Matlab Notes for Differential Equations

Lia Vas

Content

0. Review of Matlab

Introduction to Matlab

The main Matlab window is divided into four (or five, depending on the version) parts:

Menu Bar (at the very top)

Workspace Current Folder Command Window

Clicking on "New Script", which is the very first command in the Menu Bar, the Matlab window opens another part, called the

Editor

In some versions of Matlab, the Editor automatically opens on the start.

All Matlab commands are executed in the **Command Window**. If your code is not longer that one line, you can type it in the Command Window and execute it by pressing "enter".

The Editor is used for more complex code. You can have multiple Editor tabs open if you need to work on more than one set of code at a time. The default file is titled "Untitled" so when you type some code in it, you need to save the file and give it a meaningful name. One benefit of typing the code in the Editor is that you can save it while the code typed in the Command Window cannot be saved, just executed. Hence, you can execute the saved code multiple times while executing a command in the Command Window again requires typing the whole command again.

The Directory (**Current Folder** part) shows your current location. If you want to execute some code typed in the Editor, change the location of your current folder to match the folder where you saved the code from the Editor. The current location of the Directory is listed in the long white bar above the Editor.

You can change your current folder by clicking on the folder icon with a green arrow (see the icons on the right side of the white bar listing your current directory). Note that the directory that is opened when Matlab starts is usually not the directory where your files are saved so you will need to change the current folder.

The **Workspace** lists all variables that have been defined and specifies their type and value.

The **Menu Bar** contains some familiar commands, such as New, Open, and Save. When the Editor is open, the Menu Bar contains also Comment and Run. Comment allows us to write text in the code that is not executed which helps your code be more meaningful to you, the programmer, or to a user. Alternatively, you can type "%" before any text and it will be included as a comment.

<u>Solving Equations. Representing Functions.</u>

You can use +, -, *, \ and ^ to add, subtract, multiply, divide or raise to a power, respectively. To perform symbolic calculations, use **syms** to declare the variables you plan to use. For example, suppose that you need factor x^2-3x+2. First you need **syms x** (you are declaring that x is a variable). Then you can use the command **factor(x^2-3*x+2)** to get the answer
ans =(x-1)*(x-2).

Note that we entered **3*x** to represent 3x in the command above. **Entering * for multiplication is always necessary in Matlab.**

For solving equations, you can use the command **solve**.

- Represent the variable you are solving using **syms** command.
- Move every term to the left side of the equation so that the equations of the form
  g(x)=h(x) become g(x)-h(x)=0
- If the term on the left side is f(x) and the equation is
  f(x)=0
  the command you want to execute is **solve(**f(x)**)**

Note that the left side of the equation is in parenthesis. Thus, the command **solve** has the form **solve(***the left side of the equation if the right side is 0***)**

For example, to solve the equation x^3-2x-4=0, you type **solve(x^3-2*x-4)** and get the following answer: **ans = 2 -1+i -1-i.** Here *i* stands for the imaginary number $\sqrt{-1}$. This answer tells us that there is just one real solution, 2.

The command solve often gives you a symbolic answer. For example, when solving the equation 3x^2-8x+2=0 by **solve(3*x^2-8*x+2)** we obtain the answer as
ans = 4/3-10^(1/2)/3 10^(1/2)/3+4/3. If we want to get, often more meaningful, numerical answer in the decimal form with, say, three significant digits, use the command **vpa,** refer to the previous answer as **ans**, and specify how many digits of the answer you want to see. For example **vpa(ans, 3)** produces **ans =0.279 2.39.**

The command **vpa** has the general form

vpa(*expression you want to approximate***,** *number of significant digits***)**

You can solve more than one equation simultaneously. For example suppose that we need to solve the system x^2+ x+ y^2 = 2 and 2x-y = 2. We can use: **>> syms x y**
>> [x,y] =solve(x^2+ x+ y^2-2, 2*x-y-2) to get the answer
x = 1 2/5 y = 0 -6/5
meaning that there are two solutions x=1, y=0 and x=2/5, y=-6/5. Note that the **[x,y]=** part at the beginning of the command is necessary.

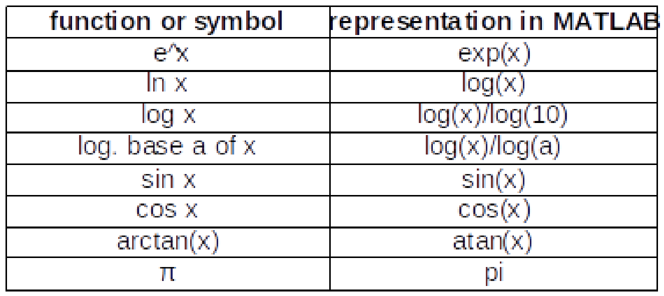
To represent a function given by a formula containing a variable x, start by **syms x** if you have not defined x to be your variable already. If we want to call the function f, the following command defines f(x) to be a function defined by the given formula.

f = @(x) *formula defining the function*

For example, the command **f = @(x) x^2+3*x-2** defines the function x^2+3x-2.

After defining a function, we can evaluate it at a point. For example, **f(2)** produces the answer **ans = 8.**

The following table gives an overview of how most commonly used functions or expressions are represented in Matlab.



function or symbol	representation in MATLAB
e^x	exp(x)
ln x	log(x)
log x	log(x)/log(10)
log. base a of x	log(x)/log(a)
sin x	sin(x)
cos x	cos(x)
arctan(x)	atan(x)
π	pi

As when using the calculator, one must be careful when representing a function. For example,

- $\frac{1}{x(x+6)}$ should be represented as **1/(x*(x+6))** not as **1/x*(x+6)** nor as **1/x(x+6),**
- $\frac{3}{x^2+5x+6}$ should be represented as **3/(x^2+5*x+6)** not as **3/x^2+5*x+6,**
- e^{5x^2} should be represented as **exp(5*x^2)** not as **e^(5*x^2), exp*(5*x^2), exp(5x^2)** nor as **exp^(5*x^2).**
- *ln*(x) should be represented as **log(x)**, not **ln(x).**
- log_3(x^2) should be represented as **log(x^2)/log(3)** not as **log(x)/log(3)*x^2.**

Basic Graphing. Differentiation and Integration

When graphing functions depending on a variable *x,* you can start your session by declaring x for your variable by **syms x**

The simplest way to graph a function is to use the command **ezplot** (easy plot). For example, to graph the function x^2+x+1, use **ezplot(x^2+x+1).** If we need to specify the domain, for example if the variable *x* should take values between -2 and 2, we can use **ezplot(x^2+x+1, [-2, 2])**. If we need to modify the window, for example, if we want to see the graph for *x* in the interval [-2, 2] and for *y* in the interval [1, 4], we can use **axis([-2 2 1 4]).**

Using **hold on** and **hold off**, together with **ezplot** command, you can plot multiple curves on the same window. For example, to graph the functions sin(*x*) and $e^{-x^{\wedge}2}$, you can use

ezplot(sin(x))   **hold on**   **ezplot(exp(-x^2))**   **hold off**

The command for differentiation is **diff**. It has the following form   **diff(***function***)**

For example, the derivative of x^3-2x+5 can be found using **syms x** and **diff(x^3-2*x+5)**.The answer is **ans = 3*x^2-2.**

For the n-th derivative use   **diff(***function, n***)**

For example, to get the second derivative of x^3-2x+5, use **diff(x^3-2*x+5, 2)**
ans = 6*x
Similarly, the 23rd derivative of sin(x) is obtained by **diff(sin(x), 23)**
ans =-cos(x)

For the indefinite integrals, start with **syms x** followed by the command   **int(***function***)**
For example, **int(x^2)** evaluates the integral $\int x^2 dx$. The answer is **ans = 1/3*x^3.**

For definite integrals, the command is   **int(***function, lower bound, upper bound***)**
For example, **int(x^2, 0, 1)** evaluates the integral $\int_0^1 x^2 dx$ The answer is **ans = 1/3.**

1. Using the Editor – basics of programming

Suppose that you need to execute the same operation several times with different input values. Matlab allows you to create a script that you can execute repeatedly (i.e. to write a program). Such a script should be entered as a new file in the Editor (see the previous section). Keep in mind that you should change your current directory to match the directory where you saved the code from the Editor. So, the “Current Folder" should be **the same directory** as the one where your script is saved. In this case, your file should be listed in the Current Folder window.

<u>Example 1.1.</u> Suppose that you want to solve the equation of the type $ax^2 + bx + c = 0$. We can create a function which has the values of a, b and c as the input values and which produces the solution(s) of the quadratic equation x_1 and x_2 as the output values. Let us call such function “quadratic”.

Start your script by

```
function [x1, x2] = quadratic(a, b, c)
```

which specifies that your script is a function with input **a,b,c** and output **x1** and **x2**. The word **“quadratic”** is the name chosen by us. This is followed by the formulas computing x1 and x2. These values of x1 and x2 are automatically displayed by listing **x1** and **x2** in the first line of the code so the two formulas below end by the semicolon symbols.

```
x1 = (-b+sqrt(b^2-4*a*c))/(2*a);
x2 = (-b-sqrt(b^2-4*a*c))/(2*a);
```

It is recommended to save this script using the same name you used in it. It appears in the folder you chose to save it in as quadratic.m.

To execute this script and find solutions of $x^2 -3x + 2 = 0$, for example, you can execute the following command in the command window (or another editor window, but not the same editor window where you have the script quadratic)

```
>> [x1, x2] = quadratic(1, -3, 2)
```

The output is **x1 = 2** **x2 = 1**

Alternatively, the following script can be used.

```
function [ ] = quadratic2(a, b, c)
x1 = (-b+sqrt(b^2-4*a*c))/(2*a)
x2 = (-b-sqrt(b^2-4*a*c))/(2*a)
```

Here we do not need to list the output variables in the first line since we did not use the semicolon symbols when computing the values of x1 and x2 so, those values will be displayed. You can execute this script to solve the same equation as above by **quadratic(1, -3, 2)** instead of by **[x1, x2] = quadratic(1, -3, 2).** The output is the same as before:

```
x1 =    2                              x2 =    1.
```

Both versions of this script produce the complex values if the solutions of a quadratic equation are complex numbers. For example, to solve $x^2 + 4 = 0$, using the second version, we can execute **quadratic(1, 0, 4)** and obtain the answers **x1 = 0 + 2.0000i x2 = 0 - 2.0000i**
Hence, the solutions are $\pm 2i$.

<u>Example 1.2</u>. Let us write a script which calculates the Cartesian coordinates *(x,y)* of a point given by polar coordinates *(r,θ)*.
One possibility is the following.

```
function [x, y] = polar(r, theta)
x = r*cos(theta);
y = r*sin(theta);
```

For example, to calculate *(x,y)* coordinates of for r=3 and $\theta = \pi$, we can use the command

[x,y]=polar(3, pi) and get the answer **x = -3** and **y = 3.6739e-016.**
Note that *y* is very close to 0. The inaccuracy appears because **pi** is only an approximation of *π* not the exact value. Using **sym('pi')** represents *π* more accurately and the command **[x,y]=polar(3, sym('pi'))** gives you the expected answer **x = -3** and **y = 0**.

2. The First Order Differential Equations

<u>Symbolic solutions using **dsolve**</u>

A solution of a differential equation given by a formula is said to be a symbolic solution. Matlab command **dsolve** produces a symbolic solution. In this case, **y(x)** can be declared as a function of x using the command **syms** and the derivative of the function *y* can be represented by **diff(y, x)**. Note that the order of variables here (dependent first, independent second) matches the order in the expression dy/dx. The equality sign in the equation is represented by **==**. The command **dsolve** has the following form.

dsolve(*equation***,** *independent variable***)**

If the equation also has an initial condition, one can get the **particular solution** using the following format. The equality sign in the initial condition should also be represented by **==**.

dsolve(*equation***,** *initial condition***,** *independent variable***)**

<u>**Example 2.1**</u>. Consider the equation $x\,y' - y = 1$. a) Find the general solution. b) Find the particular solution corresponding to the initial condition $y(1)=5$ and graph it.

a) You can find the general solution by **syms y(x)** followed by
>> dsolve(x*diff(y,x)-y==1, x) producing **ans = C1*x-1**
Thus, the general solution consists of functions of the form $y = cx-1$, where c is any constant.

b) The particular solution can be found by
>> dsolve(x*diff(y,x)-y==1, y(1)==5, x) producing the answer **ans = 6*x-1**
This solution can be graphed by **ezplot(ans)**

<u>Numerical solutions using **ode45**.</u>

Many differential equations cannot be solved explicitly in terms of elementary functions. for those equations, approximate solutions can be obtained using numerical methods. Approximate solutions can be found by using the command **ode45**. The following example illustrates the use of this command.

The command **ode45** requires that the equation is in the form $y'=f(x,y)$. Thus, if the equation is not given in this form, you have to solve for y' first. Then you want to represent the formula on the right side as a function **f**. In this case,

ode45(f, [x-initial, x-final**],** y-initial**)**

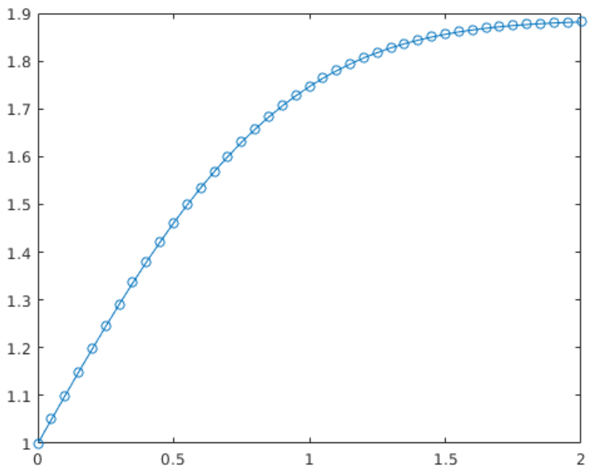
produces a graph of the solution on interval [x-initial, x-final] and

[x,y]=ode45(f, x-initial:step size:x-final**,** y-initial**)**

produces a list of x-values starting at x-initial, ending at x-final with the specified step size and a list of corresponding y-values.

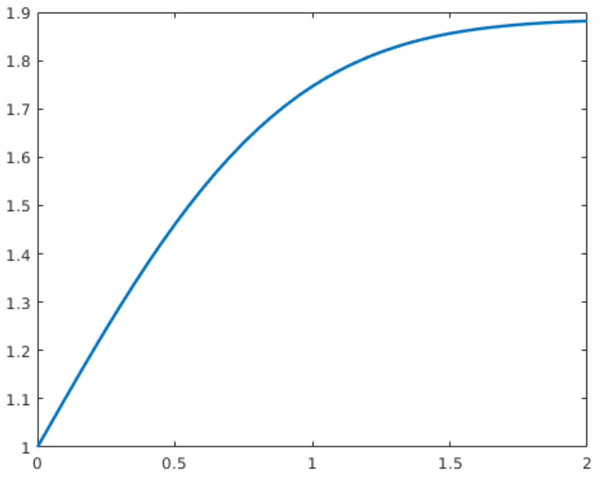
Example 2.2. Consider the initial value problem y' = $e^{(-x^2)}$, y(0)=1. Graph the solution on the interval [0, 2] and display the y-values for x=0, 0.5, 1, 1.5 and 2.

Represent the right side of the equation as a function first by **f=@(x,y) exp(-x^2)**. Then graph on the required interval by **ode45(f, [0 2], 1).**



If you want the graph without the circles around the points at which **ode45** calculates the solution, you can do the following:

```
>> [x, y] = ode45(f, [0 2], 1);
>> plot(x, y)
```



To display the numerical values of the solution, use:

```
>> [x, y] = ode45(f, 0:.5:2, 1)
```

If you want the x and y values to be displayed in two columns next to each other (so that it is easy to see the y-value for a corresponding x-value), use

```
>> [x,y]
```

and get the following output.

```
x =                y =
      0            1.0000
   0.5000          1.4613
   1.0000          1.7468
   1.5000          1.8562
   2.0000          1.8821
```

Besides the previous script, we can also obtain a similar one using **ode45** instead of **dsolve**. To execute it, one first need to solve the equation for y' and represent the right side of the equation as a function in Matlab. This function is the input of the script below.

```
function[ ]=several_solutions(f)
hold on
        for c=the first y-value at initial x : step size : the last y-value at initial x
                [x,y]=ode45(f, [initial x, final x], c);
                plot(x,y)
        end
hold off
```

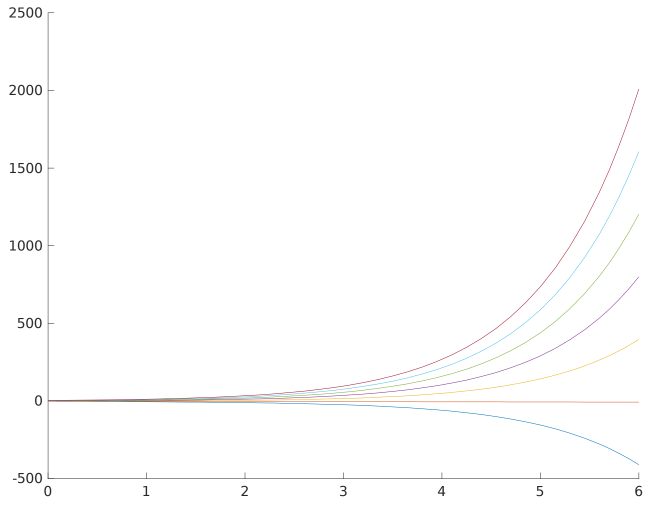
Example 2.3. Graph the solutions of the equation *y' = x+y* for *y(0)* taking integer values between -2 and 4 on interval [0,6].

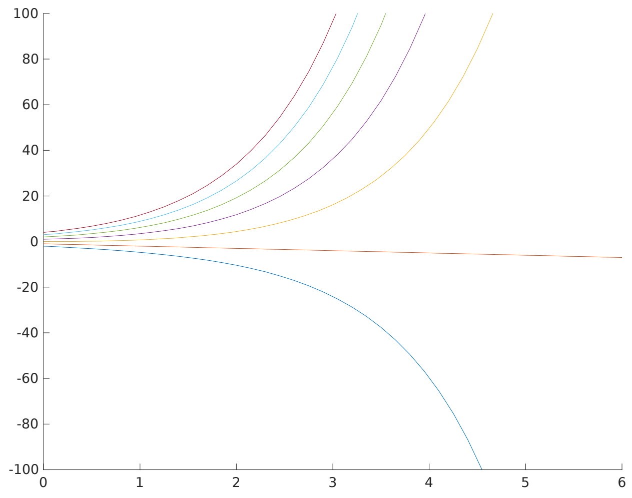
Start by representing the right side of the equation as a function. **>> f = @(x,y) x+y**
Modify the general format of the above script to fit the values from this problem.

```
function[ ]=several_solutions(f)
hold on
        for c=-2:1:4          (this means that the variable c is taking values starting at -2 and
                                                                  ending at 4 with step size of 1.
                [x,y]=ode45(f, [0, 6], c);    (the domain [0,6] is specified in this line)
                plot(x,y)
        end
hold off
```

You can execute the script by **several_solutions(f)**

You can modify the window by using the command **axis([**x_{min}, x_{max}, y_{min}, y_{max}**])**
if necessary. For example, in the previous example you can use **axis([-1 6, -100 100])** to produce the second graph below.



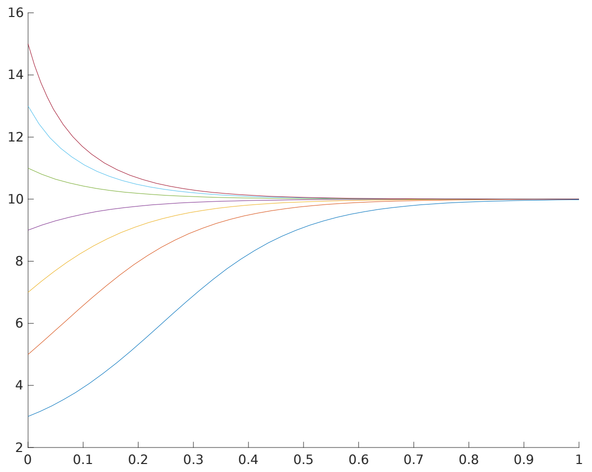


You can modify the script to produce graphs of solutions of different equations as the next example illustrates.

<u>**Example 2.4.**</u> Graph the solutions of the equation $y' = (y\text{-}2)(10\text{-}y)$ for sufficiently many values of *y(0)* larger than 3 so that the limiting behavior of the solutions can be determined.

Represent the right side of the equation as a function first: **f =@(x,y) (y-2)*(10-y)**
Since this is an autonomous equation with the equilibrium solutions at y=2 and y=100, in order to get a telling graph, we can graph a few solutions with initial conditions less than 10 and a few larger than 10. For example, 3,5,7,…,15. Thus, change **c=-2:1:4** to **c=3:2:15**, for example. If you run the script with the line **[x,y]=ode45(f, [0, 6], c)** unchanged, you will see that the solutions converge too quickly for you to be able to see them well. So, you can make

the domain [0,6] smaller, for example [0,1]. With this modification, the graph on the right is obtained.



<u>Euler’s Method</u>

The following script calculates the numerical values of the solution of an initial value problem using Euler’s method. It approximates the solution of the initial value problem $y' = f(x,y)$, $y(x_0) = y_0$ on the interval $[x_0, x_n]$ using n steps using the following formulas:

$h = (x_n - x_0)/n$ (step size), $\quad x_{i+1} = x_i + h$, $\quad$ and $\quad y_{i+1} = y_i + f(x_i, y_i)\, h$

for i=0,1...,n-1. The input is the function f, x_0, y_0, x_n and n. The output is the list of x and y values of the approximate solution.

```
function [x, y] = euler(f, xinit, yinit, xfinal, n)
h = (xfinal - xinit)/n;  (calculates the step size)
x = zeros(n+1, 1);
y = zeros(n+1, 1);  (initialize x and y as column vectors of size n+1 initially filled by zeros)
x(1) = xinit;
y(1) = yinit; (the first entry in the vectors x and y is x0 and y0 respectively)
for i = 1:n
    x(i + 1) = x(i) + h; (every entry in vector x is the previous entry plus the step size h)
    y(i + 1) = y(i) + h*f(x(i), y(i)); (Euler’s Method formula)
end
```

Note that in some cases the step size h, not the number of steps n may be given. Since h = $(x_n - x_0)/n$, you can calculate the number of steps n as n=$(x_n - x_0)/h$ in those cases.

<u>Example 2.5.</u> Approximate the y-value of the solution of the initial value problem y' = $e^{(-x^2)}$ y(0)=1 for x=2. Use Euler’s method with 20 steps. Display the y-values for x=1 and x=2 and the graph of the solution.

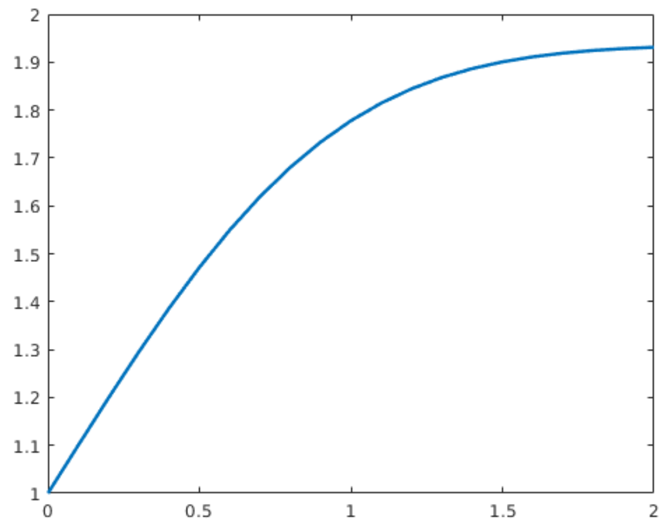
To execute the script **euler**, represent the right side of the equation as a function f.

```
>> f=@(x,y) exp(-x^2);
```

The given initial problem has x_0=0, y_0=1, x_n=2 and n=20. So, execute the script by

```
>> [x,y]=euler(f, 0, 1, 2, 20)
```

The outcome of the command is a list of (x,y)-values. Similarly as for **ode45**, if you want your x and y values to be displayed in two columns next to each other (so that it is easy to see the y-value for a corresponding x-value), simply type **[x,y]**. To graph this list, use **plot(x,y).**



The list consists of 21 (x,y) points. The list starts with the initial condition (x,y)=(0,1). The x-values are all h=(2-0)/20=0.1 units apart. The last x-value is 2 and the corresponding y-value is 1.9311. From this list you can see the y-values corresponding to x=1 and x=2.

x	y
1	1.7778
2	1.9311

Alternatively, if you need to display a specific (x,y)-value, you need to determine the i-value that corresponds to this point (i.e. you need to count the steps performed until the point has been calculated). Then you can display the point by typing x(i) and y(i).

For example, the point with x=1 is the 11th point calculated (Note: not 10th point - recall that x=0 corresponds to first point calculated, not the zeroth point calculated). So, to display this x-value you can type

>> x(11) and obtain the answer **ans = 1.000**

To display the corresponding y-value, you can type

>> y(11) and obtain the answer **ans = 1.7778**

Similarly, the point with x=2 is the 21th point calculated. You can display the x and y values as follows.

```
>> x(21)      ans =   2.000
>> y(21)      ans =   1.9311
```

Practice problems 1

1. a) Find the general solution of the differential equation *y'-2y=6x*.
   b) Find the particular solution with initial condition y(0)=3.
   c) Plot the particular solution on interval [0,2] and find the value of this solution at 2.

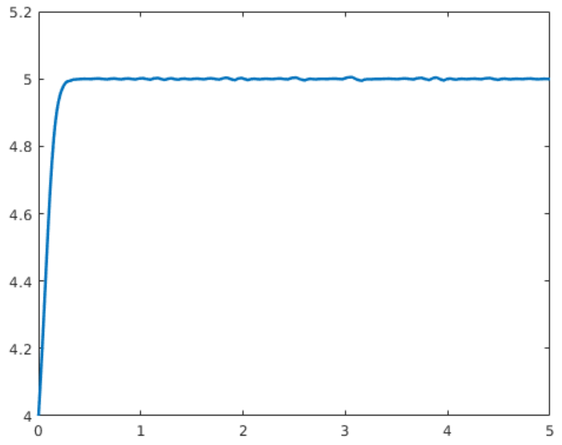
2. Consider the equation y' = (y+1)(3-y)^2(5-y). Using **ode45**, find the value of the solution with the initial condition y(0)=4 at x=5. Graph the solution with initial condition y(0)=4 for x-values in [0, 5].

3. Modify the script several_solutions to graph the solutions of the differential equation *y'=0.1y(1-y)* for the initial condition *y*(0) taking values 0.1, 0.3, 0.5 and 0.7.

4. Consider the equation y' = (y-1)(5-y). Using the script **euler**, find the value of the solution with the initial condition y(0)=2 at x=3 for step size of 0.25. Graph the solution with initial condition y(0)=2 for x-values in [0, 3].

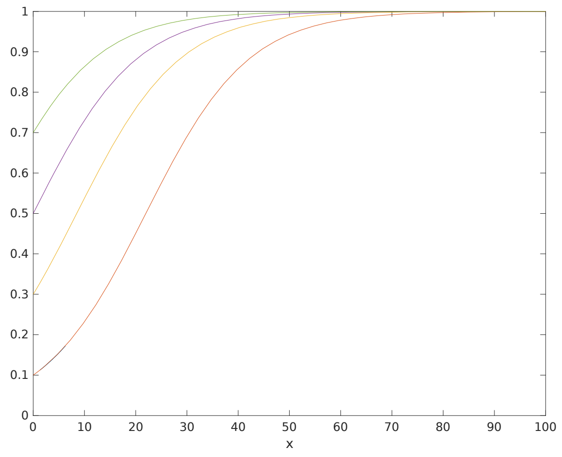
Solutions.

1. a) General solution: **dsolve(diff(y,x)-2*y==6*x, x)** **ans=(C1*exp(2*x))/2-3*x- 3/2**
b) Particular solution: **dsolve(diff(y,x)-2*y==6*x, y(0)==3, x) ans=(9*exp(2*x))/2-3*x- 3/2**
c) **dsolve(diff(y,x)-2*y==6*x, y(0)==3, x) ezplot(ans, [0 2])** To find the value at 2: **f=@(x) (9*exp(2*x))/2-3*x- 3/2** **f(2)** **ans=238.19**



2. First, define the right side equation as a function by **f=@(x,y) (y+1)*(3-y)^2*(5-y)**. To display x and y values use **[x,y]=ode45(f, [0,5], 4)**. The last y-value listed in the output corresponds to value at x=5 and it is **y=4.9997**. Plot the solution by **plot(x,y)**. Obtain the graph on the right.

3. Modify the **several_solutions** script so that the second line is **c=0.1:0.2:0.7** (so that c= 0.1, 0.3, 0.5 and 0.7). Also, to see better the limiting behavior of the solutions, you may want to graph the solutions on domain that includes large values of x, for example [0,100]. Modify the line with ode45 to be **[x,y]=ode45(f, [0, 100], c)** in this case. Execute **f =@(x,y)0.1*y*(1-y)** and **several_solutions(f)** and obtain the graph on the right.



4. First, define the right side equation as a function by **f=@(x,y) (y-1)*(5-y).** Note that the step size of 0.25 corresponds to the number of steps n=(3-0)/0.25=12. Execute the script **euler** by **[x,y]=euler(f, 0, 2, 3, 12)**. To display the answers in two columns, one corresponding to x and the other to y-values use simply **[x,y]**. The list starts by x=0 and y=2. The list ends with x=3 and y=5.000. Graph the list by using **plot(x,y)**. The graph should look similar to the graph from the second problem.

3. Second and Higher Order Differential Equations

<u>Symbolic solutions using **dsolve**</u>

Analogously to the set-up for the first order differential equations, **dsolve** produces a symbolic solution and **ode45** a numerical solution of a second order equation. For command **dsolve**, recall that we represent the first derivative of the function *y* with **diff(y,x)**. The second derivative y'' of **y(x)** can be represented by **diff(y, x, 2)**. The command for a general solution has the same format as for a first order equation **dsolve(***equation***,** *independent variable***)**. The command for a particular solution has the following form

dsolve(*equation***,** *initial condition***,** *independent variable***)**

where the equality sign in the initial conditions should be represented by **==**. For the second initial condition, y’ can be represented as **Dy** and then one can use **Dy(0)** for the second initial condition.

<u>Example 3.1.</u> a) Find the general solution of $y''-3y'+2y = \sin x$.
b) Find the particular solution of the same equation with the initial conditions $y(0) = 1, y'(0)=-1.$

a) For the general solution use **syms y(x)** followed by
>> dsolve(diff(y, x, 2)-3*diff(y, x)+2*y==sin(x), x) producing the answer
ans = C1*exp(x) + (10^(1/2)*cos(x - atan(1/3)))/10 + C2*exp(2*x)
b) For the particular solution use **Dy=diff(y,x)** followed by
>> **dsolve(diff(y, x, 2)-3*diff(y, x)+2*y==sin(x), y(0)==1, Dy(0)==-1, x)** producing the answer **ans = (5*exp(x))/2 - (9*exp(2*x))/5 + (10^(1/2)*cos(x - atan(1/3)))/10**

<u>Numerical solution using **ode45**</u>

For equations that can not be solved in terms of elementary functions, we use numerical methods. For ode45, the second order differential equation must be ***converted to a system of two first order equations*** using the substitution

(S) $y=y_1$ and $y'=y_2$.

The first equation of the new system reflects the relation between the two new functions: the second one is the derivative of the first. So, the first equation is

(1) $y_1'=y_2$.

The second equation is obtained by applying the substitution to the original differential equation and solving it for $y''=y_2'$. For example, the equation $y''-3y'+2y = \sin x$ from the previous example with substitution (S) becomes $y_2'-3y_2 +2 y_1 = \sin x$. Solving for y_2' produces the second equation of the system to be $y_2'=\sin x+3y_2-2y_1$. Thus, the system is

(1) $y_1'=y_2$ and (2) $y_2'=\sin x+3y_2-2y_1$

The two new functions can be denoted by **y(1)** and **y(2)** in Matlab. The solution **y** is a vector function **y=[y(1); y(2)]**. Keep in mind that **y(1)** corresponds to the solution y of the original equation and that **y(2)** corresponds to the derivative y'. In most cases, we are looking for the function y, not for the derivative y'. So, you can consider **y(2)** to be a byproduct.

If you represent the formulas on the right side as a function **f** by **f=@(x, y) [**first eq; second eq**]**, you can graph the solutions $y=y_1$ and $y'=y_2$ as functions of x by

ode45(f, [x-initial**,** x-final**], [**y-initial**;** y’-initial**])**

You can obtain the list of x, y and y’ values by

[x,y]=ode45(f, x-initial: step size:x-final**, [**y-initial**;** y’-initial**])**

We illustrate these commands in the following example.

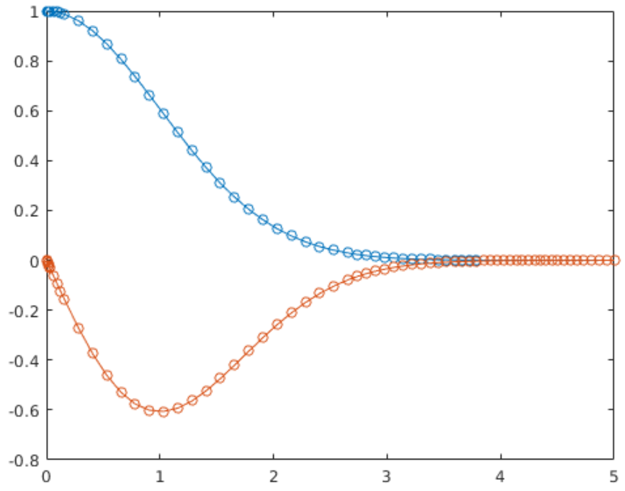
<u>Example 3.2.</u> Consider the following initial value problem

$y'' + x\,y' + y = 0$ with $y(0)=1$ and $y'(0)=0$.

(a) Using ode45, graph the solution on interval [0, 5].
(b) Using ode45, display the list of *y*-values of the solution for the integer *x*-values from 0 to 5.

First, you need to convert the given second order differential equation into a system of two first order equations using the substitution (S) $y=y_1$ and $y'=y_2$. The first equation of the new system is (1) $y_1'=y_2$. The second equation is obtained by using the substitution for the given equation $y'' + xy' + y = 0$ to obtain $y_2' + xy_2 + y_1 = 0$ and then solving for y_2' to get (2) $y_2' = -xy_2 - y_1$. Thus, the newly obtained system is

(1) $y_1'=y_2$ and (2) $y_2'=-x\,y_2 - y_1$.



Represent the right side of the two equations as a vector function **f** that depends on independent variable **x** and dependent variable **y** consisting of two functions **y(1)** and **y(2)**.

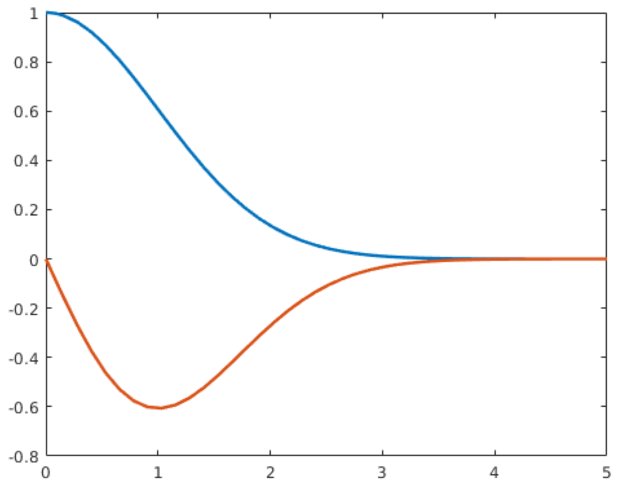
f=@(x,y) [y(2); -x*y(2)-y(1)]

The first entry of **f** is the right side of the first equation and the second entry of **f** is the right side of the second equation.

(a) Graph the solution on interval [0, 5] by

ode45(f, [0, 5], [1;0]).

In this command, **[0, 5]** indicates the interval for *x* and **[1;0]** indicate the initial values *y*(0)=1 and *y*'(0)=0. The output is a graph with two functions **y(1)** representing the solution *y*, plotted in blue, and **y(2)** representing its derivative *y*', plotted in orange.



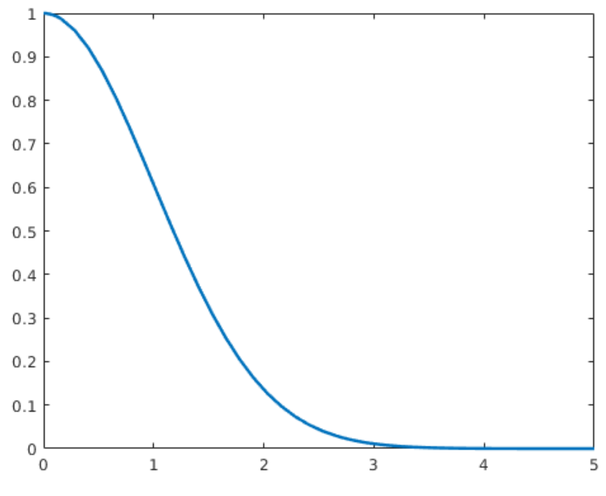
Alternatively, you can use

[x,y]=ode45(f, [0, 5], [1;0]); followed by **plot(x,y)**

which produces graphs without displaying the values used to plot them. The command starting by "**[x,y]=**" calculates numerical values of the solutions and the semicolon at its end suppresses them to be displayed. The result is the second graph on this page.

To display graph of y without the graph of *y*', you can use **[x,y]=ode45(f, [0, 5], [1;0]);** followed by **plot(x,y(:,1))**

This produces the third graph on this page.



(b) To obtain numerical values of the solution use

[x, y] = ode45(f, [0:1:5], [1;0])

Here **[0:1:5]** indicates that the x is taking values starting at 0, ending at 5 at step 1 away from each other. **[1;0]** indicates the initial values, just as in part (a). The vector **y** in the output will consist of two columns. The first **y(1)** consists of the values of the solution *y* at *x*-values between 0 and 5 and the second **y(2)** consists of the values of the derivative *y*' at these points. If you want the *x* and *y* values to be displayed in columns next to each other (so that it is easy to see the y-value for a corresponding *x*-value), you can use **[x,y].** Obtain the list

```
ans =        0      1.0000    0
        1.0000    0.6065   -0.6065
        2.0000    0.1354   -0.2707
        3.0000    0.0111   -0.0333
        4.0000    0.0003   -0.0013
        5.0000    0.0000   -0.0000
```

Here the three columns represent *x, y* and *y*' values respectively.

To display the list without the y'-values, use **[x,y(:,1)].**

Finding zeros of polynomials.

Unlike the situation for quadratic equation, there is no general formula for polynomials of degrees higher than 4 (Wikipedia is a good source for more details). Even for cubic or quartic polynomials when such formula exists, it is rather complex to use. Thus, unless a polynomial is easy to factor or to use the n-th root formula, it is convenient to find approximate solutions using Matlab or some other technology.

In Matlab, you can find zeros of polynomial $a_n x^n + a_{(n-1)} x^{(n-1)} + \ldots + a_1 x + a_0 = 0$ using the command **roots**. Represent the polynomial as a vector of length n+1 with coefficients of the polynomial as the entries

p=[a_n a_(n-1) …. a_1 a_0]

and then use the command **roots(p)**

Example 3.3. Find general solution of the equation $-90y^{(4)} + 100y''' - 54y' + 16y = 0$ by using Matlab to find solutions of the characteristic equation.

The characteristic equation is $-90r^4 + 100r^3 - 54r + 16 = 0$. Represent the left side in Matlab as **p=[-90 100 0 -54 16]** and use the command **roots(p)** to get the solutions r= -0.6900, 0.3511, 0.7250 +0.4562i and 0.7250 -0.4562i .

This gives you four fundamental solutions y_1= $e^{(-0.69x)}$, y_2= $e^{(0.3511x)}$, y_3= $e^{(0.7250x)}\cos(0.4562x)$ and y_4= $e^{(0.7250x)}\sin(0.4562x)$. So, the general solution is $y = c_1 e^{(-0.69x)} + c_2 e^{(0.3511x)} + c_3 e^{(0.7250x)}\cos(0.4562x) + c_4 e^{(0.7250x)}\sin(0.4562x)$.

Practice problems 2

1. a) Find the general solution of the equation *y*''-4 *y*'+4 *y*= e^x +*x*². b) Find particular solution of the initial value problem with *y*(0)=8, *y*'(0)=3.
2. Using **ode45** graph the solution of *y*'' + *x*² *y*' + *y* = cos 2*x*, *y*(0)=1, *y*'(0)=-1 for t in [0, 4].
3. Find general solutions of the equation $-18y^{(5)} + 25y^{(4)} - 27y'' + 16y' + 20y = 0$. Use Matlab to find the solutions of the characteristic equation.

Solutions

1. a) **dsolve(diff(y,x,2)-4*diff(y,x)+4*y==exp(x)+x^2, x)**
b) **Dy=diff(y,x)** **dsolve(diff(y,x,2)-4*diff(y,x)+4*y==exp(x)^2+x^2, y(0)==8, Dy(0)==3, x)**
ans = x/2 + (61*exp(2*x))/8 -(51*x*exp(2*x))/4 + (x^2*exp(2*x))/2 + x^2/4 + 3/8

2. Convert to a system using $y=y_1$ and $y'=y_2$. The first equation of the system is $y_1'=y_2$ and the second is obtained from $y'' + x^2y' + y = \cos 2x \rightarrow y_2' + x^2y_2 + y_1 = \cos 2x \rightarrow y_2'=\cos 2x - x^2 y_2 - y_1$. So the system is (1) $y_1'=y_2$ and (2) $y_2'=\cos 2x - x^2 y_2 - y_1$
Represent the right side of the two equations as a vector function **f** by

f=@(x,y) [y(2); cos(2*x)-x^2*y(2)-y(1)]

To graph the solution on interval [0, 4], you can use **ode45(f, [0, 4], [1;-1])**. The values **[1;-1]** correspond to the y and y' values from the initial conditions.

3. The characteristic equations corresponds to a polynomial which can be represented in Matlab as **p=[-18 25 0 -27 16 20]**. The command **roots(p)** gives you the following values: 1.29, 0.76+0.97i, 0.76-0.97i, -0.72+0.20i, and -0.72-0.20i. Thus, the general solutions is

$$y=c_1e^{(1.29x)}+c_2e^{(0.76x)}\cos(0.97x)+c_3e^{(0.76x)}\sin(0.97x)+c_4e^{(-0.72x)}\cos(0.20x)+c_5e^{(-0.72x)}\sin(0.20x).$$

4. Systems of Differential Equations

Symbolic solutions using **dsolve**

The command **dsolve** can be used for finding symbolic solutions of a system of differential equations. The next example illustrates finding general and particular solutions using this command.

Example 4.1. Consider the system dx/dt=2x-y dy/dt=3x-2y
a) Find the general solution of this system.
b) Find the particular solution of the initial value problem with x(0)=1 and y(0)=2.
c) Graph the particular solution on interval 0≤t≤20.

a) **syms x(t) y(t)** followed by

[x,y] = dsolve(diff(x,t)==2*x – y, diff(y,t)==3*x – 2*y, t)

produces **x = C1*exp(t) + (C2*exp(-t))/3** and **y =C1*exp(t) + C2*exp(-t)**
b) **syms x(t) y(t)** followed by

[x,y] = dsolve(diff(x,t)==2*x - y, diff(y,t)==3*x - 2*y, x(0)==1, y(0)==2, t)

produces **x = exp(-t)/2 + exp(t)/2** and **y =(3*exp(-t))/2 + exp(t)/2**
c) To graph these two solutions on [0,20], you can use

ezplot(x, [0,20]) **hold on** **ezplot(y, [0,20])** **hold off**

Numerical solution using **ode45** and phase plane graphs

Finding symbolic solutions might be very limiting because many systems of differential equations cannot be solved explicitly in terms of elementary functions. For those equations or systems of equations, numerical methods are used. We focus on the command **ode45**. In order to use it, the system needs to be in the form x'=f(x,y,t) and y'=g(x,y,t) and the right sides of the equations should be represented as a vector function first. The function x can be represented as **y(1)** and the function y as **y(2)**. The first entry of the function **f** is the right side of the first equation and the second entry of **f** is the right side of the second equation. The next example illustrates this.

Example 4.2. Consider the system

$$dx/dt=2x-x^2-xy \qquad dy/dt=xy-y$$

with the initial conditions $x(0)=1$ and $y(0)=2$.

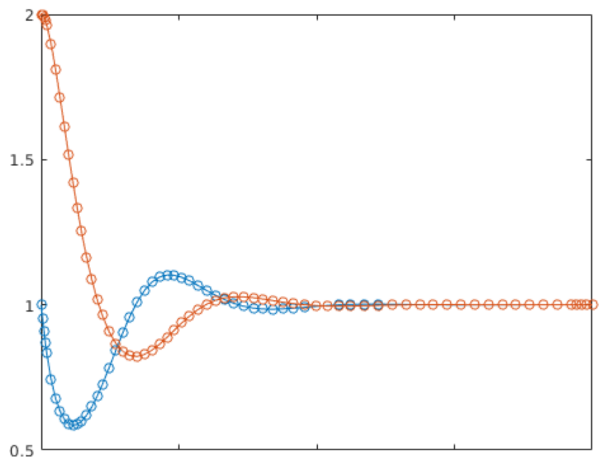
(a) Display the (x,y)-values of a numerical solution for t taking integer values between 0 and 6.
(b) Graph the solutions on interval [0,20].
(c) Graph the solution from b) in the phase plane.
(d) Plot sufficiently many solutions of this system (without the given initial conditions) in the phase plane to determine the type of the equilibrium point (1,1).
(e) Graph the solutions from part d) in the tx and ty planes.

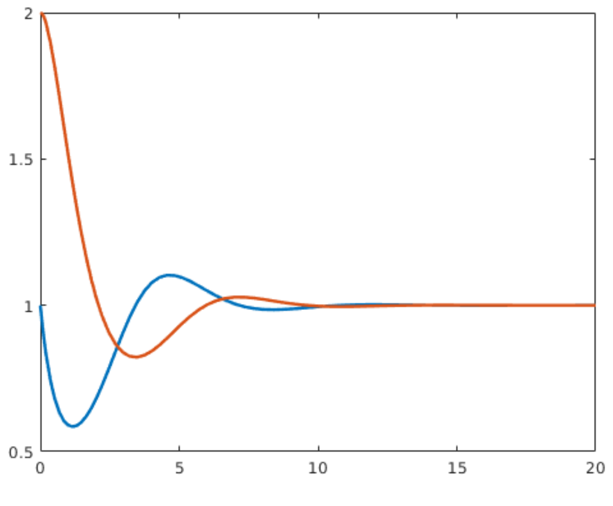
Represent the right side of equation as a vector function of independent variable t. The unknown functions x and y are represented by **y(1)** and **y(2)** respectively.

f = @(t,y) [2*y(1)-y(1)^2-y(1)*y(2); y(1)*y(2)-y(2)]

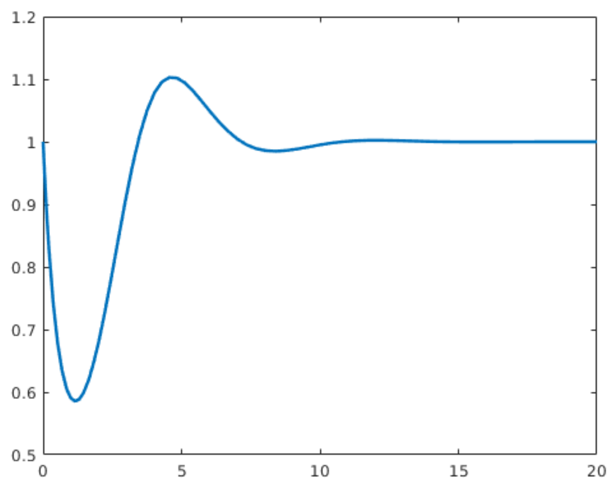
(a) The command **[t,y]=ode45(f, [0:1:6], [1;2])** computes the (x,y)-values at t=0,1,2,...,6. The part **[0:1:6]** indicates that t-values start at 0, end at 6 and are 1 step away from each other. The part **[1;2]** reflects the initial conditions $x(0)=1$ and $y(0)=2$. Note that here **y** is a vector whose entries will be the values of **y(1)** representing x and **y(2)** representing y. To display t, x and y values in columns next to each other (so that it is easy to see the (x,y)-value for a corresponding t-value), you can use **[t,y].** Obtain the list

```
ans =    0    1.0000    2.0000
    1.0000    0.5909    1.5148
    2.0000    0.6791    1.0294
    3.0000    0.9123    0.8370
    4.0000    1.0763    0.8424
    5.0000    1.0980    0.9281
    6.0000    1.0502    1.0016
```



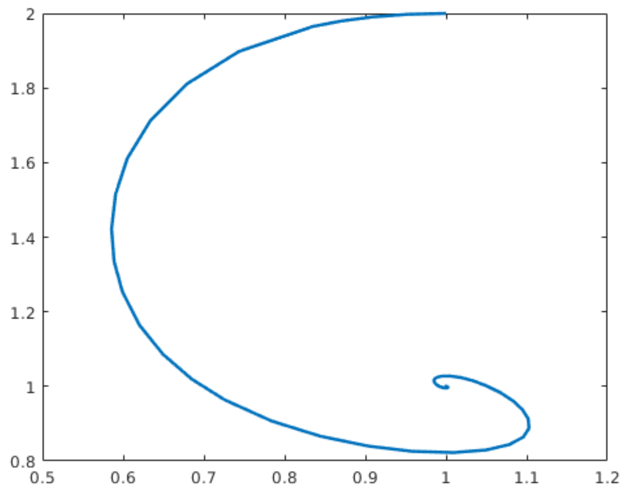


(b) The command **ode45(f,[0,20],[1;2])** graphs the two solutions on the same plot as functions of t. The function x will be graphed in blue and y in orange (the first graph).



Alternatively, you can graph using **[t,y]=ode45(f,[0 20],[1;2]);** followed by **plot(t,y)** which produces graphs without displaying the values used to plot them. The command starting by "**[x,y]=**" calculates numerical values of the solutions and the semicolon at its end suppresses them to be displayed. The result is the second graph. Note that both x and y approach 1 for large values of t. This may be relevant when determining the stability of the equilibrium point (1,1).

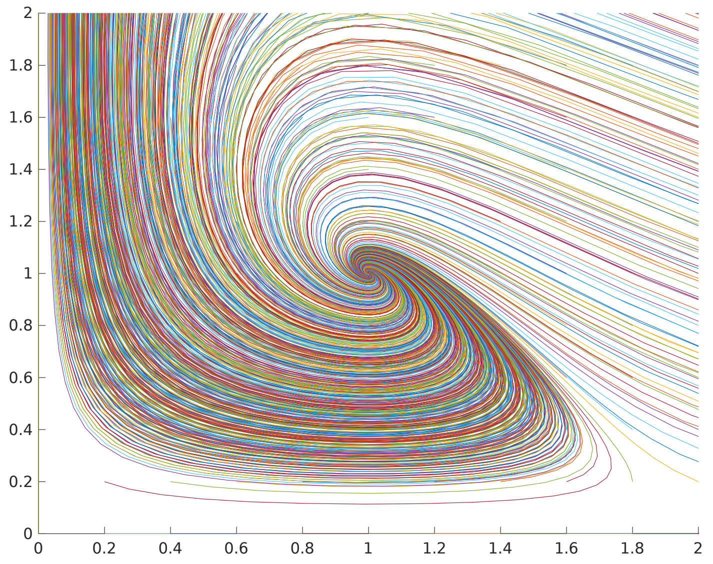
You may need to display just the first or just the second function. In this case, the command **[t,y]=ode45(f,[0 20],[1;2]);** followed by **plot(t,y(:,1))** plots just *x*-values (the graph on the right).



Similarly, the command **[t,y]=ode45(f,[0 20],[1;2]);** followed by **plot(t,y(:,2))** plots just *y*-values.

(c) The command **[t,y] = ode45(f,[0,20],[1;2]);** followed by **plot(y(:,1),y(:,2))** plots the solution in the phase plane: it plots (*x,y*) as a parametric curve of parameter *t* in (*x,y*)-plane.

The graph on the right displays the output of this command. Note that we can determine the **orientation** of this parametric curve using the previous graphs. In particular, from the previous graphs we can note that both x and y values converge to 1. Moreover, the initial condition (x,y)=(1,2) indicates that this is the initial point. Thus, the curve is traced starting from the point (1,2) and ending at the point (1,1).

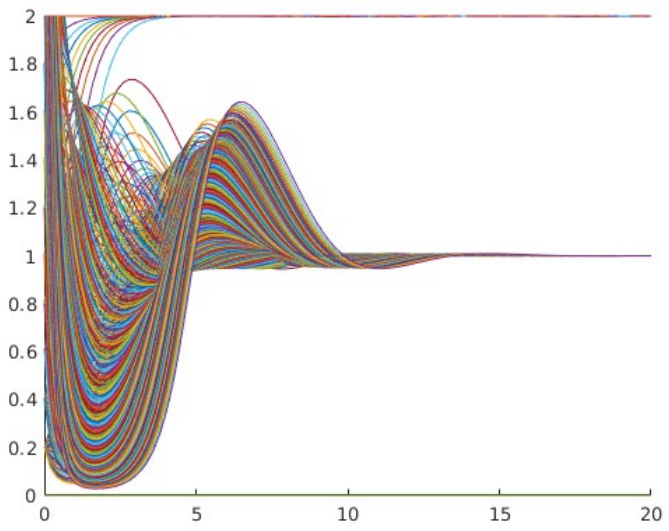


(d) The following script can be used to graph the trajectories in the phase plane for x(0) and y(0) taking initial values 0,1,2,…,5.

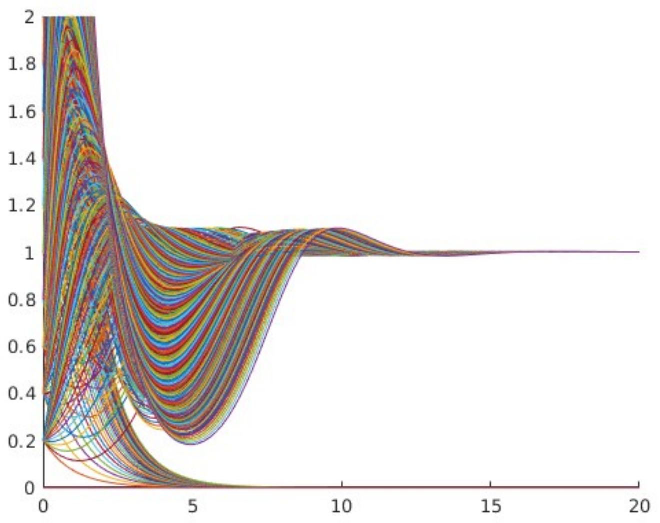
close all; hold on
for a = 0:0.2:5
  for b = 0:0.2:5 (modify these values if necessary to change the density and starting points of the curves)
    [t, y] = ode45(f, 0:0.2:20, [a; b]);
    plot(y(:,1), y(:,2))
  end
end

```
hold off
axis([0 2 0 2])
```
(modify these values to change the window)

From the graph, we can tell that (1,1) is a spiral point. The stability can be determined by analyzing the graphs of the solution in part a and b for example which indicate that both x and y approach 1. Thus, (1,1) is a stable spiral point.



(e) One can obtain the graphs in the tx-plane and the ty-plane by changing the command **plot(y(:,1), y(:,2))** of the above script to **plot(t, y(:,1))** and **plot(t, y(:,2))** respectively. One should also change the axis command in the end to match the t-initial and t-final values from the ode45 command. For example, to get the graphs of several solutions of this system in the tx-plane, one can use the following script.



```
close all; hold on
for a = 0:.2:5
  for b = 0:.2:5
    [t, y] = ode45(f, 0:0.2:20, [a; b]);
    plot(t, y(:,1))
  end
end
hold off
axis([0 20 0 2])
```

Analogous script, with **plot(t, y(:,2))** instead of **plot(t, y(:,1))** can be used for graphs in the ty-plane. The second graph on the right is the outcome of such script. Note that the two graphs indicate that the point (2,0) is also an equilibrium point.

<u>Practice problems 3</u>

1. Consider the system dx/dt = 3x-y and dy/dt = 4x-2y.
   a) Show that the point (0,0) is the only equilibrium point of the system.
   b) Graph a number of the solution in the phase plane and use the graph to determine the type of the equilibrium point (0,0) and its stability.
   c) Graph the solution with initial conditions x(0)=1 and y(0)=2 as a function of t and use the graphs to determine the orientation of the solution in the phase plane.
   d) Use your conclusions to determine the limiting values of the general solutions x and y for t $\rightarrow\infty$ for various values of initial conditions of x and y.
2. The sizes R and W of a population of rabbits and a population of wolves are described

using the predator-prey model

dR/dt=0.08R-0.001RW   and   dW/dt=-0.02W+0.00002RW.

a) Find the equilibrium points.

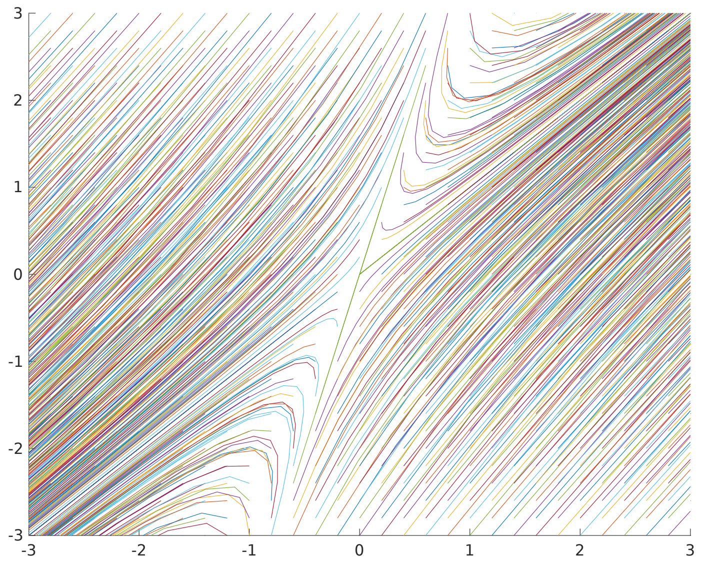
b) Graph a number of solutions in the phase plane. Classify the equilibrium points and determine their stability.

c) Graph the solutions with initial conditions R(0)=400 and W(0)=100 as functions of t. Use this graph to indicate the direction in which the curves in the phase place are traced as the parameter increases. Discuss the long term tendencies of the system.

d) Using the second graph, estimate the values in between which the solutions on it oscillate.

Solutions

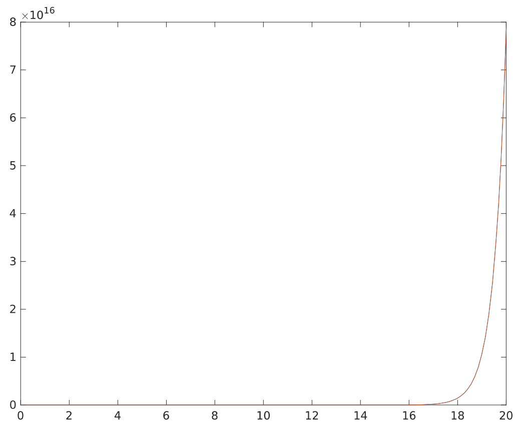
1. a) Solve the equations 3x-y=0 and 4x-2y=0 to obtain that x=0 and y=0 is the only solution. Thus, (0,0) is the only equilibrium point.



b) Modify the script for graphs in the phase plane to make it centered at 0 and sufficiently many solutions displayed. For example,

```
close all; hold on
for a = -3:.2:3
  for b = -3:.2:3
    [t, y] = ode45(f, 0:0.2:20, [a; b]);
    plot(y(:,1), y(:,2))
  end
end
hold off
axis([-3 3 -3 3])
```

Then represent the right side as a function **f** by **f=@(t,y)[3*y(1)-y(2); 4*y(1)-2*y(2)]**, execute the script and obtain the graph on the right. We see that (0,0) is a saddle point and, as such, unstable.



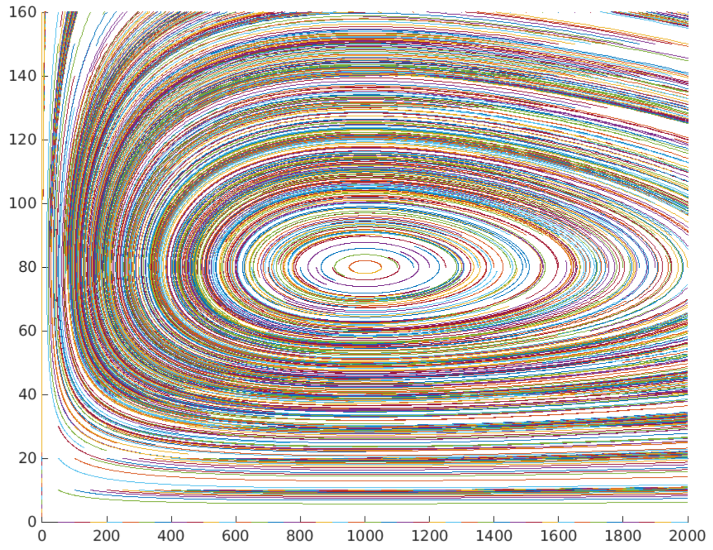
c) Graph by **[t,y]=ode45(f,[0 20],[1;2]); plot(t,y)**. From the graph, we can see that y →∞ when t →∞. Find the point (1,2) on the phase plane graph and note that y-value increase to infinity when the curve is traced moving to the right so that both x and y values are increasing to infinity.

d) The previous part tells us that the solution visible as a green line "halving" the solutions is a separatrix. If the initial point (x(0), y(0)) is on the left side of it, both x and y decrease to negative infinity when t →∞. If the initial point (x(0), y(0)) is on the right side of the separatrix, both x and y increase to positive infinity when t →∞.

2. a) Solve the system 0.08R-0.001RW=0 and -0.02W+0.00002RW=0. The first equation produces two cases R=0 and W=80. In the first case, the second equation implies W=0. In the second case, the second equation becomes -1.6+.0016R=0 which implies R=1000. Hence, the equilibrium points are (0,0) and (1000, 80).

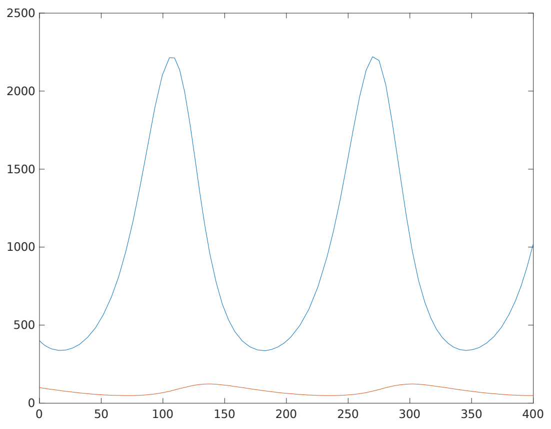
b) Modify the script for graphs in the phase plane to make both critical points visible. Since negative numbers of rabbits and wolves are not relevant, you can display only the first quadrant. Do not make the step size too small since the x-values are large because it may take a long time for the graph to appear otherwise. For example,

```
close all; hold on
for a = 0:50:2000
  for b = 0:10:160
    [t, y] = ode45(f, 0:0.2:100, [a; b]);
    plot(y(:,1), y(:,2))
  end
end
hold off
axis([0 2000 0 160])
```



Then use **f=@(t,y)[.08*y(1)-.001*y(1)*y(2); -.02*y(2)+.00002*y(1)*y(2)]**, execute the script, and obtain the first graph on the right. We see that (0,0) is a saddle point and, as such, unstable and that (1000, 80) is a center and, as such, stable but not asymptotically stable.

The solutions oscillate: x-values about 1000 and y-values about 80. The amplitude of a solution depends on the initial conditions.



b) Graph by **[t,y]=ode45(f,[0 20],[400;100]); plot(t,y)**. The graph enables us to determine the direction of the trajectories on the first graph: since starting with 400, the x-values decrease a bit at first but then increase and the y-values decrease first starting at 100. Hence, we can conclude that the curves in the phase plane are traversed in the counter clock-wise direction.

c) With 400 rabbits and 100 wolves initially, the number of rabbits oscillates between about 460 and 2300 and the number of wolves between about 50 and 200.