Arange and Plotting
Numpy and Matplotlib Basics 1

Dr. Ryan Krauss

Grand Valley State University

pdm

Learning Outcomes

- use the `arange` function to create time vectors

pdm

Learning Outcomes

- use the `arange` function to create time vectors
- create figures, plot data, add labels and legends, and save figures

pdm

Arange

- function from `numpy`

pdm

Arange

- function from `numpy`
- create an array over a range

pdm

Arange

- function from `numpy`
- create an array over a range
- usage:

pdm

Arange

- function from `numpy`
- create an array over a range
- usage:
  - `vect = arange(start, stop, step)`

pdm

Arange

- function from `numpy`
- create an array over a range
- usage:
  - `vect = arange(start, stop, step)`
  - `t = arange(0,1,0.01)`

pdm

Arange

- function from `numpy`
- create an array over a range
- usage:
  - `vect = arange(start, stop, step)`
  - `t = arange(0,1,0.01)`
  - `t = np.arange(0,1,0.01)`

pdm

Sin, Cos, and Pi

- I assume these `numpy` functions and variables are obvious

pdm

Sin, Cos, and Pi

- I assume these `numpy` functions and variables are obvious
  - `np.sin`

pdm

Sin, Cos, and Pi

- I assume these `numpy` functions and variables are obvious
  - `np.sin`
  - `np.cos`

pdm

Sin, Cos, and Pi

- I assume these `numpy` functions and variables are obvious
  - `np.sin`
  - `np.cos`
  - `np.pi`

pdm

Plotting Functions

From `matplotlib.pyplot`:

- `figure`
- `clf`
- `plot`
- `xlabel`
- `ylabel`
- `legend`
- `savefig`

pdm

figure

- create or activate a figure:

pdm

figure

- create or activate a figure:
  - `figure(1)`

pdm

figure

- create or activate a figure:
  - `figure(1)`
- activate if that figure number already exists; otherwise create a new figure

pdm

figure

- create or activate a figure:
  - `figure(1)`
- activate if that figure number already exists; otherwise create a new figure
- create as many figures as you want/need

pdm

figure

- create or activate a figure:
  - `figure(1)`
- activate if that figure number already exists; otherwise create a new figure
- create as many figures as you want/need
- the active figure is the one that gets drawn on by subsequent `plot` commands

pdm

clf

- clear the current figure

pdm

clf

- clear the current figure
- usage:

pdm

clf

- clear the current figure
- usage:
  - `clf()`

pdm

clf

- clear the current figure
- usage:
  - `clf()`
- `matplotlib` turns "hold" on by default

pdm

clf

- clear the current figure
- usage:
  - `clf()`
- `matplotlib` turns "hold" on by default
- subsequent runs of your code will draw onto the same figure

pdm

clf

- clear the current figure
- usage:
  - `clf()`
- `matplotlib` turns "hold" on by default
- subsequent runs of your code will draw onto the same figure
  - use `clf` sort of as a replacement for `close all` (Matlab)

pdm

clf

- clear the current figure
- usage:
  - `clf()`
- `matplotlib` turns "hold" on by default
- subsequent runs of your code will draw onto the same figure
  - use `clf` sort of as a replacement for `close all` (Matlab)
  - you can use `close('all')`, but your code will run a little slower

pdm

plot

- `plot(x,y)`

pdm

plot

- `plot(x,y)`
- actually plot `y` vs. `x` on the current axis

pdm

plot

- `plot(x,y)`
- actually plot `y` vs. `x` on the current axis
- several different formats:

pdm

plot

- `plot(x,y)`
- actually plot `y` vs. `x` on the current axis
- several different formats:
  - `plot(x, y, 'r:')`

pdm

plot

- `plot(x,y)`
- actually plot `y` vs. `x` on the current axis
- several different formats:
  - `plot(x, y, 'r:')`
  - `plot(x, y, label='$y_1$')`

pdm

plot

- `plot(x,y)`
- actually plot `y` vs. `x` on the current axis
- several different formats:
  - `plot(x, y, 'r:')`
  - `plot(x, y, label='$y_1$')`
    - makes legend creation very easy, but can only handle one x/y pair per `plot` command

pdm

plot

- `plot(x,y)`
- actually plot `y` vs. `x` on the current axis
- several different formats:
  - `plot(x, y, 'r:')`
  - `plot(x, y, label='$y_1$')`
    - makes legend creation very easy, but can only handle one x/y pair per `plot` command
  - `plot(x1, y1, x2, y2)`

pdm

plot

- `plot(x,y)`
- actually plot `y` vs. `x` on the current axis
- several different formats:
  - `plot(x, y, 'r:')`
  - `plot(x, y, label='$y_1$')`
    - makes legend creation very easy, but can only handle one x/y pair per `plot` command
  - `plot(x1, y1, x2, y2)`
  - `plot(x1, y1, 'r-', x2, y2, 'g-.')`

pdm

plot

- `plot(x,y)`
- actually plot `y` vs. `x` on the current axis
- several different formats:
  - `plot(x, y, 'r:')`
  - `plot(x, y, label='$y_1$')`
    - makes legend creation very easy, but can only handle one x/y pair per `plot` command
  - `plot(x1, y1, x2, y2)`
  - `plot(x1, y1, 'r-', x2, y2, 'g-.')`
- keep in mind that hold is on by default

pdm

xlabel

- `xlabel("Time (sec.)")`

pdm

xlabel

- `xlabel("Time (sec.)")`
- add a string to the x-axis label

pdm

Scientific string note

- `matplotlib` supports sub-scripts, super-scripts, and symbols using LaTeX syntax with dollar signs `$`:

pdm

Scientific string note

- `matplotlib` supports sub-scripts, super-scripts, and symbols using LaTeX syntax with dollar signs `$`:
  - `$y_1$`

pdm

Scientific string note

- `matplotlib` supports sub-scripts, super-scripts, and symbols using LaTeX syntax with dollar signs `$`:
  - `$y_1$`
  - `$x^2$`

pdm

Scientific string note

- `matplotlib` supports sub-scripts, super-scripts, and symbols using LaTeX syntax with dollar signs `$`:
  - `$y_1$`
  - `$x^2$`
  - `$\\theta$`

pdm

Scientific string note

- `matplotlib` supports sub-scripts, super-scripts, and symbols using LaTeX syntax with dollar signs `$`:
  - `$y_1$`
  - `$x^2$`
  - `$\\theta$`
    - note that you have to escape the backslash

pdm

ylabel

- `ylabel("$y_1(t)$")`

pdm

ylabel

- `ylabel("$y_1(t)$")`
- add a string to the y-axis label

pdm

legend

- main usages:

pdm

legend

- main usages:
  - `legend([label1, label2])`

pdm

legend

- main usages:
  - `legend([label1, label2])`
  - `legend([label1, label2], loc=2)`

pdm

legend

- main usages:
  - `legend([label1, label2])`
  - `legend([label1, label2], loc=2)`
- if you labeled all your plots:

pdm

legend

- main usages:
  - `legend([label1, label2])`
  - `legend([label1, label2], loc=2)`
- if you labeled all your plots:
  - `legend()`

pdm

legend

- main usages:
  - `legend([label1, label2])`
  - `legend([label1, label2], loc=2)`
- if you labeled all your plots:
  - `legend()`
  - `legend(loc=2)`

pdm

savefig

- save the current figure to a file

pdm

savefig

- save the current figure to a file
- optionally specify dpi if saving to png

pdm

savefig

- save the current figure to a file
- optionally specify dpi if saving to png
- `savefig("fig1.png", dpi=300)`

pdm

savefig

- save the current figure to a file
- optionally specify dpi if saving to png
- `savefig("fig1.png", dpi=300)`
- `savefig("fig1.eps")`

pdm

savefig

- save the current figure to a file
- optionally specify dpi if saving to png
- `savefig("fig1.png", dpi=300)`
- `savefig("fig1.eps")`
- I mainly use png for websites and eps for documents

pdm

savefig

- save the current figure to a file
- optionally specify dpi if saving to png
- `savefig("fig1.png", dpi=300)`
- `savefig("fig1.eps")`
- I mainly use png for websites and eps for documents
  - I convert eps to pdf using a shell script

pdm

savefig

- save the current figure to a file
- optionally specify dpi if saving to png
- `savefig("fig1.png", dpi=300)`
- `savefig("fig1.eps")`
- I mainly use png for websites and eps for documents
  - I convert eps to pdf using a shell script
- direct pdf support may have improved over the years

pdm

savefig

- save the current figure to a file
- optionally specify dpi if saving to png
- `savefig("fig1.png", dpi=300)`
- `savefig("fig1.eps")`
- I mainly use png for websites and eps for documents
  - I convert eps to pdf using a shell script
- direct pdf support may have improved over the years
- from the help: "Most backends support png, pdf, ps, eps and svg"

pdm

savefig

- save the current figure to a file
- optionally specify dpi if saving to png
- `savefig("fig1.png", dpi=300)`
- `savefig("fig1.eps")`
- I mainly use png for websites and eps for documents
  - I convert eps to pdf using a shell script
- direct pdf support may have improved over the years
- from the help: "Most backends support png, pdf, ps, eps and svg"
  - I haven't played with svg

pdm

savefig

- save the current figure to a file
- optionally specify dpi if saving to png
- `savefig("fig1.png", dpi=300)`
- `savefig("fig1.eps")`
- I mainly use png for websites and eps for documents
  - I convert eps to pdf using a shell script
- direct pdf support may have improved over the years
- from the help: "Most backends support png, pdf, ps, eps and svg"
  - I haven't played with svg
- png is easy to stick in a Word document, but pixelation usually leads to poor print quality

pdm

Plotting example

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


t = np.arange(0,1,0.01)
y = np.sin(2*np.pi*t)

plt.figure(1)
plt.clf()
plt.plot(t,y)

plt.xlabel('Time (sec.)')
plt.ylabel('y(t)')

plt.show()
```

pdm