Differential Equations : Plotting in MATLAB

Amanda Falke : abstractmachines @ gmail October 2014

Preface

Part II (equilibrium): Matlab FSOLVE tutorial for equilibrium solutions which student used as a basis for student solutions is the intellectual property of Seonmin Ahn, a Computer TA at Brown University. (seonmin_ahn@brown.edu) link here to tutorial. Student also used this MIT tutorial for Part II. All other tutorials and code are from tutorials on Mathworks.com.

Continued on next page.

I. Slope Fields : Quiver functionality in MATLAB

<u>(1) Linearization of Matrices for directional fields :</u>

<u>Source:</u> The methods for using MATLAB for slope fields are found in an instructional PDF by Professor Yonatan Katznelson at University of California, Santa Cruz. <u>Link here.</u>

Each matrix must be broken into systems of equations in order to use the quiver functionality in Matlab (see Blanchard textbook section 3.5, and see the mathworks.com website about Matlab). Here are those calculations. *Each equation was linearized before put into the system. Handwritten work is also attached to this report to verify this work.*

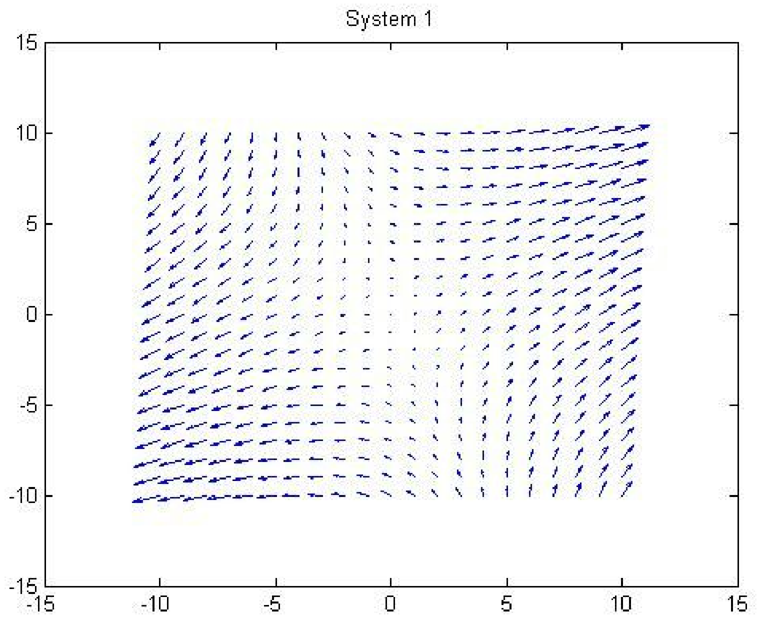
<u>System 1: Matrix A = [4 2 ; 3 -1]</u>

dx/dt = 4x + 2y   dy/dt = 3x - y

CODE:

```
[x,y] = meshgrid([-10:1:10],[-10:1:10]);
Ax = 4.*x + 2.*y;
Ay = 3.*x - y;
figure
quiver(x,y,Ax,Ay)
title('System 1')
```

IMAGE:



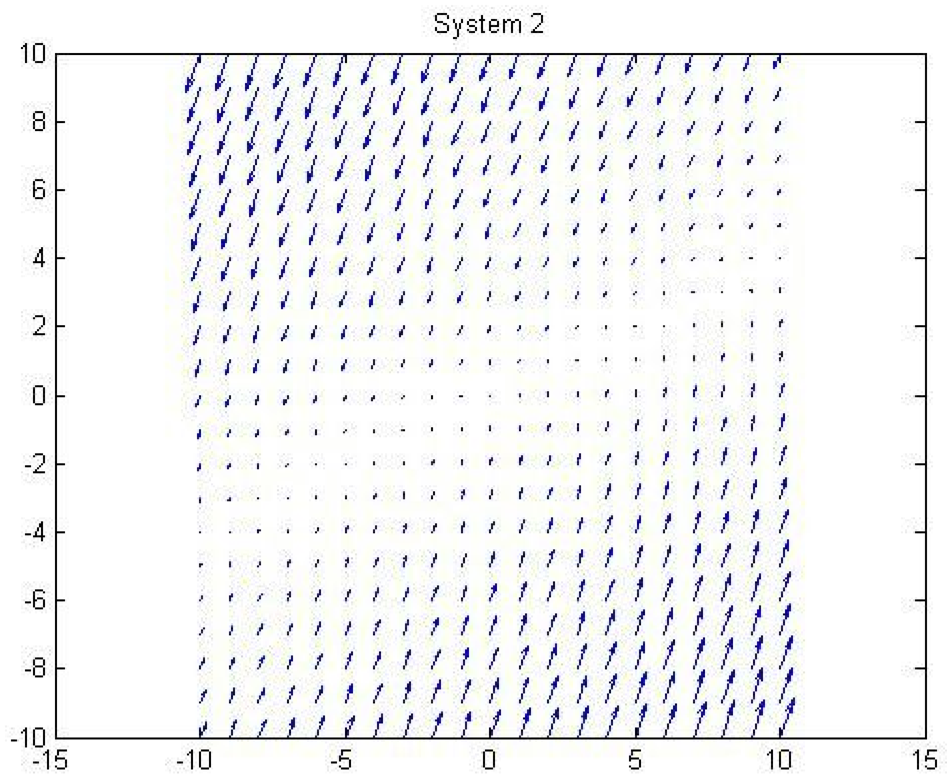
System 2: Matrix A = [1 -3 ; 3 7]

dx/dt = x - 3y   dy/dt = 3x + 7y

CODE:

```
[x,y] = meshgrid([-10:1:10],[-10:1:10]);
Ax = x - 3.*y;
Ay = 3.*x - 7.*y;
figure
quiver(x,y,Ax,Ay)
title('System 2')
```

IMAGE:

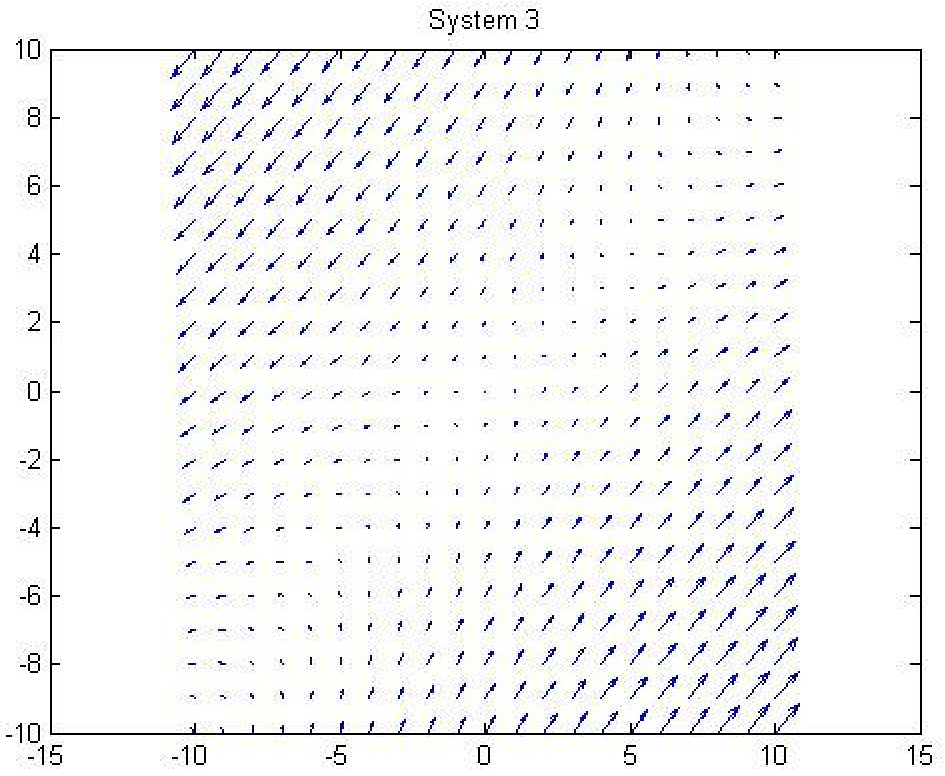


System 3: Matrix A = [4 -3 ; 3 4]
dx/dt = 4x - 3y dy/dt = 3x + 4y

CODE:

```
[x,y] = meshgrid([-10:1:10],[-10:1:10]);
Ax = 4.*x - 3.*y;
Ay = 3.*x - 4.*y;
figure
quiver(x,y,Ax,Ay)
title('System 3')
```

IMAGE:



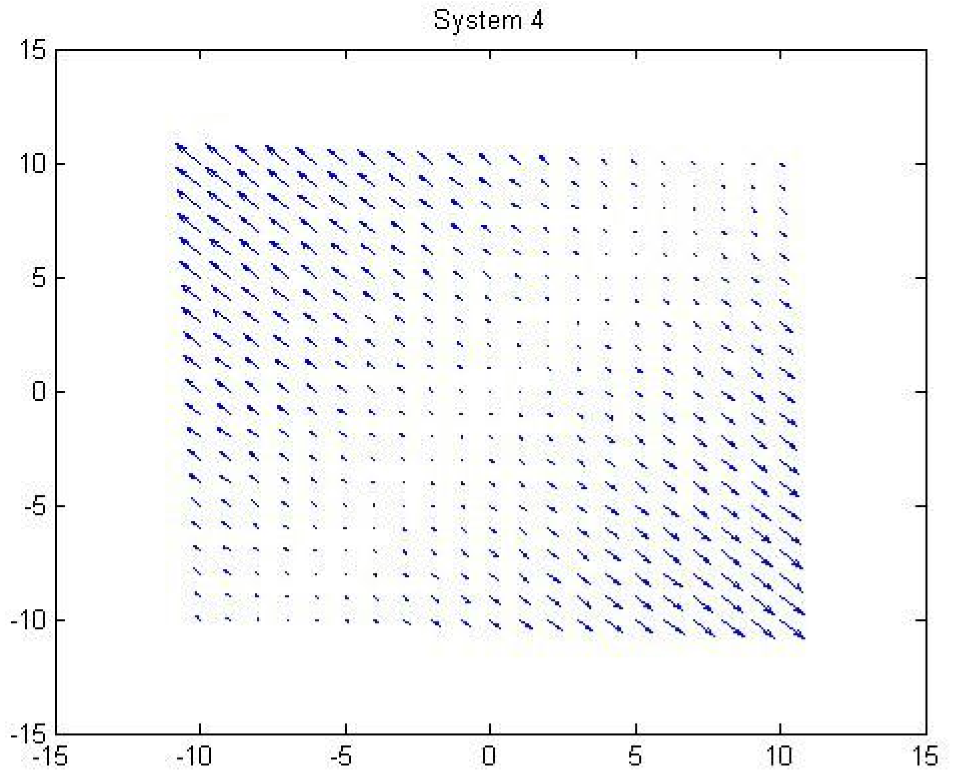
<u>System 4: Matrix A =[4 -3 ; -4 3]</u>

dx/dt = 4x - 3y   dy/dt = -4x + 3y

CODE:

```
[x,y] = meshgrid([-10:1:10],[-10:1:10]);
Ax = 4.*x - 3.*y;
Ay = -4.*x + 3.*y;
figure
quiver(x,y,Ax,Ay)
title('System 4')
```

IMAGE:



II. Identify all equilibria, trivial and non-trivial. fsolve in MATLAB.

System 1: Matrix A = [4 2 ; 3 -1]

Recall the system is: dx/dt = 4x + 2y dy/dt = 3x - y

SYSTEM 1 CODE:

```
function system1_equilibrium
% VARIABLES: x(1) = x    x(2) =y
% x0 is our Initial Condition
x0 = [1 1];
 options=optimset('Display','iter');
x = fsolve(@system1_equilibrium,x0,options)
end

function F = system1_equilibrium(x)
% Rewrite the equation in the form F(x) % = 0 %
 F = [4*x(1)+ 2*x(2);
   3*x(1)-x(2)];
end
```

SYSTEM 1 OUTPUT:

```
> system1_eq

                                    Norm of      First-order   Trust-region
 Iteration  Func-count     f(x)          step         optimality    radius
     0         3             40                          30            1
     1         6         2.75445            1          4.7             1
     2         9     7.70372e-32     0.685695      8.33e-16            2.5

Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>

x =

   1.0e-15 *

            0.0555   -0.1110
```

System 2: Matrix A = [1 -3 ; 3 7]

Recall the system is: dx/dt = x - 3y dy/dt = 3x + 7y

SYSTEM 2 CODE:

```
function system2_equilibrium
% x(1) = x, x(2) = y
% x0 is initial condition
x0 = [1 1];
options=optimset('Display','iter');
x = fsolve(@system2_equilibrium,x0,options)
end

function F = system2_equilibrium(x)
% Rewrite the equation in the form F(x) = 0 %
 F = [x(1)- 3*x(2);
   3*x(1)-7*x(2)];
end
```

SYSTEM 2 OUTPUT:

```
>> system2_eq


                             Norm of      First-order   Trust-region
 Iteration  Func-count     f(x)         step        optimality    radius
     0          3            20                          34            1
     1          6      0.0127667            1        0.0254            1
     2          9    4.46816e-32      0.465667      1.61e-15          2.5

Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>



x =

  1.0e-16 *

       0    0.2776
```

System 3: Matrix A = [4 -3 ; 3 4]

Recall the system is: dx/dt = 4x - 3y dy/dt = 3x + 4y

SYSTEM 3 CODE:

```
function system3_equilibrium
% x(1) = X, x(2) = Y
% x0 is initial condition
x0 = [1 1];
options=optimset('Display','iter');
x = fsolve(@system3_equilibrium,x0,options)
end

function F = system3_equilibrium(x)
% Rewrite the equation in the form F(x) = 0 %
 F = [4*x(1)- 3*x(2);
   3*x(1)+4*x(2)];
end
```

SYSTEM 3 OUTPUT:

```
>> system3_eq

                              Norm of     First-order   Trust-region
 Iteration  Func-count    f(x)        step        optimality    radius
     0          3            50                        25            1
     1          6        4.28932            1          7.32            1
     2          9             0      0.414214            0           2.5

Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>


x =

     0     0

>>
```

__**System 4: Matrix A =[4 -3 ; -4 3]**__
Recall that system is: dx/dt = 4x - 3y dy/dt = -4x + 3y

__SYSTEM 4 CODE:__

```
function system4_equilibrium
% x(1) = X, x(2) = Y
% x0 is a initial condition
x0 = [1 1];                    % Make a starting guess at the solution
options=optimset('Display','iter');   % Option to display output
x = fsolve(@system4_equilibrium,x0,options)        % Call solver
end

function F = system4_equilibrium(x)
% Rewrite the equation in the form F(x) = 0%
 F = [4*x(1)- 3*x(2);
   -4*x(1)+3*x(2)];
end
```

__SYSTEM 4 OUTPUT:__

```
>> system4_eq

                                Norm of      First-order   Trust-region
 Iteration  Func-count     f(x)          step         optimality    radius
     0          3             2                          8             1
     1          6      3.9443e-31          0.2       3.55e-15             1

Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>



x =

    0.8400    1.1200

>>
```

III. Find all eigenvectors and eigenvalues: Null function in MATLAB.

System 1: input and output: *from Matlab command line*	System 2: input and output: *from Matlab command line*	System 3: input and output: *from Matlab command line*	System 1: input and output: *from Matlab command line*
>> A1 = [4 2; 3 -1]	>> A2 = [1 -3; 3 7]	>> A3 = [4 -3; 3 4]	>> A4 = [4 -3; -4 3]
A1 =	A2 =	A3 =	A4 =
4 2	1 -3	4 -3	4 -3
3 -1	3 7	3 4	-4 3
>> eig(A1)	**>> eig(A2)**	**>> eig(A3)**	**>> eig(A4)**
ans =	ans =	ans =	ans =
5	4.0000 + 0.0000i	4.0000 + 3.0000i	7
-2	4.0000 - 0.0000i	4.0000 - 3.0000i	0
>> [V D] = eig(A1)	**>> [V D] = eig(A2)**	**>> [V D] = eig(A3)**	**>> [V D] = eig(A4)**
V =	V =	V =	V =
0.8944 -0.3162	-0.7071 + 0.0000i	0.7071 + 0.0000i	0.7071 0.6000
0.4472 0.9487	-0.7071 - 0.0000i	0.7071 + 0.0000i	-0.7071 0.8000
	0.7071 + 0.0000i	0.0000 - 0.7071i	
	0.7071 + 0.0000i	0.0000 + 0.7071i	
D =	D =	D =	D =
5 0	4.0000 + 0.0000i	4.0000 + 3.0000i	7 0
0 -2	0.0000 + 0.0000i	0.0000 + 0.0000i	0 0
	0.0000 + 0.0000i	0.0000 + 0.0000i	
	4.0000 - 0.0000i	4.0000 - 3.0000i	

IV. General Solution. For some reason I couldn't find an overall general solution within a reasonable amount of time, as both the "ode" and "ode45" as well as the "dsolve" functions within MATLAB seem to break systems up into general solutions for each "separated" variable. Wolfram Alpha and many resources showed ways to program, in MATLAB, a *single* ODE, but programming in the general solution for a system of ODE's is much more challenging. The following equations do apply, however:

<u>SYSTEM 1:</u>
<u>CODE:</u>

```
syms x(t) y(t)
Y = dsolve(diff(x) == 4*x+2*y, diff(y) == 3*x-y)
x(t) = Y.x
y(t) = Y.y
```

<u>OUTPUT:</u>

```
>> general_solution_system1
Y =
y: [1x1 sym]
x: [1x1 sym]

x(t) =
-(exp(-2*t)*(C1 - 6*C2*exp(7*t)))/3
y(t) =
exp(-2*t)*(C1 + C2*exp(7*t))
>>
```

<u>SYSTEM 2:</u>
<u>CODE:</u>

```
syms x(t) y(t)
Y = dsolve(diff(x) == x-3*y, diff(y) == 3*x+7*y)
x(t) = Y.x
y(t) = Y.y
```

<u>OUTPUT:</u>

```
>> general_solution_system2
Y =
   y: [1x1 sym]
   x: [1x1 sym]
x(t) =
(C18*exp(4*t))/3 - C17*exp(4*t) - C18*t*exp(4*t)
```

```
y(t) =
C17*exp(4*t) + C18*t*exp(4*t)
>>
```

<u>SYSTEM 3:</u>

<u>CODE:</u>

```
syms x(t) y(t)
Y = dsolve(diff(x) == 4*x-3*y, diff(y) == 3*x+4*y)
x(t) = Y.x
y(t) = Y.y
```

<u>OUTPUT:</u>

```
>> general_solution_system3
Y =
y: [1x1 sym]
x: [1x1 sym]

x(t) =
- C20*cos(3*t)*exp(4*t) - C19*sin(3*t)*exp(4*t)

y(t) =
C19*cos(3*t)*exp(4*t) - C20*sin(3*t)*exp(4*t)
>>
```

<u>SYSTEM 4:</u>

<u>CODE:</u>

```
syms x(t) y(t)
Y = dsolve(diff(x) == 4*x-3*y, diff(y) == -4*x+3*y)
x(t) = Y.x
y(t) = Y.y
```

<u>OUTPUT:</u>

```
>> general_solution_system4
Y =
y: [1x1 sym]
x: [1x1 sym]
x(t) =
(3*C22)/4 - C21*exp(7*t)
y(t) =
C22 + C21*exp(7*t)
>>
```