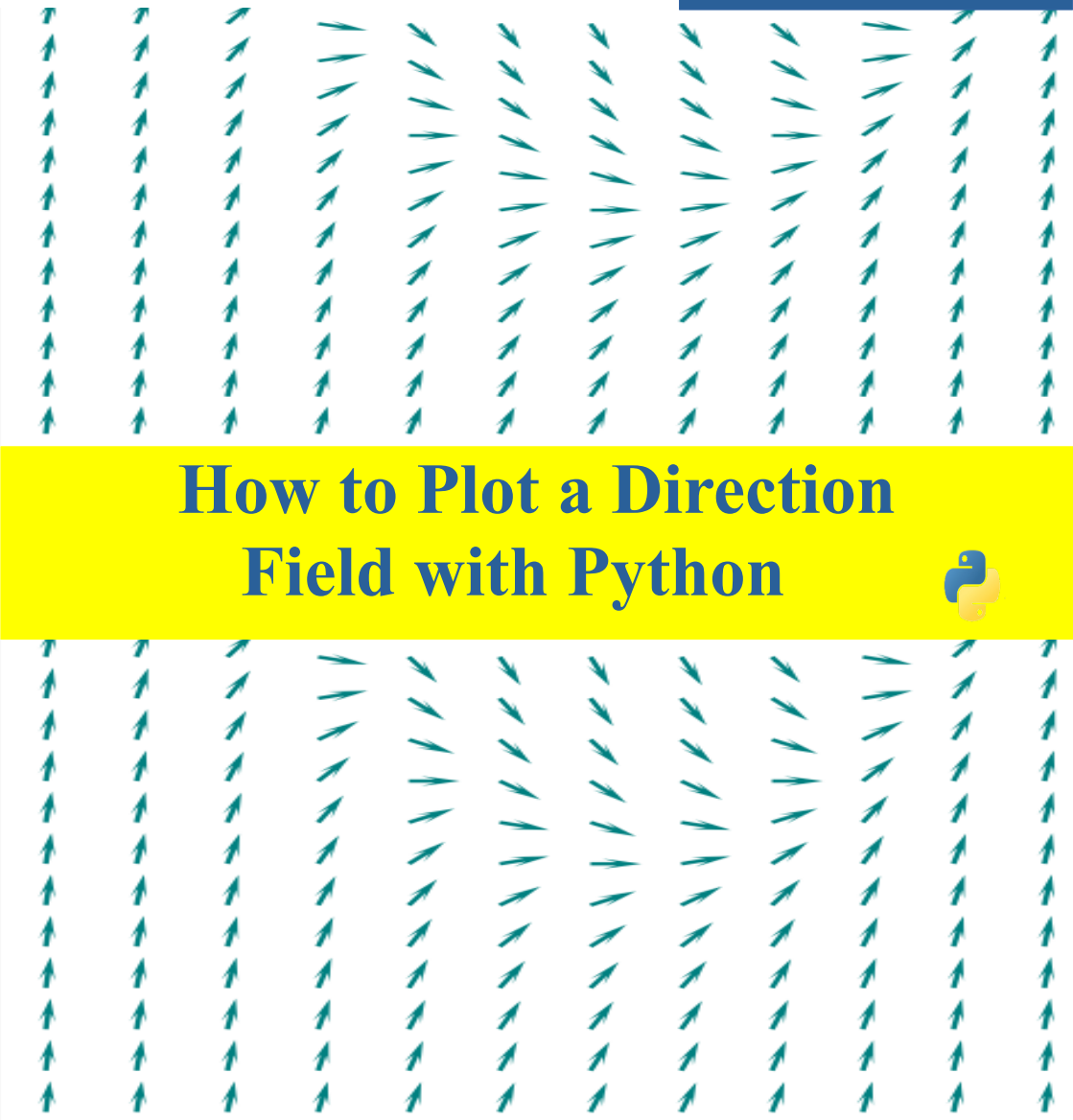
A quick guide...



How to Plot a Direction Field with Python

Oluwatosin E. Odubanjo

How to Plot a Direction Field with Python

Oluwatosin E. Odubanjo

About this material

This material is a quick guide on how to use Python to plot a direction field. The examples I have used are examples found in Howard's material and Boyce and DiPrima's textbook. I have also included useful links, and my python scripts used in generating the plots.

Thank you.

Table of Contents

1

Direction Field

A Direction Field (DF) is also known as a Slope field, it is a graphical representation of the solutions of differential equations (DEs) of the form:

$$\frac{dy}{dx} = f(x, y)$$

f is a function of two variables, x and y.

1.1 Why do we need to plot a DF

To understand the behaviour of a DE and its solutions.

1.2 Behaviour of a DE

Usually, we want to explore a differential equation to understand the interactions of the variables, that is, the relationships that exist between the variables. Understanding these relationships / interactions is what helps in establishing past, present and future conclusions. Also, the solutions of a DE are a consequence of these relationships / interactions.

2

Constructing a Direction Field

The construction of a Direction Field (DF) is usually the first step in exploring a Differential Equation (DE).

The construction of a DF of a DE does not require that we solve the DE first. We only just have to repeatedly evaluate the given function over a grid of points, which are numerical intervals. Furthermore, DF's can be constructed for even the most challenging DE's.

Finally, you need a computer and a suitable application software to repeatedly evaluate a given function over some numerical interval.

2.1 How to construct a DF

1. Draw a rectangular grid of a few hundred points;

2. Evaluate the given function, (DE), at each point: At each point draw a short line segment whose slope is equal to the value of the function at that point. Each line segment drawn is the tangent to the graph of the solution passing through that point.

3

Creating a Numerical Interval Using numpy.arange()

3.1 numpy.arange()

The numpy.arange() function is used to generate an array of a sequence of numbers of the type integer, float.

3.2 Python code – numpy.arange()

```
import numpy as np

a = np.arange(0, 5)

b = np.arange(0, 5, 2)

c = np.arange(0, 5, 0.5)

print(a)

print(b)

print(c)


>> [0 1 2 3 4]

>> [0 2 4]

>> [0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5]
```

see the syntax for the numpy.arange() function here: https://numpy.org/doc/stable/reference/generated/numpy.arange.html

4

Drawing a Rectangular Grid of a Few Hundred Points Using numpy.meshgrid()

4.1 numpy.meshgrid()

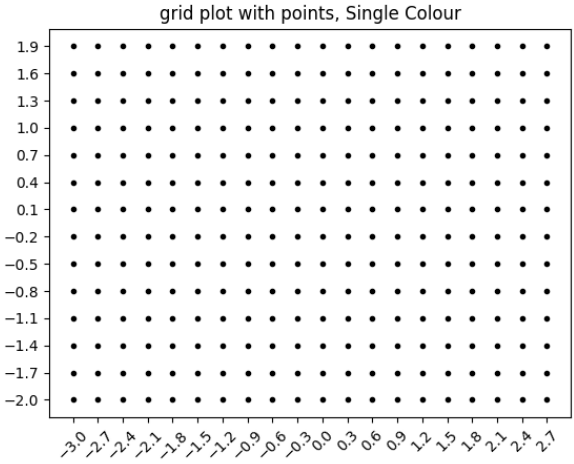
The numpy.meshgrid() function is used to create a rectangular grid out of two declared one-dimensional arrays.

4.2 Python code – numpy.meshgrid()

```
import numpy as np
import matplotlib.pyplot as plt

nx, ny = .3, .3
x = np.arange(-3, 3, nx)
y = np.arange(-2, 2, ny)
X, Y = np.meshgrid(x, y)

plt.plot(X, Y, marker=".", color='k', linestyle='none')
plt.xticks(x, rotation='45')
plt.yticks(y)
plt.title('grid plot with points, Single Colour')
plt.show()
```



see the syntax for the numpy.meshgrid() function here:

https://numpy.org/doc/stable/reference/generated/numpy.meshgrid.html

5

Evaluating a Differential Equation and constructing its Differential Field using matplotlib.pyplot.quiver()

5.1 matplotlib.pyplot.quiver()

A quiver plot is a type of 2-D plot that is made up of vector lines, these vector lines are in shape of arrows to indicate the direction of the vectors (the lines can however be drawn without arrows). These plots are mainly used to visualize slopes / gradients.

The matplotlibrary quiver() function is used to visualize a vector field over a grid of points which are numerical intervals; the vector at a point represents the magnitude of the field vector at that point.

see the syntax for the matplotlib.pyplot.quiver() function here:

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.quiver.html

6

Exercise 1: y’ (x) = x + sin(y)

6.1 Direction Field for y’ (x) = x + sin(y)

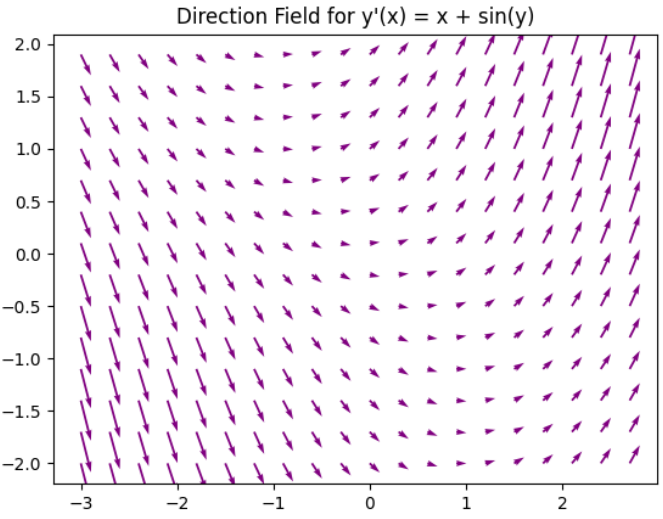
```
import numpy as np
import matplotlib.pyplot as plt

nx, ny = .3, .3
x = np.arange(-3, 3, nx)
y = np.arange(-2, 2, ny)

# rectangular grid with points
X, Y = np.meshgrid(x, y)

# derivative
dy = X + np.sin(Y)
dx = np.ones(dy.shape)

# plot
plt.quiver(X, Y, dx, dy, color='purple')
plt.title('Direction Field for y\'(x) = x + sin(y)')
plt.show()
```



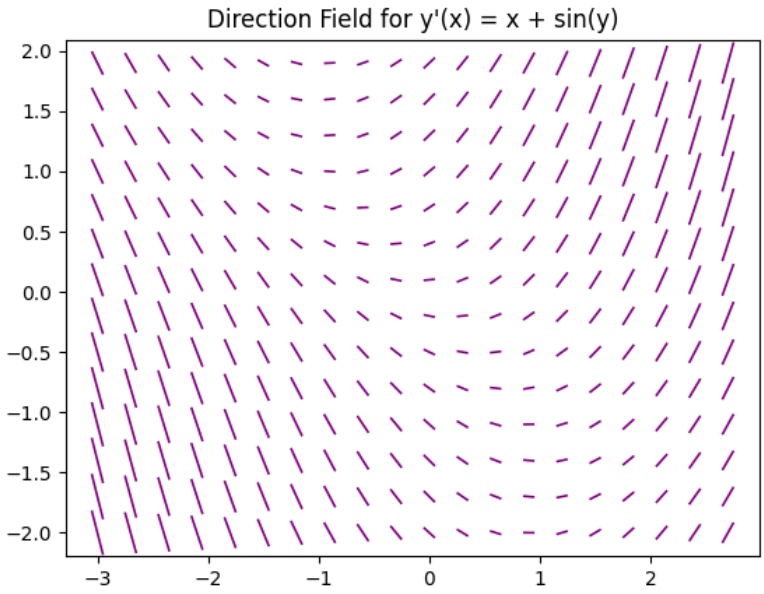
Without arrows:

```
nx, ny = .3, .3
x = np.arange(-3, 3, nx)
y = np.arange(-2, 2, ny)

# rectangular grid with points
X, Y = np.meshgrid(x, y)

# derivative
dy = X + np.sin(Y)
dx = np.ones(dy.shape)

# plot
plt.quiver(X, Y, dx, dy, color='purple', headaxislength=2, headlength=0, pivot='middle',
scale=10, linewidth=.2, units='xy', width = .02, headwidth=1)
plt.title('Direction Field for y\'(x) = x + sin(y)')
plt.show()
```



6.2 Normalization

If we are just particular about the direction, we can make the lines with and without arrows of unit length – this is what is called normalization.

6.3 Normalized Direction Field for y' (x) = x + sin(y)

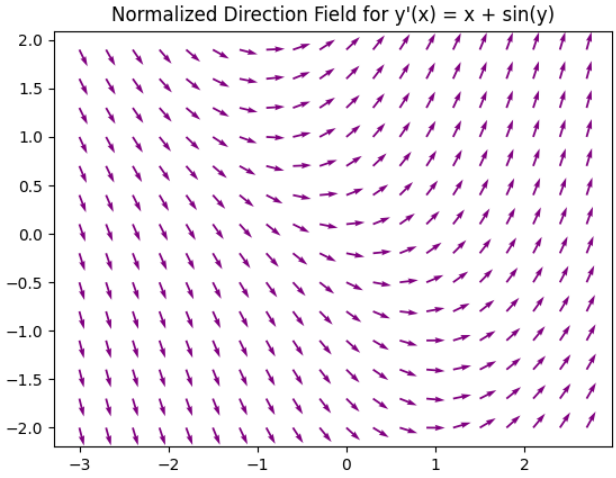
```
import numpy as np
import matplotlib.pyplot as plt

nx, ny = .3, .3
x = np.arange(-3, 3, nx)
y = np.arange(-2, 2, ny)
X, Y = np.meshgrid(x, y)

dy = X + np.sin(Y)
dx = np.ones(dy.shape)

# normalize arrows
dyu = dy /np.sqrt(dx**2 + dy**2)
dxu = dx /np.sqrt(dx**2 + dy**2)

# plot
plt.quiver(X, Y, dxu, dyu, color='purple')
plt.title('Normalized Direction Field for y\'(x) = x + sin(y)')
plt.show()
```



without arrows:

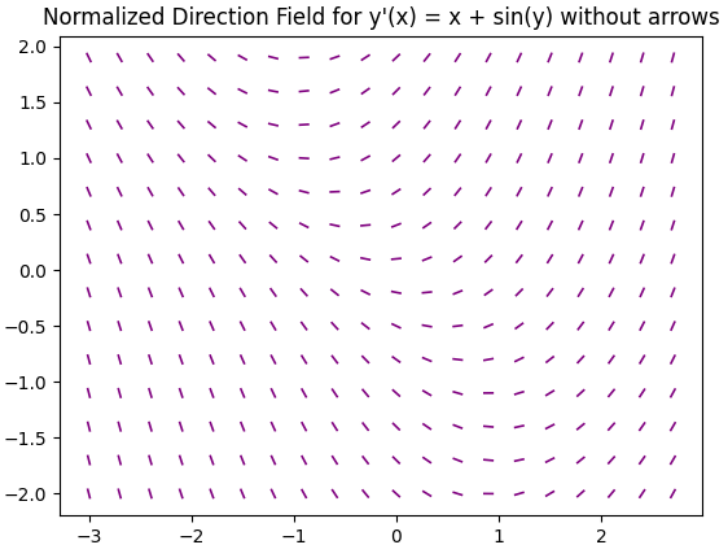
```
nx, ny = .3, .3
x = np.arange(-3, 3, nx)
y = np.arange(-2, 2, ny)

# rectangular grid with points
X, Y = np.meshgrid(x, y)

# derivative
dy = X + np.sin(Y)
dx = np.ones(dy.shape)

# normalize arrows
dyu = dy /np.sqrt(dx**2 + dy**2)
dxu = dx /np.sqrt(dx**2 + dy**2)

# plot
plt.quiver(X, Y, dxu, dyu, color='purple', headaxislength=2, headlength=0, pivot='middle',
scale=10, linewidth=.2, units='xy', width = .02, headwidth=1)
plt.title('Normalized Direction Field for y\'(x) = x + sin(y) without arrows')
plt.show()
```



7

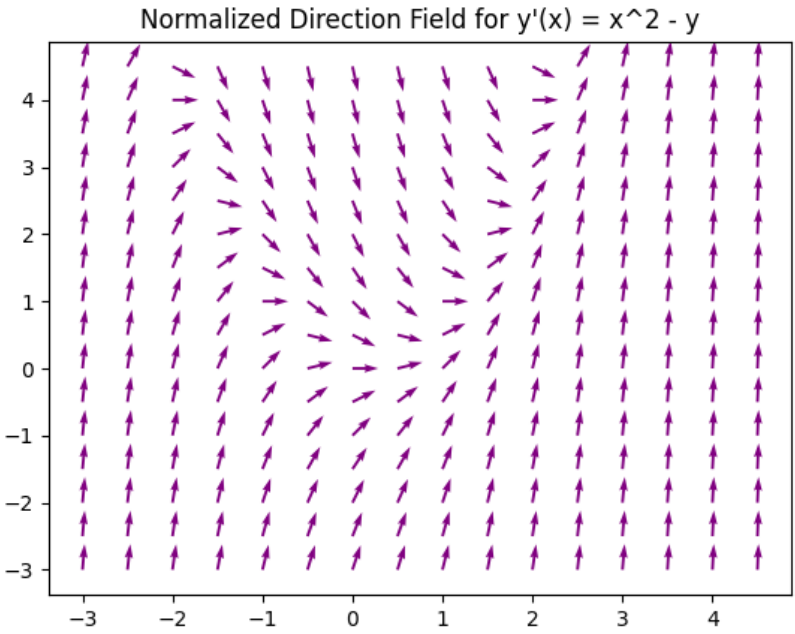
Exercise 2: y’ (x) = $x^2 – y$

7.1 Normalized Direction Field for y’ (x) = $x^2 – y$

```
nx, ny = .5, .5
x = np.arange(-3, 5, nx)
y = np.arange(-3, 5, ny)
X, Y = np.meshgrid(x, y)

dy = X**2 - Y
dx = np.ones(dy.shape)
dyu = dy /np.sqrt(dx**2 + dy**2)
dxu = dx /np.sqrt(dx**2 + dy**2)

plt.quiver(X, Y, dxu, dyu, color='purple')
plt.title('Normalized Direction Field for y\'(x) = x^2 - y')
plt.show()
```



8

Exercise 3: A Falling Object

8.1 DE of a falling object

The DE that describes the motion of a falling object in the atmosphere near sea level is given as:

$$m\frac{dv}{dt} = mg - c_d v \qquad (8.1)$$

where:

m = mass of the object in kilogram (kg)

dv/dt = acceleration of the object in meters/second^2 (m/s^2)

g = acceleration due to gravity approximately equal to 9.8m/s^2

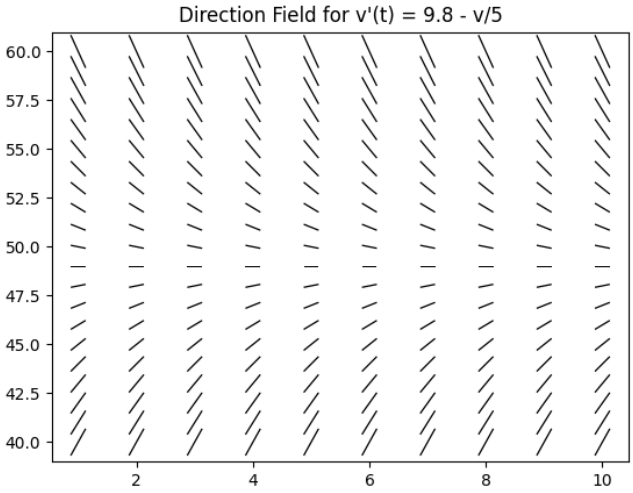
c_d = Drag coefficient (constant value which varies from one object to another)

v = velocity of the object in meter/second (m/s)

To solve the equation (8.1) we need to find a function v = v(t) that satisfies the equation. Our concern is to however show how to plot the direction field, to do this, we can assign values to *m and* C_d. Using 10kg and 2kg/second respectively, this would simplify the equation to:

$$\frac{dv}{dt} = 9.8 - \frac{v}{5} \qquad (8.2)$$

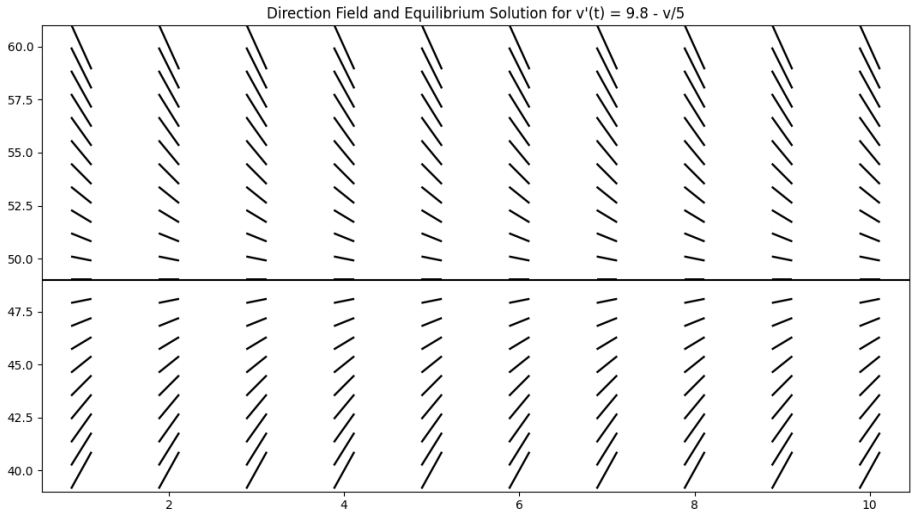
We can now plot the direction field for equation (8.2):



8.2 Equilibrium solution

The equilibrium solution of a DE is also known as the balance solution of the DE. It is obtained when we make the left hand side of the DE equal to zero and then solve for the dependent variable. At the equilibrium solution, the DE is constant with time. For equation (8.2), v(t) = 49, is the equilibrium solution. In the graph above, we see that at value v(t) = 49, the DE does not change with time.

Now, we plot the equilibrium solution superimposed on the direction field:



8.3 Python code: falling object

```
import numpy as np

import matplotlib.pyplot as plt


nt, nv = 1, 1

t = np.arange(1, 11, nt)

v = np.arange(40, 61, nv)


# rectangular grid with points

T, V = np.meshgrid(t, v)
```

```
# derivative
dv = 9.8 - (V / 5)
dt = np.ones(dv.shape)


def direction_field():
    plt.quiver(T, V, dt, dv, color='black', headaxislength=5, headlength=0,
               pivot='middle', scale=2, linewidth=.2, units='xy', width = .05,
               headwidth=1)
    plt.title('Direction Field for v\'(t) = 9.8 - v/5')
    plt.show()


def direction_equilibrium():
    zero_index = np.where(dv == 0) # get index of zero elements
    equilibrium_solution = V[zero_index] # get the values of V which makes
                                         #dv == 0


    # Plot equilibrium position
    plt.axhline(y=equilibrium_solution[0], color='black', linestyle='-')


    # Plot direction field
    plt.quiver(T, V, dt, dv, color='black', headaxislength=5, headlength=0,
               pivot='middle', scale=2, linewidth=.2, units='xy', width = .05,
              headwidth=1)
```

```
plt.title('Direction Field and Equilibrium Solution for v\'(t) = 9.8 - v/5')
    plt.show()


direction_field()
direction_equilibrium()
```

9

Download Python Scripts

The Python scripts used in generating the direction fields used in this material can be downloaded from github using the following steps:

1. Clone repository:

$ git clone https://github.com/olutosinbanjo/plots.git

$ change directory to direction_field folder

OR

2. Download the zip file

$ Go to https://github.com/olutosinbanjo/plots.git

$ Download zip

$ unzip file

$ change directory to direction_field folder

References

[1] W. E Boyce and R. C DiPrima, *Elementary Differential Equations and Boundary Value Problems*, eigth edition. USA: John Wiley & Sons, Inc. 2005, pp 1- 5.

[2] P. Howard, *Ordinary Differential Equations in MATLAB*. 2013.