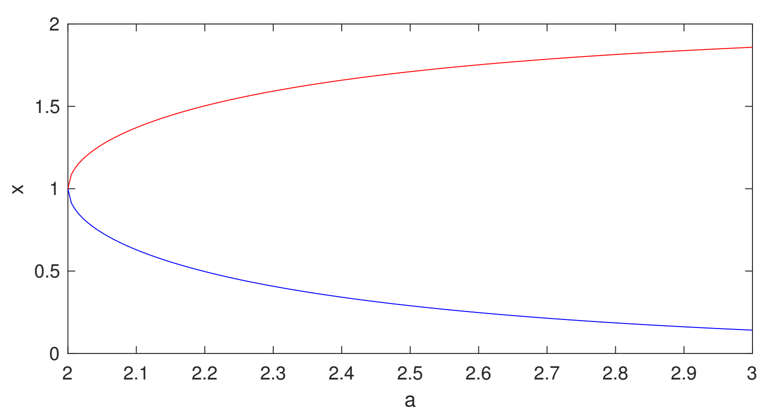
MA2507 Computing Mathematics Laboratory: Week 8

1. Anonymous function. This is a quick method to define short and simple functions. Consider the function $f(x, a) = xe^{a(1-x)} + x - 2$. You can verify that $f(1, a) = 0$. For $a > 2$, we have two more solutions $x = p_1(a)$ and $x = p_2(a)$ satisfying $p_1 + p_2 = 2$. In the following program, we solve p_1 and p_2 for $2 \le a \le 3$, and show p_1 and p_2 as functions of a. To solve p_1, we use MATLAB internal program `fzero`. It requires a single-variable function and an initial guess.

```
n=200;
a = linspace(3,2,n);     % start at a=3, go down to a=2
x0 = 2;                  % initial guess for a=3
for j=1:n
    f = @(x) x*exp(a(j)*(1-x))+x-2;
    p(j) = fzero(f, x0);
    x0 = p(j);           % use p(j) as initial guess for next a
end
% plot p and (2-p) as functions of a
plot(a, p, 'r', a, 2-p, 'b')
xlabel('a'), ylabel('x')
```

At $a = 2$, the two solutions merge to $x = 1$, thus it is easier to solve p_1 for $a = 3$. Therefore, we vary a from 3 to 2. Once p_1 is solved for a_j, we use the solution as the initial guess to solve the equation for a_{j+1}. The program gives the following figure.



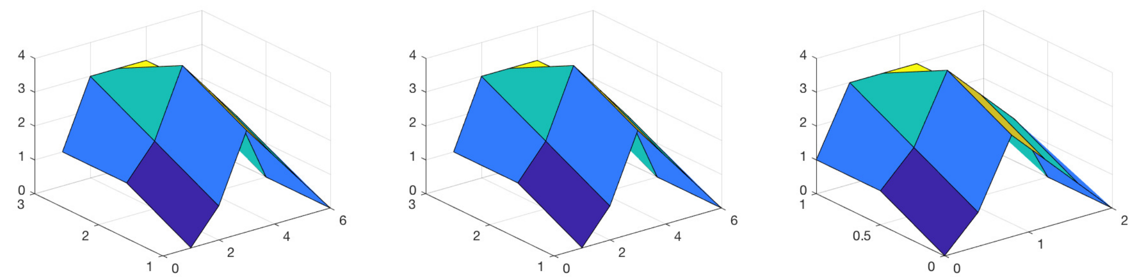
2. 2D plots. MATLAB has a few different commands for plotting 2D figures. First, we take a look at `surf`. Consider the following program

```
f = [0 1 3 2 1 0; 1 2 4 3 2 0; 1 3 3 3 2 1]
subplot(1,3,1), surf(f)
jj = 1:6;
ii = 1:3;
subplot(1,3,2), surf(jj,ii,f)
x = linspace(0,2,6);
y = linspace(0,1,3);
subplot(1,3,3), surf(x,y,f)
```

It gives the matrix `f`:

```
f =
     0     1     3     2     1     0
     1     2     4     3     2     0
     1     3     3     3     2     1
```

and the figure below. The first plot uses only the matrix `f` as the input, the two horizontal axes



are just integers from 1 to 6 (shown as from 0 to 6), and from 1 to 3. In our usual coordinate system, the x direction corresponds to integer vector $[1, 2, ..., 6]$, and the y direction corresponds to integer vector $[1, 2, 3]$. As a test, I define two integer vectors `jj` and `ii` and plot the figure again, using `jj` and `ii` also as inputs. This gives the second plot which is identical to the first one. Usually, we think the matrix `f` is obtained from a function $f(x, y)$ for some discrete values of x and y. Let us assume x is from $[0, 2]$ and y is from $[0, 1]$, then I sample x by 6 points and sample y by 3 points, and now use x and y as inputs. This gives the third plot. It is still the same figure. Since the first two plots actually show a horizontal axis from 0 to 6 (instead of from 1 to 6), we see a slight difference betwen the third and the first two plots. Importanly, the (i, j) entry of `f` is NOT $f(x_i, y_j)$, it is $f(x_j, y_i)$. Namely,

$$f_{ij} = f(x_j, y_i)$$

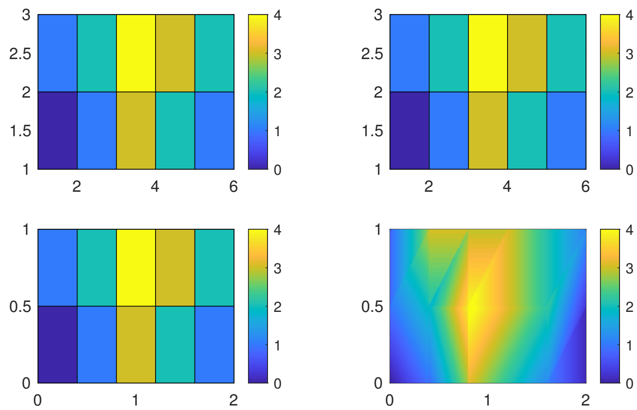
We have six x_j (for $1 \le j \le 6$) and three y_i (for $1 \le i \le 3$), the above gives a 3×6 matrix which is exactly right. If you set $\mathtt{f}_{ij} = f(x_i, y_j)$, you will get a 6×3 matrix. Finally, we see that `surf` gives continuous and piecewise plane segments. That means `surf` interpolates the points linearly.

A more useful command is `pcolor`. Let us look at the following program.

```
f = [0 1 3 2 1 0; 1 2 4 3 2 0; 1 3 3 3 2 1];
subplot(2,2,1)
pcolor(f), colorbar
jj = 1:6;
ii = 1:3;
subplot(2,2,2)
pcolor(jj,ii,f), colorbar
x = linspace(0,2,6);
y = linspace(0,1,3);
subplot(2,2,3)
pcolor(x,y,f), colorbar
subplot(2,2,4)
```

```
pcolor(x,y,f), colorbar
shading interp
```

The program produces the following figures. The first two are identical. That means if we only



input the matrix `f`, MATLAB uses interger vectors as the x and y coordinates. In the 3rd and 4th plots, we force MATLAB to use our own x and y vectors. The first three plots show piecewise constant figures. There is no interpolation. More importantly, notice that only two rows and five columns are shown (instead of 3 rows and 6 columns). What we see in the first three plots are 2×5 elements of `f` arranged as

$$\begin{array}{ccccc} f_{21} & f_{22} & f_{23} & f_{24} & f_{25} \\ f_{11} & f_{12} & f_{13} & f_{14} & f_{15} \end{array}$$

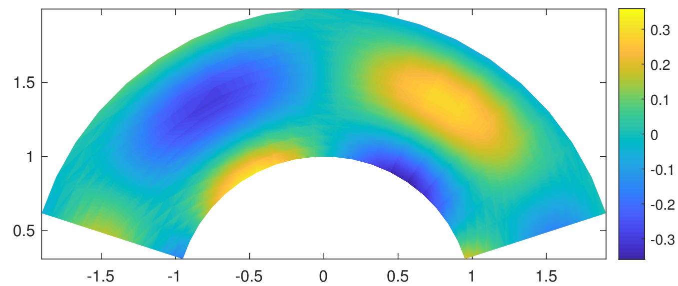
The first row is at the bottom. That is good, because we usually want to have y increase in the vertical direction. The last row and last column are missing. You can get the last row, last column back by `shading interp`. It also gives you linear interpolation, so that the figure is now continuous. This is what we see in the 4th plot.

Another way of using `pcolor` is to input three matrices X, Y and Z. This is useful, if the area we are plotting is not a rectangle in the xy plane. The following example plots a function for $r \in [1, 2]$ and $\theta \in [0.1\pi, 0.9\pi]$, where r and θ are polar cooridnates. The function is $J_3(5r)\cos(3\theta)$, where $J_m(z)$ is a Bessel function.

```
m = 15;
r = linspace(1,2,m);
n = 20;
t = pi*linspace(0.1,0.9,n);
for j=1:n
    for i=1:m
        Z(j,i) = myfn(r(i),t(j));
        X(j,i) = r(i)*cos(t(j));
        Y(j,i) = r(i)*sin(t(j));
    end
end
pcolor(X,Y,Z), colorbar
shading interp
```

```
axis equal tight

function f = myfn(r,t)
f = besselj(3,5*r)*cos(3*t);
end
```

The program gives the following figure.



3. Finding minima. To find a minimum of a function, we can use `fminbnd`. For example, the following lines

```
>> f =@(x) x^4+x^2 - 3*sin(x);
>> format long
>> fminbnd(f,0,2)
ans =
   0.648846918337408
```

find a minimum of $f(x) = x^4 + x^2 - 3\sin(x)$ in the interval $[0, 2]$. It gives you the position of x where the minimum is achieved. If you want to know the minimum value of f, just put that x into the function f. The first input of `fminbnd` is the name of a function. Since the first line above defines the function f, you do not need to add the symbol `@`. But for the following script file

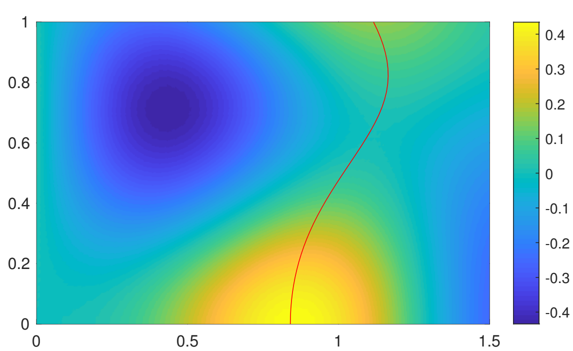
```
% main script
format long
fminbnd(@f,0,2)
%
% function defined after the main script
function y=f(x)
y = x^4+x^2 - 3*sin(x);
end
```

the function f is defined below the main script. You need to add `@` in front of f. Otherwise, MATLAB will give you an error message.

Now for the function $f(x, y) = J_3(5r)\cos(3\theta)$, we first plot it in the rectangle $[0, 1.5] \times [0, 1]$, then find the maximum of f, as a function of x only, for each fixed $y \in [0, 1]$.

```
% plot f(x,y) on [0,1.5]x[0,1]
n = 60;
x = linspace(0,1.5,n);
m = 40;
y = linspace(0,1,m);
for i=1:m
    for j=1:n
        F(i,j) = myfn(x(j),y(i));
    end
end
pcolor(x,y,F), shading interp, colorbar, axis equal tight
%
% find max of f(x,y) for each fixed y
for i=1:m
    g =@(x) -myfn(x,y(i));     % multiple -1, max becomes min
    xx(i) = fminbnd(g,0,1.5);
end
hold on
plot(xx,y,'r')
hold off

% function J_3(5r)cos(3theta)
function f = myfn(x,y)
z = x+1i*y;
r = abs(z);
t = angle(z);
f = besselj(3,5*r)*cos(3*t);
end
```

Here, we use a loop for different y, define a function of x for each y, then use `fminbnd`. The positions of the maxima (for each y) are plotted as a curve on top of the figure for f. The program gives the following figure.



4. SVD and image compression. An $m \times n$ matrix F may have many small singular values. For a real matrix, the so-called singular value decomposition (SVD) is the relation $F = USV^{\mathsf{T}}$, where

U and V are two orthogonal matrices (an orthogonal matrix is one whose inverse is simply the transpose), S is a diagonal matrix with the singular values on the diagonal, that is

$$S = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \ddots \end{bmatrix},$$

where $\sigma_1 \geq \sigma_2 \geq ... \geq 0$. If there are many small singular values, we can choose an integer k, and replace σ_j by 0 for $j > k$. This means that F is approximated by

$$F_k = \sum_{i=1}^{k} \sigma_i u_i v_i^{\mathsf{T}},$$

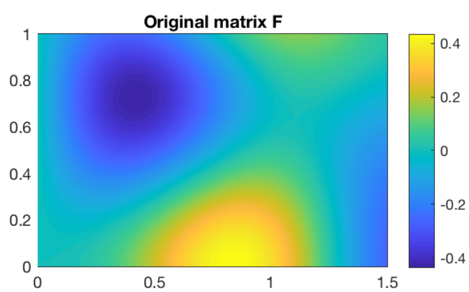
where u_1, u_2, ..., are the columns of U, v_1, v_2, ..., are the columns of V. The following program constructs a matrix F for the function $J_3(5r)\cos(3\theta)$ on $[0, 1.5] \times [0, 1]$, calculates its SVD, approximates F by F_3 (keeping only three non-zero singular values), shows the first 5 singular values, and compares the figures for F and F_3.

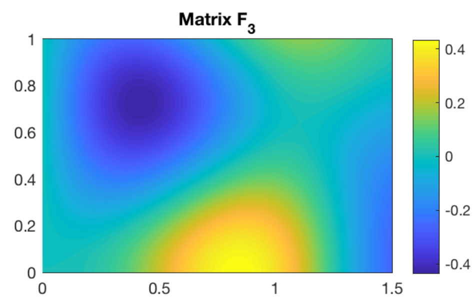
```
% main script
m = 100;
y = linspace(0,1,m);
n = 150;
x = linspace(0,1.5,n);
for i=1:m
    for j=1:n
        F(i,j) = myfn(x(j),y(i));
    end
end
subplot(1,2,1)
pcolor(x,y,F), title('Original matrix F')
shading interp, colorbar, axis equal tight
%
[U,S,V] = svd(F);
S(1:5,1:5)
F3 = U(:,1:3)*S(1:3,1:3)*(V(:,1:3))';
subplot(1,2,2)
pcolor(x,y,F3), title('Matrix F_3')
shading interp, colorbar, axis equal tight

% a function below the main script
function f = myfn(x,y)
z = x+1i*y;
r = abs(z);
t = angle(z);
f = besselj(3,5*r)*cos(3*t);
end
```

The first five singular values are shown by the line `S(1:5,1:5)`.

```
ans =
   19.8646         0         0         0         0
         0   13.4750         0         0         0
         0         0    3.1757         0         0
         0         0         0    0.0037         0
         0         0         0         0    0.0001
```

The program produces the following two figures. It is hard to see any difference between the





two. For F_k, it is only necessary to store k columns of U, k columns of V, and k singular values, Therefore, if F represents an image, we have a way to compress it. That is useful for storage and transmission of images.