Unit 2

September 29, 2015

1 External function routines

We have already seen that Matlab has built-in functions, such as

- `sin()`
- `asin()`
- `sinh()`
- `acos()`
- `cos()`
- `cosh()`
- `tan()`
- `atan()`
- `tanh()`
- `exp()`
- `log()`
- `log10()`
- `sqrt()`
- `abs()`
- `sign()`
- `factorial()`
- ...

which are ready to use. As explained in the following, there are also other ways to create your-own function in Matlab.

2 Anonymous functions

An anonymous function can be defined at any point in a script and it is a way of defining a mathematical function rather than a block of operations. For example

```
f=@(x)(x.^3-exp(-0.5*x).*x)
x=linspace(-2,2,100);
plot(x, f(x))
```

Note that you can have functions with more than one argument, such in the following example

```
g=@(x,a)(x.^3-exp(-0.5*x).*x+a*x.^3)
x=linspace(-3,3,100);
plot(x, g(x,0.1))
```

Do not get confused between **indices** and **arguments of a function**, i.e., put great attention about what you put in parenthesis after a vector or a function. If we define

```
x=linspace(0,1,10);
f=2*x-x.^2
```

then f is a vector whose values `f(1)`, `f(2)`, ... can be accessed only for **integer values of the indices**, i.e. $f(i)$, where the index `i` can only be a natural number `i=1, 2, ..., 10`. Clearly, if you type `f(0.5)`, it will give you an error message:

```
??? Attempted to access f(0.5); index must be a positive
integer or logical
```

In this case, if we want to plot the function `f`, we use the command `plot(x,f)`. If we instead define an anonymous function via the command `f=@(x)(...)`, i.e.,

```
x=linspace(0,1,10);
f=@(x)(2*x-x.^2)
```

then `f(x)` is an anonymous function that can be used to evaluate `f` at any given **real** value of `x`, exactly as you would do with matlab built-in functions. Here, if we want to plot the function, we will use the command `plot(x,f(x))`.

2.0.1 Exercise:

Write a script which plots $x(t) = vt + A\cos(\omega t)$ and $y(t) = A\sin(\omega t)$ as a function of $t \in [0, 10\pi]$ by making use of anonymous functions. Generate 4 subplots for different values of the coefficients A (amplitude) and v: in two subplots fix A and change v and in the other two subplots fix v and change A. Generate a second figure with 4 subplots where you plot the trajectories $y(x)$ using the same parameters values of the previous subplots. N.B. Remember you already solved this problem in Unit 1 so already have the script for it, to which you only have to make the necessary (small) changes.

3 Script functions

Script functions require that you write a separate file.m with the name of the file being the same as the one of the function. The important advantage of script functions is that they can contain a block of certain operations that you plan to do repeatedly in another script, or later in time in a different script — i.e., you can create your own script functions for differentiating at a given order, integrate at a given order, or any other operation that we will see in the following units, such as evaluating zeros of functions with a certain precision and with a given method, solving differential equations, and so on.

Let's consider the following example which evaluates the first derivative of a given function. We will see other examples of script functions all along the course.

```
clear all
close all
clc
% script to evaluate the first derivative of a function
% in a given interval
% and compare the numerical result with the exact one
a=0; b=1; N=20;
fun=@(x)(x.^2+5*x.^5);
df_exact=@(x)(2*x+25*x.^4)

% call the external script function diff_first.m
% a, b, N, fun are INPUTs given above
% x, fp, deltax, il, ir are OUTPUTs
[x, fp, deltax, il, ir]=diff_first(a, b, N, fun);

hold on
plot(x,fun(x),'b-','Linewidth', 2)
plot(x(il),fp,'r-s')
plot(x(ir),fp,'g-s')
plot(x(il)+deltax/2,fp,'ks')
plot(x,df_exact(x),'k--')
legend('function', 'approx 1', 'approx 2', 'approx 3',
'exact','Location','NorthWest')
hold off
```

In the same directory where you run this script, you have to create the following file named diff_first.m:

```
%--------------------------------------------%
% diff_first():  evaluates the derivative of f %
%--------------------------------------------%
% INPUTs:  to be provided
% a --- scalar:  lower bound of the interval
% b --- scalar:  upper bound of the interval
% N --- number of point in the grid
% fun --- anonymous function

function [x, fp, deltax, il, ir] = diff_first(a, b, N, fun)
x=linspace(a ,b, N);
f=fun(x);
il=(1:1:N-1); ir=(2:1:N);
deltax=x(ir)-x(il);
deltaf=f(ir)-f(il);
fp=deltaf./deltax;
end
```

Thus, summarising, to create a function,

$$[\mathtt{a}, \mathtt{b}, \mathtt{c}] = \mathtt{your\_function}(\mathtt{x}, \mathtt{y}, \mathtt{z}, \mathtt{t})$$

you need to write a separate script file and name it your_function.m. In this file, `x, y, z, t` are INPUT variables, while `a, b, c` are OUTPUT variables, i.e., what the function returns. To use this function in a script, you then call the function by `[a, b, c] = your_function(x,y,z,t)`. Note that the names of the variables inside the function do not need to be the same as the names of the variables you pass to the function! Also note that before you can be able to use an external function, you must set the path to tell Matlab where that function can be found (we will see this in class).

4 Some old exercises redone with the use of external function routines

4.0.2 Exercise:

By making use of an external function routine, plot the value of the numerical derivative of the function $f(x) = x^2 + 5x$ at $x_0 = 2$ for different values of the increment δx with first and second order approximation schemes; establish how fast the numerical value converges to the exact one, $f'(2) = 9$ and comment the result you get.

4.0.3 Exercise:

By writing an external function routine for the integral (in different approximations schemes), consider the following function

$$f(x) = \int_0^x ds e^{-s^2} \cos(3s) ,$$

evaluate it and plot it in the interval $x \in [0, 2\pi]$ — remember the use of the command `cumsum()`.

4.0.4 Exercise:

By writing an external function routine for the integral (choose one approximation scheme, for example the trapezoidal one), consider the following function

$$f(x) = \int_0^x ds \frac{\sin(s)}{s} ,$$

evaluate it and plot it in the following interval $x \in [0, 6\pi]$. By choosing larger and larger intervals, can you guess the value of the integral $\int_0^\infty ds \frac{\sin(s)}{s} (= \frac{\pi}{2})$?

4.0.5 Exercise

The equation of motion for the one-dimensional harmonic oscillator can be derived from the Newton's law $F = ma = m\frac{d^2x}{dt^2}$ and the Hooke's law, $F = -kx$, describing the motion of an object of mass m which, displaced from its equilibrium position by a distance x, experiences a restoring force proportional to the displacement:

$$m\frac{d^2x(t)}{dt^2} + kx(t) = 0 \; . \tag{1}$$

This differential equation, with initial conditions $x(t = 0) = x_0$ and $v(t = 0) = v_0$, where $v(t) = \frac{dx(t)}{dt}$ is the velocity, admits the exact solution

$$x(t) = \sqrt{x_0^2 + \frac{v_0^2}{\omega_0^2}} \cos(\omega_0 t + \phi) \; , \tag{2}$$

where $\omega_0 = \sqrt{\frac{k}{m}}$ and $\phi = \arccos \frac{x_0}{\sqrt{x_0^2 + v_0^2/\omega_0^2}}$.

1. Create an external function `[x] = harm_oscill(k, m, x0, v0, t)` that evaluates the position of the harmonic oscillator on the time interval specified by the input time vector `t`.
2. Plot the position $x(t)$ in the interval of time $t \in [0, 4\pi/\omega_0]$ (label the plot axis with the correct units) with the initial conditions $x_0 = 3.2$ m and $v_0 = -2$ m/s, and for the following values of the system parameters: mass $m = 2$ kg and spring constant $k = 4$ kg/s^2;
3. by writing an external function routine for differentiating, evaluate numerically the velocity $v(t)$ and plot the numerical result together with the analytical one;
4. by using the same external function routine generated previously, evaluate numerically the acceleration $a(t)$ and plot the numerical result together with the analytical one.

5 Loops and conditions: Loop `for`

The loop `for` allows to repeat certain commands a specified number of times and it is useful when one wants to repeat a certain action in a predetermined way. All loop structures are started with a keyword, in this case `for`, and end with the word `end`. After the `for` command a loop vector is given and Matlab will loop through for each value of the vector. For example

```
for j=1:10
    j
end
```

In this simple example the loop goes around 10 times, each time changing the value of the variable `j` and printing it. Compare the example above with the following one and explain the difference:

```
for j=1:10
    x(j)=j
end x
```

Note that in certain simple cases Matlab provides short-cuts to the loop `for`, i.e., Matlab does the loop on its own automatically; for example, imagine to have defined a certain vector `x = linspace(0,10,21)` and you want to store in another vector `f` of the same length the values of $\sqrt{x}$, then you just have to type `f = sqrt(x)`. This means that for this simple case this operation is equivalent to

```
for i=1:21
    f2(i)=sqrt(x(i));
end
```

Try it and compare the two vectors `f` and `f2`. Clearly there are cases where Matlab does not provide this short-cut and you have no other ways than to use the loop `for`.

5.0.6 Exercise:

Define the following vector `y`

```
x=linspace(0,1,5);
for i=1:5
    y(i)=x(i)^i;
end
```

Now define an identical vector `z` without using the loop `for`.

5.0.7 Exercise:

Define the following matrix

$$M = \begin{pmatrix} 1 & 9 & 2 & 23 \\ 2 & 8 & 13 & 2 \\ -2 & 9 & 7 & 1 \\ 45 & 2 & 82 & -11 \end{pmatrix}$$

and, by making use of a loop `for`, redefine a new 4×4 matrix, `M2`, where, starting from the second row its elements are obtained by subtracting the previous row (and the first row is left unchanged), i.e.:

$$M2 = \begin{pmatrix} 1 & 9 & 2 & 23 \\ 1 & -1 & 11 & -21 \\ -4 & 1 & -6 & -1 \\ 47 & -7 & 75 & -12 \end{pmatrix}$$

5.0.8 Exercise:

After having defined the vector `x=linspace(0,2,20)`, use the loop `for` in order to find an equivalent way of defining the vector `f=x.*x`.

5.0.9 Exercise:

Write a script that, by making use of the loop `for`

1. generates five different figures: `sin(1*x)` in `figure(1)`, `sin(2*x)` in `figure(2)`, and so on;
2. repeat the exercise of “Achilles and the Tortoise” of *Unit 1 — part 2* (Ex. 2.0.4); in particular use the loop `for` in order to find an estimate of time and distance at which Achilles finally meets the Tortoise; also use the loop `for` in order to generate some of the various vertical and horizontal segments of Fig.1 of that exercise.

5.0.10 Exercise:

Write a script that evaluates the factorial `n!` of a given natural number `n` and compare the results with the built-in function `factorial(n)` — remember that the factorial is defined as $n! = n(n-1)(n-2)\dots 2*1$; Hint: store the result in a variable `f` that needs to be initialised to `f=1` prior to the loop.

5.0.11 Exercise:

Given two matrices `A` and `B`, we have seen in the previous unit the two possible multiplication operations we can use:

$\texttt{C} = \texttt{A} * \texttt{B}$	mathematical multiplication (row × column)
$\texttt{D} = \texttt{A}.*\texttt{B}'$	component-wise multiplication

In particular, define the following two matrices

$$A = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix} \qquad\qquad B = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix};.$$

1. Use the loop `for` rather than explicitly the `*` operation in order to evaluate the matrix `C = A * B` and compare the two results — remember the formula for the mathematical multiplication of matrices $C_{ij} = \sum_k A_{ik}B_{kj}$;
2. use the loop `for` rather than the `.*` operation in order to evaluate the matrix `D = A .* B'` and compare the two results — remember that now you are multiplying component-wise, thus `A` and `B'` have to have the same dimension, in this case 3×2.

5.0.12 Exercise:

Let us consider the geometric series

$$a_{exact} = \sum_{n=0}^{\infty} x^n = \frac{1}{1-x} \qquad \text{for } |x| < 1 \ . \tag{3}$$

1. Choose a numerical value for x (say $x = 1/2$) and plot the sum of the first N terms of the geometric series, i.e., it's approximated numerical value

   $$a(N) = \sum_{n=0}^{N} x^n \ , \tag{4}$$

   as a function of N and check that it coverges to $1/(1-x)$. Compare the values you get for a fixed N with the exact analytical expression $a(N) = \frac{1-x^{N+1}}{1-x}$.

2. Plot now $a(N)$ as a function of N^{-1}.

5.0.13 Exercise:

By making use of the loop `for N=10:100`, evaluate the numerical integral of $f(x) = e^x$ between $[a, b] = [0, 1]$ for different values of the number of points on the grid N. Observe how the result converges to the exact one by plotting the numerical integral first as a function of N and then as a function of $1/N$.

6 Loops and conditions: Command `if`

When one needs a statement to be executed only in limited circumstances, the command to use is `if`. This command executes a statement if the stated condition is met — as usual, you can familiarise yourself with this command by typing `help if` and `doc if`. For example you can compare two numbers with the following script:

```
x=input('type x');
y=input('type y');

if x>y
    disp('x greater than y')
elseif x<y
    disp('x smaller than y')
else
    disp('x equal to y')
end
```

The conditions are comparisons which include:

- $<$ (less than)
- $>$ (greater than)

- <= (less or equal than)
- >= (greater or equal than)
- == (equal)
- ~= (not equal).

There are two possible formulations of the `if` command:

```
% case 1:  if-then-formulation
if condition
    command
end
```

```
% case 2:  if-then-else-formulation
if condition
    command
else
    some other command
end
```

6.0.14 Exercise: the half-wave rectifier

Consider the function $f(x) = \sin(x)$ in the interval $x \in [-2\pi, 2\pi]$ and redefine a new function $g(x)$ which is zero in the interval where $f(x)$ is negative. Plot both functions.

7 Loops and conditions: Loop `while`

Another command that executes loops is the `while ...end` construct. This command will repeat the command which is contained in between `while ...end` an indefinite number of times until some logical condition is satisfied. You can find more information about it by typing either `help while` or `doc while`. For example we can define a vector n with ten components from 1 to 10 by using the command `while`:

```
i=1;
while i<=10,
      n(i)=i;
      i=i+1;
end
n
```

In many cases using the loop `while` can make for a more efficient algorithm than using the loop `for`, even though the particular example above is not the case. We will see many examples later on, when numerically solving differential equations.

As for the case of the condition `if`, also for the loop `while` you can use the following logical conditions:

- < (less than)

- $>$ (greater than)
- $<=$ (less or equal than)
- $>=$ (greater or equal than)
- $==$ (equal)
- $\sim=$ (not equal).

Let's consider the following simple example

```
f=64;
n=0;
while f>1
      f=f/2
      n=n+1;
end
n
```

This is a way of determining how often a number can be divided by 2, and you can check that `2^n` gives you the initial number `64` — of course there is a better and faster way to do the same, which is evaluating `log2(64)`!

7.0.15 Exercise:

Use the loop `while` to generate a routine which is able to establish whether a natural number n is a prime number (i.e., a natural number which can only be divided by 1 and itself) or not. Rewrite the same routine with the loop `for`.

Hints to solve Exercise 7.0.15

- You can use the command `rem(n,div)` (or equivalently the command `mod(n,div)`) to evaluate the rest of a division; if an integer number `div` exists such that `mod(n,div)=0`, then clearly it means that `n` cannot be a prime number;
- remember the use of `if` (see previous section);
- you can terminate a loop like `while` and `for` with the command `break` (e.g., once the conditions you are looking for are met, without the need to leave the computer run till the natural end the loop; in this way you shorten the evaluation time);
- once you have established the conditions you are looking for, you can print "n is a prime number" (or "n is not a prime number") with the command `disp('n is a prime number')`.

8 Applications: Vector rotations

Let us consider a two-dimensional vector $\mathbf{r}_1 = (x1, y1)$, whose components in polar coordinates can be written as $x_1 = r\cos\phi_1$ and $y_1 = r\sin\phi_1$. And now

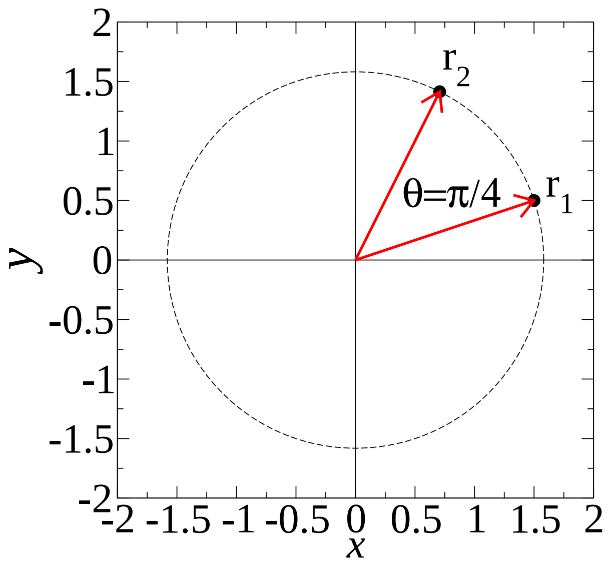

Figure 1: Rotating anti-clockwise the vector $\mathbf{r}_1 = (x_1, y_1) = (1.5, 0.5)$ by an angle $\theta = \pi/4$.

let us suppose we want to rotate $\mathbf{r}_1$ anti-clockwise by an angle θ (see Fig. 1) so that to get a new vector $\mathbf{r}_2 = (x2, y2) = (r\cos\phi_2, r\sin\phi_2)$, where $\phi_2 = \phi_1 + \theta$. Thus the new component of the rotated vector can be written as

$$\begin{aligned} x_2 &= r\cos(\phi_1+\theta) = r(\cos\phi_1\cos\theta - \sin\phi_1\sin\theta) = x_1\cos\theta - y_1\sin\theta \\ y_2 &= r\sin(\phi_1+\theta) = r(\sin\phi_1\cos\theta + \cos\phi_1\sin\theta) = y_1\cos\theta + x_1\sin\theta\ . \end{aligned}$$

Thus, the rotation can be written as the following matrix operation acting on the vector $\mathbf{r}_1$:

$$\mathbf{r}_2 = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \mathbf{r}_1\ . \tag{5}$$

You can easily show that both vectors $\mathbf{r}_1$ and $\mathbf{r}_2$ have the same norm, i.e., length: $|\mathbf{r}_1| = \mathbf{r}_2 = r$.

8.0.16 Exercise: Rotation

Consider a vector in two-dimensions $\mathbf{r}_1 = (x_1, y_1) = (1.5, 0.5)$ as in Fig. 1.

1. Rotate $\mathbf{r}_1$ anti-clockwise around the z-axis by an angle $\theta = \pi/4$, and find the coordinates of the rotated vector $\mathbf{r}_2 = (x_2, y_2) = A\mathbf{r}_1$:

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = A \begin{pmatrix} x_1 \\ y_2 \end{pmatrix} \qquad\qquad A = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} .$$

2. What do you have to change if you would like to rotate $\mathbf{r}_1$ clockwise?
3. Which matrix is the inverse of the rotation matrix A, i.e., A^{-1}? (the command in Matlab is `inv(A)`).
4. Which operation are you doing by considering either `A * inv(A)` or `inv(A) * A` and what do you get?

8.0.17 Exercise: Rotation and shifting of parametric curves

Let's consider the following ellipse

$$x = a\sin(\theta) \qquad\qquad y = b\cos(\theta) \ ;$$

where $\theta \in [0, 2\pi]$ and $a = 2$ and $b = 1$.

1. Plot the ellipse (use the command `axis('equal')` to have the same axis ranges for both x and y).
2. Plot now the ellipse rotated clockwise by an angle $\theta = \pi/8$ and shifted by $(x_0, y_0) = (2, 3)$, i.e., now centered in $(x_0, y_0) = (2, 3)$.

9 Some additional exercise

9.0.18 Exercise

Find the crossing points between a circle centered in $(x_0, y_0) = (2, -3)$ and radius $R = 5$ and the exponential function $f(x) = e^{-x/2} - 4$ (answer: $x_1 \simeq -2.42$ and $x_2 \simeq 6.91$). Plot both functions and mark the crossing points you have found numerically.

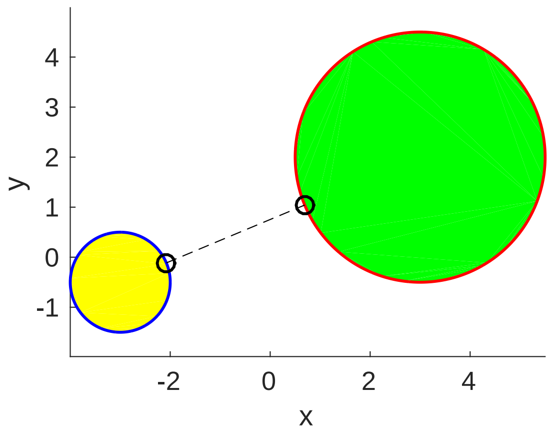

Figure 2: Minimal distance between two circles.

9.0.19 Exercise: Maxwell-Boltzmann distribution

The Maxwell-Boltzmann distribution describes the velocity (v) distribution of a classical gas of atoms with mass m at thermal equilibrium at a given temperature T:

$$f(v) = 4\pi \left(\frac{m}{2\pi k_B T}\right)^{3/2} v^2 e^{-\frac{mv^2}{2k_B T}} \ . \tag{6}$$

In SI units the Boltzmann constant is $k_B = 1.38 \times 10^{-23}$ J/K and let's consider the specific case of Argon atoms, whose mass is $m = 67 \times 10^{-27}$ kg.

1. Numerically evaluate the maximum of this distribution `fmax` for a temperature $T = 300$ K and its corresponding velocity `vmax`; use the loop `for` and compare your result with the built-in operation `max`, taking into account the syntax `[fmax,imax]=max(f)`, where `imax` is the index of the maximum value of the vector `f`;

2. Assuming the system is kept at a pressure low enough to always remain in its gaseous phase in the temperature range $T \in [1, 1000]$ K, plot in two separate subplots `fmax` and `vmax` as a function of temperature.

9.0.20 Exercise: Minimal distance between two circles

Consider the following two circles,

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \ , \tag{7}$$

one with center $(x_{01}, y_{01}) = (-3, -0.5)$ and radius $r_1 = 1$ and the other with $(x_{02}, y_{02}) = (3, 2)$ and $r_2 = 2.5$. Numerically evaluate the minimal distance between the two circles, compare the result you get with the exact one, $\sqrt{(x_{02} - x_{01})^2 + (y_{02} - y_{01})^2} - (r_1 + r_2)$, and represent graphically your result by plotting the two circles and the segment representing the minimal distance (see Fig. 2).

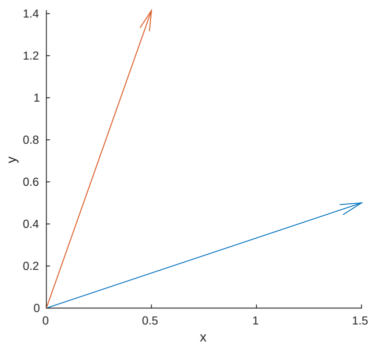

Figure 3: Rotating anti-clockwise the vector $\mathbf{r}_1 = (x_1, y_1) = (1.5, 0.5)$ by an angle $\theta = \pi/4$: representation with the command `quiver(x,y,vx,vy)`.

10 Advanced plotting: scalar fields

In order to plot functions of two variables, such as $f(x, y) = x^2 + y^2$, one needs to generate a two-dimensional grid of points with the commands

```
xmesh=linspace[a,b,N];
ymesh=linspace[c,d,M];
[x,y]=meshgrid(xmesh,ymesh);
f=x.^2+y.^2;
```

One can then choose one of the following options for the plotting of the surface: `contour(x,y,f)`, `mesh(x,y,f)`, `surf(x,y,f)`, `surfc(x,y,f)`, `surfl(x,y,f)`. By using the `help` and by trying the various options, find out what these commands corresponds to.

10.0.21 Exercise:

Write a scripts which plots the following function of two variables $f(x, y) = (x^2 + y^2) - (x^2 + y^2)^2/2$ in the interval $x, y \in [-1, 1]$ using the commands `meshgrid` and `mesh(x,y,f)`. Evaluate the maximum value of the function $f(x, y)$, by making use of a loop `for` and the condition `if`. Compare the result you find this way with the one found with the command `max`, and check they match with the exact answer — N.B. For a matrix A, `max(A)` returns a row vector containing the maximum element from each column. When you have doubts about a command, type `help command`, e.g. in this case `help max`.

11 Advanced plotting: vector fields

Matlab can plot vectors with the command `quiver(x,y,vx,vy)`, which plots velocity vectors as arrows with components `(vx,vy)` at the points `(x,y)`. Here, `x`, `y`, `vx`, and `vy` must be matrices all of the same size — `x` and `y` can also be vectors to specify a uniform grid. N.B. Quiver automatically scales the arrows to fit within the grid. Use the version `quiver(x,y,vx,vy,S)` with `S=0` to plot the arrows without the automatic scaling. Consider the following two examples.

In the first one we reproduce the exercise of Fig. 1; the result is plotted in Fig. 3.

```
r1=[1.5; 0.5];
theta=pi/4;
A=[cos(theta) -sin(theta); sin(theta) cos(theta)];
r2=A*r1;

S=0;
hold on
quiver(0,0,r1(1),r1(2),S)
quiver(0,0,r1(2),r2(2),S)
xlabel('x')
ylabel('y')
axis image
hold off
```

In the second example we plot a two-dimensional random vector field on a grid:

```
N=10; range=2;

xmin=-range; xmax=range;
ymin=-range; ymax=range;

xmesh=linspace(xmin,xmax,N);
ymesh=linspace(ymin,ymax,N);
[x,y]=meshgrid(xmesh,ymesh);

vx=rand(N,N)-0.5;
vy=rand(N,N)-0.5;

quiver(x,y,vx,vy)
axis image
```

11.0.22 Exercise:

Consider the energy potential $U(x,y) = -(x^2+y^2) + (x^2+y^2)^2/2$, plot the potential using the commands `meshgrid` and `mesh(x,y,f)`, evaluate the force field $(F_x, F_y) = -(\partial_x U, \partial_y U)$ and plot it with the command `quiver`. Describe the motion of a particle under such a vector force field. Use first your routine to evaluate the partial derivatives $\partial_x U$ and $\partial_y U$, and then use the build-in Matlab routine `[Fx, Fy] = gradient(U)` to calcualte the gradient of a given scalar field `U`.

12 Applications: Charge distribution

The electric (Coulomb) potential of a point charge Q at a distance $|\mathbf{r}-\mathbf{r}_0|$ from the charge position $\mathbf{r}_0$ is given, in SI units, by

$$U(r) = \frac{Q}{4\pi\epsilon_0|\mathbf{r}-\mathbf{r}_0|} , \tag{8}$$

where ϵ_0 is the vacuum permittivity — in the CGS units, one can fix $1/(4\pi\epsilon_0) = 1$.

12.0.23 Exercise:

Plot the electric potential for an electric charge $Q = -1$ positioned at $\mathbf{r}_0 = (1, 1.5)$ in a two-dimensional (2D) plane $\mathbf{r} = (x, y)$ — generate a 2D mesh covering an interval around $\mathbf{r}_0$ and use the command `meshgrid()`; plot the potential using `contour()`, `mesh()`, `surf()`, either in multiple figures (using `figure()`) or in one figure (using `subplot()`).

12.0.24 Exercise:

Plot the potential for a system of two point charges in a 2D plane, one with charge $Q = -1$ positioned at $\mathbf{r}_0 = (1, 1.5)$ and another of charge $Q = +1$ positioned at $\mathbf{r}_0 = (-1, -1.5)$ — the electric potential for a system of point charges is the sum of the individual potentials.

The electric field of a single point charge located at the origin is given by

$$\mathbf{E} = \frac{Q}{4\pi\epsilon_0}\frac{\hat{\mathbf{r}}}{r^2} . \tag{9}$$

12.0.25 Exercise:

Imposing $1/(4\pi\epsilon_0) = 1$, write a script which plots the electric field $\mathbf{E}$ of a point charge $Q = 1$ located at $(x_0, y_0) = (1, 0)$ and plot $\mathbf{E}$ in the interval $x \in [0, 2]$ and $y \in [-1, 1]$ on a grid of 10 points for each axis.

The electric field due to a collection of charges Q_1, Q_2, ..., is obtained using the principle of superposition, $\mathbf{E} = \mathbf{E}_1 + \mathbf{E}_2 + \ldots$, thus at a given position $\mathbf{r}$,

$$\mathbf{E}(\mathbf{r}) = \frac{1}{4\pi\epsilon_0}\sum_i Q_i \frac{\mathbf{r}-\mathbf{r}_{i0}}{|\mathbf{r}-\mathbf{r}_{i0}|^3} . \tag{10}$$

12.0.26 Exercise:

Write a script which plots the electric field $\mathbf{E}$ of two point charges $Q_1 = 1$ located at $\mathbf{r}_{10} = (1, 0)$ and $Q_2 = -1$ located at $\mathbf{r}_{20} = (-1, 0)$ (dipole).