U = upper diagonal
%      a*x = b  <==>  (L+D+U)*x = b  <==>  D*x= b-(L+U)*x
% Iteration: x_j^{(k+1)} =
% (b_j - \sum{i=1,i \neq j}^n {a_{ji} x_i^{(k)}})/a_{jj}
%                                          (j=1,...,n)
% Converges if |a_{jj}| > \sum{i=1,i \neq j}^n {|a_{ji|},
% for all j=1,...,n.

n=4;
itmax=6;
x = rand(n,1);
a = rand(n,n) + n*eye(n,n); % a is diag. dominant
virhe=[];
for w=1:5
  a = a + eye(n,n);         % Increase the diag. dominance
  b = rand(n,1);
  data = [];
  for k=1:itmax
    x = x + (b - a*x)./diag(a);
    data = [data; x'];
  end;
  disp(['w = ' num2str(w)])
  disp('Jacobi iteration   (d095)');
  fprintf('%10.6f',data(itmax,1:n));
  u = a\b;
  fprintf('\nMATLAB solution = \n%s\n', mat2str(u',5) )
  v = norm(data(itmax,:)' -u);
  virhe = [virhe; v];
  disp(['Error = ' num2str( v )]);
  fprintf('\n');
end;
disp([' w      Error'])
fprintf('%2d  %12.4e\n',[(1:5)' virhe]')
% FILE d095.m ends.
```

Output:

```
w = 1
Jacobi iteration   (d095)
  0.076563  0.009159  0.115161  0.031389
MATLAB solution =
[0.076194 0.0085572 0.11475 0.030862]
Error = 0.0009738

w = 2
Jacobi iteration   (d095)
  0.127989  0.081879  0.043999  0.089311
MATLAB solution =
[0.12799 0.081889 0.044006 0.08932]
Error = 1.6346e-05

w = 3
Jacobi iteration   (d095)
  0.020257  0.032731  0.122726  0.075206
MATLAB solution =
[0.020254 0.032727 0.12272 0.075203]
Error = 5.82e-06

w = 4
Jacobi iteration   (d095)
  0.020918  0.088395  0.076160  0.002605
MATLAB solution =
[0.020917 0.088392 0.076159 0.0026021]
Error = 4.2307e-06

w = 5
Jacobi iteration   (d095)
  0.015965  0.053348  0.065926  0.030196
MATLAB solution =
[0.015965 0.053348 0.065926 0.030196]
Error = 1.394e-07

 w      Error
 1    9.7380e-04
 2    1.6346e-05
 3    5.8200e-06
 4    4.2307e-06
 5    1.3940e-07
```

When w increases, the error seems to decrease.

Solutions in Python

Problem 2.

```
# FILE d092.py begins

from mmeutil import *
from Numeric import *
from LinearAlgebra import *
from scipy import *

def descsort(a):
    # descsort returns the complex  eigenvalues sorted in
    # decreasing absolute order
    b=eigenvalues(a)
    t=argsort(-abs(b))
    v=zeros(len(a))+(1j)*zeros(len(a))
    for i in range(0,len(a)):
        k=t[i]
        v[i]=b[k]
    return v

for p in range(0,10):
    m=int(2+fix(2.0*(0.1+rdm(0,1))))
    a=ranmat(m,m)-0.5
    decreig=descsort(a)
    [u,s,v]=singular_value_decomposition(a)
    for pp in range(0,m):
        t1=abs(prod(decreig[0:pp]))
        t2=prod(s[0:pp])
        print '%12.3e' % (t2-t1)


# FILE d092.py ends
```

Output:

```
0.000e+00
2.807e-03
```

```
   0.000e+00
   2.294e-01
   0.000e+00
   1.531e-01
   0.000e+00
   1.130e-01
   2.369e-02
   0.000e+00
   2.656e-01
   2.399e-01
   0.000e+00
   1.335e-02
   0.000e+00
   1.394e-01
   6.594e-02
   0.000e+00
   7.100e-02
   0.000e+00
   1.092e-01
   5.080e-03
   0.000e+00
   1.566e-01
```

Problem 3.

```
# FILE d093.py begins

from mmeutil import *
from Numeric import *
from scipy import *
from scipy.interpolate import *


def polyarea(x,y):

    z=0
    for i in range(1,len(x)):
        z=z+0.5*(x[i-1]*y[i]-x[i]*y[i-1])
    return z

def myisoper(curtype,x,y):
    # if curtype = 0 make polygonal curve
    # if curtype = 1 make spline curve

    n=len(x)    # Vertices of polygon
    t=0.0*zeros(n)
    t[0]=0.0      # Parameter values: length of polygon
    for i in range(0,n-1):
        t[i+1]=t[i]+pow(pow(x[i]-x[i+1],2.0)+pow(y[i]-y[i+1],2.0),0.5)

    td=0.0*zeros(10*(n-1))
    for i in range(1,n):  # Refined division td
        for j in range(1,11):
            td[10*(i-1)+j-1]=t[i-1]*(9-j+1)/9.0+t[i]*(j-1)/9.0
    if (curtype==1):
        tcx=splrep(t,x,s=0)
        tcy=splrep(t,y,s=0)
        xspl=splev(td,tcx,der=0)
        yspl=splev(td,tcy,der=0)
    else:
        xspl=x
        yspl=y

    gplt.hold('on')
    gplt.plot(x,y,'notitle w p')
    gplt.plot(xspl,yspl,'notitle w l')
    m=len(xspl);
    a2=polyarea(xspl,yspl)
    le=sum(sqrt((xspl[1:m-1]-xspl[2:m])*(xspl[1:m-1]-xspl[2:m])\
      +(yspl[1:m-1]-yspl[2:m])*(yspl[1:m-1]-yspl[2:m])))
    myr=0.0*zeros(2)
    myr[0]=4.0*pi*a2/(le*le)
    myr[1]=a2
    return myr



x=array([0.0,3.0,2.0,1.0,0.0,0.0]) # Polygon must be closed, i.e
y=array([0.0,0.0,3.0,2.0,1.0,0.0]) # x[0]=x[n] and y[0]=y[n]

m1=myisoper(1,x,y)  # 1 yields spline curve
print 'Spline representation:'
print '  Isoperimetric ratio: %8.5f' % m1[0]
print '  Area: %8.5f\n' % m1[1]

m2=myisoper(0,x,y)  # 0 yields polygonal curve
print 'Polygon:'
print '  Isoperimetric ratio: %8.5f' % m2[0]
print '  Area: %8.5f\n' % m2[1]

# end of myisoper

# FILE d093.py ends
```

Output:

```
Spline representation:
  Isoperimetric ratio:  1.02653
  Area:  9.24565

Polygon:
  Isoperimetric ratio:  1.41426
  Area:  5.50000
```

Problem 4.

```
# FILE d094.py begins

from mmeutil import *
from Numeric import *
from scipy import *

def myf(x):
    w=0.0*zeros(2)
    w[0]=x[0]+3.0*log(abs(x[0]))-x[1]*x[1]
    w[1]=2.0*x[0]*x[0]-x[0]*x[1]-5.0*x[0]+1.0
    return w

def tstjaco():
    print 'Numerical test for numjaco and myjac:\n'
    t=-1
    for j in range(0,10):
        x=5.0*(ranvec(2)-0.5)
        numJ=numjaco(myf,2,x,2)
        t1=vnorm(myjac(x)-numJ)
        print 't1=  %12.3e \n' % t1
        t=max(t,t1)
    print 'Max. error= %12.3e\n' % t

def myjac(x):
    w=matrix(2,2)
    w[0,0]=1.0+3.0/x[0]
    w[0,1]=-2.0*x[1]
    w[1,0]=4.0*x[0]-x[1]-5.0
    w[1,1]=-x[0]
    return w

def dampedNewton(x,nstep,isdamp):
    print ' n       x(n)          y(n)         damp     norm(f(x(n),y(n)))'
    res=matrix(nstep,3)
    for j in range(0,nstep):
         h=LUsolve(numjaco(myf,2,x,2),myf(x))
         count=0   # counts the number of dampings
         while ((isdamp==1) and (vnorm(myf(x-h))>=vnorm(myf(x)))):
             h=0.5*h
             count=count+1
         x=x-h
         t=vnorm(myf(x))
         print '%3d %14.8f %14.8f % -6d %12.3e' % (j,x[0],x[1],count,t)
         res[j,0]=x[0]
         res[j,1]=x[1]
         res[j,2]=t
    gplt.plot(res[:,0], res[:,1],'notitle w l')
    gplt.output('d094_'+str(isdamp)+'.png','png color')
    return x

tstjaco     # Verify
x=array([2.0,2.0])
isdamp=1    # Damping is used if isdamp=1
x=dampedNewton(x,15,isdamp);
isdamp=0;   # Damping is not used if isdamp=0
x=array([2.0,2.0])
x=dampedNewton(x,15,isdamp)
```

```
# FILE d094.py ends
```

Output:

```
 n       x(n)          y(n)        damp     norm(f(x(n),y(n)))
  0     0.74007012     1.21378506  4        2.991e+00
  1     0.53102393     0.44158553  1        2.049e+00
  2     0.51783419    -0.10010948  2        1.776e+00
  3     0.55848385    -0.56378691  3        1.732e+00
  4     0.58470250    -0.69106100  6        1.720e+00
  5     0.62157768    -0.83764249  6        1.713e+00
  6     0.66576065    -0.97725408  6        1.705e+00
  7     0.74487759    -1.17599902  5        1.692e+00
  8     0.94893930    -1.53131797  3        1.629e+00
  9     1.55016069    -1.84108753  0        1.050e+00
 10     1.38921904    -1.57038443  0        1.316e-01
 11     1.37353855    -1.52574400  0        2.489e-03
 12     1.37347829    -1.52496497  0        6.122e-07
 13     1.37347835    -1.52496484  0        4.575e-14
 14     1.37347835    -1.52496484  0        4.441e-16
 n       x(n)          y(n)        damp     norm(f(x(n),y(n)))
  0   -18.15887804   -10.57943902  0        5.722e+02
  1    -8.37100537    -5.22872545  0        1.423e+02
  2    -3.55249051    -2.71907014  0        3.508e+01
  3    -1.20145765    -1.47282629  0        8.600e+00
  4    -0.00041250     0.09490574  0        2.341e+01
  5     0.11918475  -951.59339436  0        9.055e+05
  6     0.05885162  -475.79257720  0        2.264e+05
  7     0.02761647  -237.88571363  0        5.660e+04
  8     0.00981580  -118.91617854  0        1.415e+04
  9    -0.00364560   -59.38239118  0        3.543e+03
 10    -0.02038521   -29.66824349  0        8.919e+02
 11    -0.05303452   -14.71744550  0        2.255e+02
 12    -0.14636722    -7.23381541  0        5.824e+01
 13    -0.85350592    -4.16126788  0        1.891e+01
 14     0.41936173    -1.53634440  0        4.549e+00
```

Problem 5.

```
# FILE d095.py begins
```

```
from mmeutil import *
from Numeric import *
from scipy import *

# Jacobi iteration for a linear system a*x = b
#
# a = L+D+U is n by n
# D = diagonal; L = lower diagonal; U = upper diagonal
#      a*x = b  <==>  (L+D+U)*x = b  <==>  D*x= b-(L+U)*x
# Iteration: x_j^{(k+1)} =
# (b_j - \sum{i=1,i \neq j}^n {a_{ji} x_i^{(k)}})/a_{jj}
#                                             (j=1,...,n)
# Converges if |a_{jj}| > \sum{i=1,i \neq j}^n {|a_{ji|},
# for all j=1,...,n.

n=4
itmax=6
x=ranvec(n)
a=ranmat(n,n)+n*eye(n,n) # a is diag. dominant
virhe=1.0*zeros(5)
for w in range(1,6):
    a=a+eye(n,n)         # Increase the diag. dominance
    b=ranvec(n)
    data=matrix(itmax,n);
    for k in range(0,itmax):
        x=x+(b-matvecmul(a,x))/diag(a)
        data[k,:]=x
    print 'w = %d' % w
    print 'Jacobi iteration'
    showvec(data[itmax-1,:])
    u=LUsolve(a,b)
    print 'Solution = '
    showvec(u)
    v=vnorm(data[itmax-1,:]-u)
    virhe[w-1]=v
    print 'Error = %10.7e\n' % v
print ' w      Error\n'
for i in range(1,6):
    print '%2d  %12.4e' % (i,virhe[i-1])

# FILE d095.py ends
```

Output:

```
w = 1
Jacobi iteration
   0.065     0.001     0.114     0.011
Solution =
   0.065     0.001     0.114     0.011
Error = 5.8469667e-05

w = 2
Jacobi iteration
   0.111    -0.022     0.084     0.036
Solution =
   0.111    -0.022     0.084     0.036
Error = 1.0150329e-07

w = 3
Jacobi iteration
   0.061     0.043     0.095     0.067
Solution =
   0.061     0.043     0.095     0.067
Error = 3.8676127e-07

w = 4
Jacobi iteration
   0.019     0.035     0.009     0.054
Solution =
   0.019     0.035     0.009     0.054
Error = 6.1910664e-07

w = 5
Jacobi iteration
   0.019     0.056     0.046     0.052
Solution =
   0.019     0.056     0.046     0.052
Error = 1.2354850e-07

 w      Error

 1    5.8470e-05
 2    1.0150e-07
```

```
3     3.8676e-07
4     6.1911e-07
5     1.2355e-07
```