Lab 0 - Python and Jupyter notebook introduction

Maxwell A. Fine

1004714400

In [104…
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

Warm-up Exercises

Try the following commands on your jupyter notebook or python editor and see what output they produce.

In [105…
```
a = 1 + 5
b = 2
c = a + b
print(a / b)
print(a // b)
print(a - b)
print(a * b)
print(a**b)
```

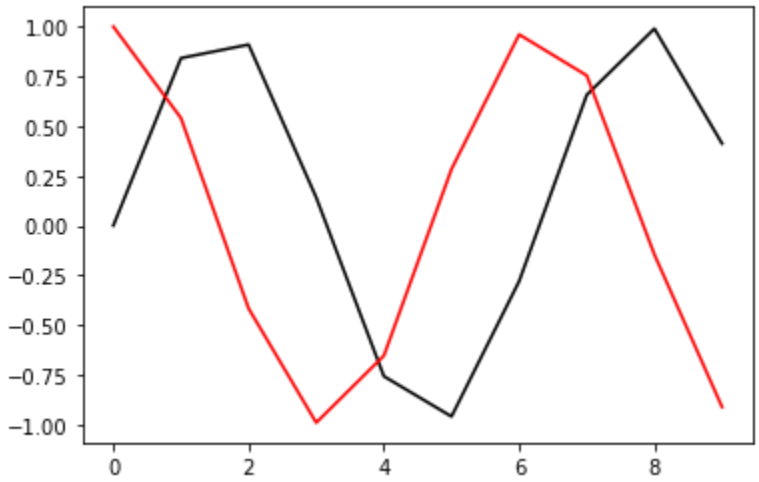
```
3.0
3
4
12
36
```

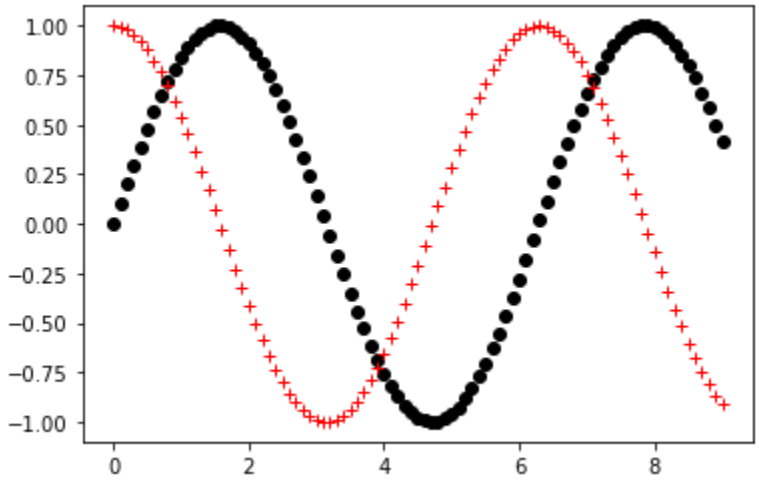
In [106…
```
a = np.array([[3, 1],
              [1, 3]])
b = np.array([[3],
              [5]])
print(a * b)
print(np.dot(a, b))
print(np.dot(b.T, a))
c = a**(-1.0)
print(c * a)
```

```
[[ 9  3]
 [ 5 15]]
[[14]
 [18]]
[[14 18]]
[[1. 1.]
 [1. 1.]]
```

In [107…
```
t = np.arange(10)
g = np.sin(t)
h = np.cos(t)
plt.figure()
plt.plot(t, g, 'k', t, h, 'r');

t = np.arange(0, 9.1, 0.1)
g = np.sin(t)
h = np.cos(t)
```

```
plt.figure()
plt.plot(t, g, 'ok', t, h, '+r');
```



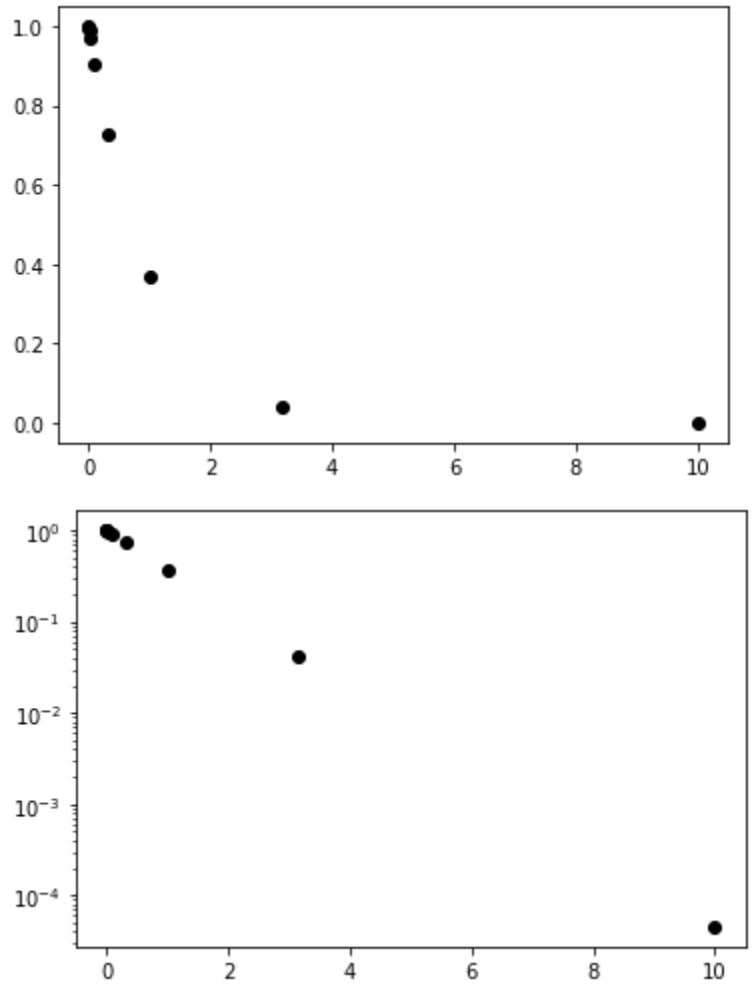


In [108…

```
t = np.linspace(0, 10, 20)
print(t)
t = np.logspace(0.001, 10, 9)
print(t)
t = np.logspace(-3, 1, 9)
print(t)
y = np.exp(-t)

plt.figure()
plt.plot(t, y, 'ok')
plt.figure()
plt.semilogy(t, y, 'ok')
```

```
[ 0.          0.52631579  1.05263158  1.57894737  2.10526316  2.63157895
  3.15789474  3.68421053  4.21052632  4.73684211  5.26315789  5.78947368
  6.31578947  6.84210526  7.36842105  7.89473684  8.42105263  8.94736842
  9.47368421 10.        ]
[1.00230524e+00 1.78186583e+01 3.16774344e+02 5.63151182e+03
 1.00115196e+05 1.77981556e+06 3.16409854e+07 5.62503203e+08
 1.00000000e+10]
[1.00000000e-03 3.16227766e-03 1.00000000e-02 3.16227766e-02
 1.00000000e-01 3.16227766e-01 1.00000000e+00 3.16227766e+00
 1.00000000e+01]
```

Out[108]: `[<matplotlib.lines.Line2D at 0x7f51effd5b80>]`



Integration Function

Here is a more complicated function that computes the integral $y(x)$ with interval dx:

$$c = \int y(x)dx \sim \sum_{i=1}^{N} y_i dx_i.$$

It can deal with both cases of even and uneven sampling.

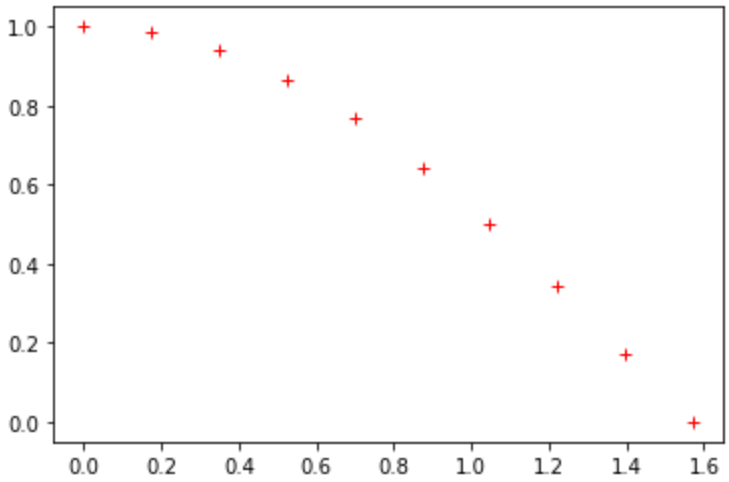
In [109…

```
def integral(y, dx):
    # function c = integral(y, dx)
    # To numerically calculate integral of vector y with interval dx:
    # c = integral[ y(x) dx]
    # ------ This is a demonstration program ------
    n = len(y) # Get the length of vector y
    nx = len(dx) if np.iterable(dx) else 1
    c = 0 # initialize c because we are going to use it
    # dx is a scalar <=> x is equally spaced
    if nx == 1: # '==', equal to, as a condition
        for k in range(1, n):
            c = c + (y[k] + y[k-1]) * dx / 2
    # x is not equally spaced, then length of dx has to be n-1
    elif nx == n-1:
        for k in range(1, n):
            c = c + (y[k] + y[k-1]) * dx[k-1] / 2
    # If nx is not 1 or n-1, display an error messege and terminate program
    else:
        print('Lengths of y and dx do not match!')
    return c
```

Use this function (`integral`) to compute $\int_0^{\pi/2} \cos(t)dt$ with an evenly sampled time series.

In [110…
```
# number of samples
nt = 10
# generate time vector
t = np.linspace(0, np.pi/2, nt)
# compute sample interval (evenly sampled, only one number)
dt = t[1] - t[0]
y = np.cos(t)
plt.plot(t, y, 'r+')
c = integral(y, dt)
print(c)
```

```
0.9974602317917262
```



Part 1

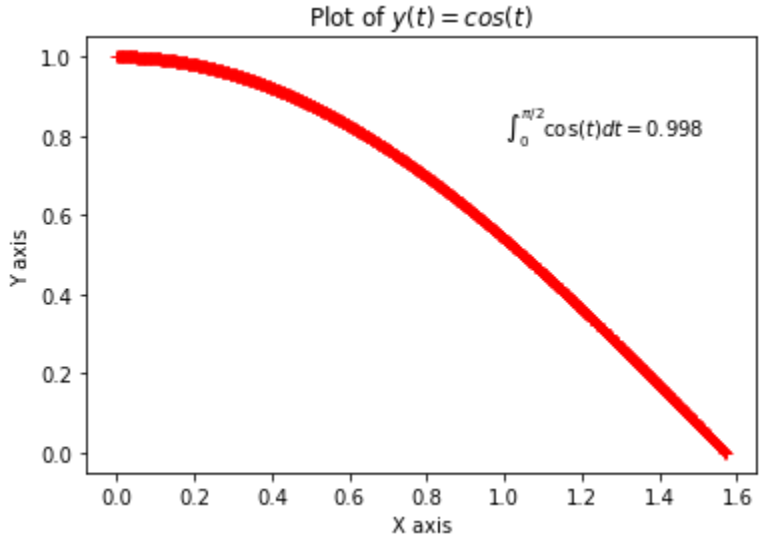
First plot $y(t)$. Is the output c value what you are expecting for $\int_0^{\pi/2} \cos(t)dt$? How can you improve the accuracy of your computation?

Answer

The analytical value of c for this problem is 1. We can improve the accuracy of the computation by using more samples and, or by using a more sophisticated numerical integration method.

In [111…
```
t = np.linspace(0, np.pi/2, 1000)
dt = (t[-1]-t[0]) / len(t)
y = np.cos(t)
plt.plot(t, y, 'r+')
c = integral(y, dt)
plt.title('Plot of $y(t) = cos(t)$')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.text(x =1, y=0.8, s='$\int_0^{\pi/2} \cos(t) dt =$' + str(c)[0:5])
print(c)
print(dt)
plt.savefig('phy408_lab00_q1.pdf')
```

```
0.9989997941774101
0.0015707963267948967
```



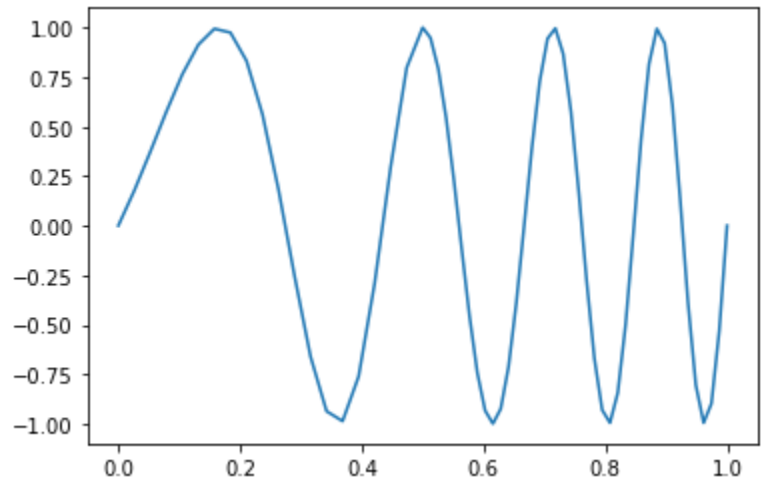
Part 2

For an unevenly spaced time series that depicts $\sin[(2\pi(t+3t^2)]$ (so-called chirp function), compute $\int_0^1 \sin[(2\pi(t+3t^2)]dt$.

In [112…

```
nt = 20
# sampling between [0,0.5]
t1 = np.linspace(0, 0.5, nt)
# double sampling between [0.5,1]
t2 = np.linspace(0.5, 1, 2*nt)
# concatenate time vector
t = np.concatenate((t1[:-1], t2))
# compute y values (f=2t)
y = np.sin(2 * np.pi * (t + 3*t**2))
plt.plot(t, y)
# compute sampling interval vector
dt = t[1:] - t[:-1]
c = integral(y, dt)
print(c)
```

```
0.08673999688870114
```

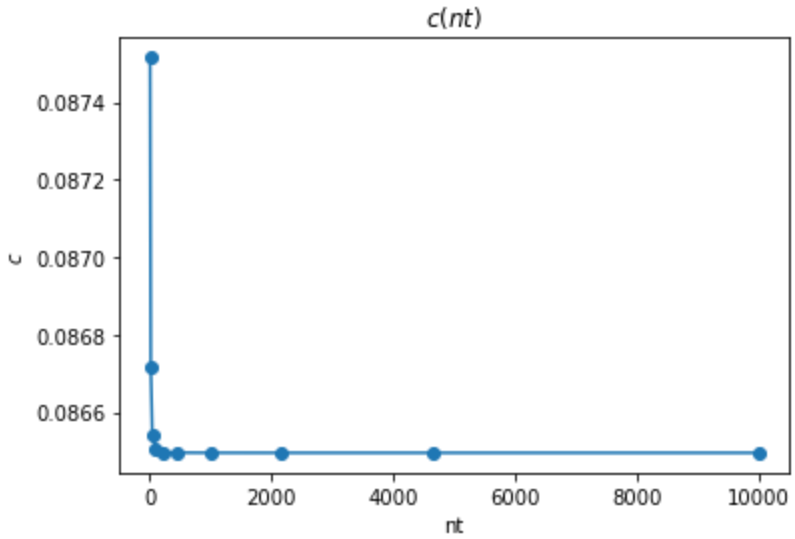


Show your plot of $y(t)$ for $nt = 50$. Try different nt values and see how the integral results change. Write a `for` loop around the statements above to try a series of nt values (e.g, 50, 100, 500, 1000, 5000) and generate a plot of $c(nt)$. What value does c converge to after using larger and larger nt? (Please include your modified Python code.)

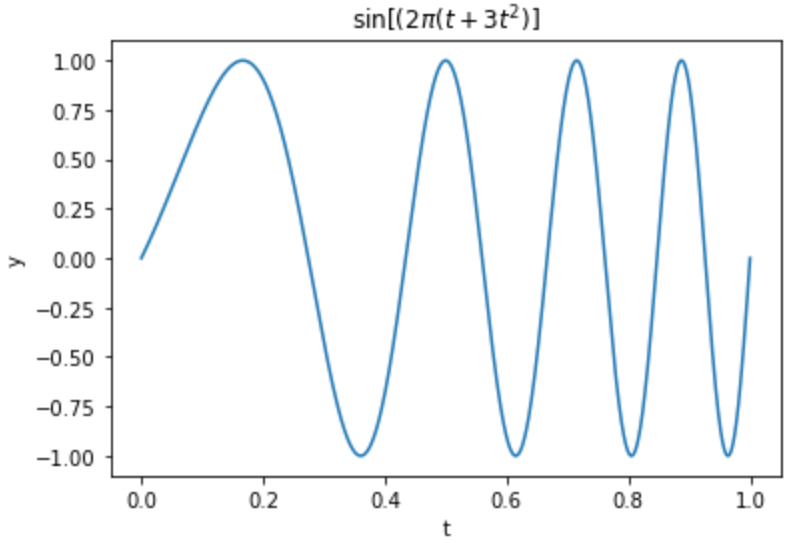
Answer

In [113…

```
nt_list = np.logspace(1,4, 10)
c_list = []

for i in range(len(nt_list)):
    nt = int(nt_list[i])
    # sampling between [0,0.5]
    t1 = np.linspace(0, 0.5, nt)
    # double sampling between [0.5,1]
    t2 = np.linspace(0.5, 1, 2*nt)
    # concatenate time vector
    t = np.concatenate((t1[:-1], t2))
    # compute y values (f=2t)
    y = np.sin(2 * np.pi * (t + 3*t**2))
    # compute sampling interval vector
    dt = t[1:] - t[:-1]
    c = integral(y, dt)
    c_list.append(c)
   # print(c)

plt.plot(nt_list, c_list)
plt.scatter(nt_list, c_list)
plt.title('$c(nt)$')
plt.xlabel('nt')
plt.ylabel('$c$')
plt.show()

print(' $c$ convergs to ~' + str(c_list[-1])[0:6])
plt.plot(t, y)
plt.title('$\sin[(2 \pi (t+ 3t^2)]$')
plt.xlabel('t')
plt.ylabel('y')
plt.show()
```




```
$c$ convergs to ~0.0864
```



Accuracy of Sampling

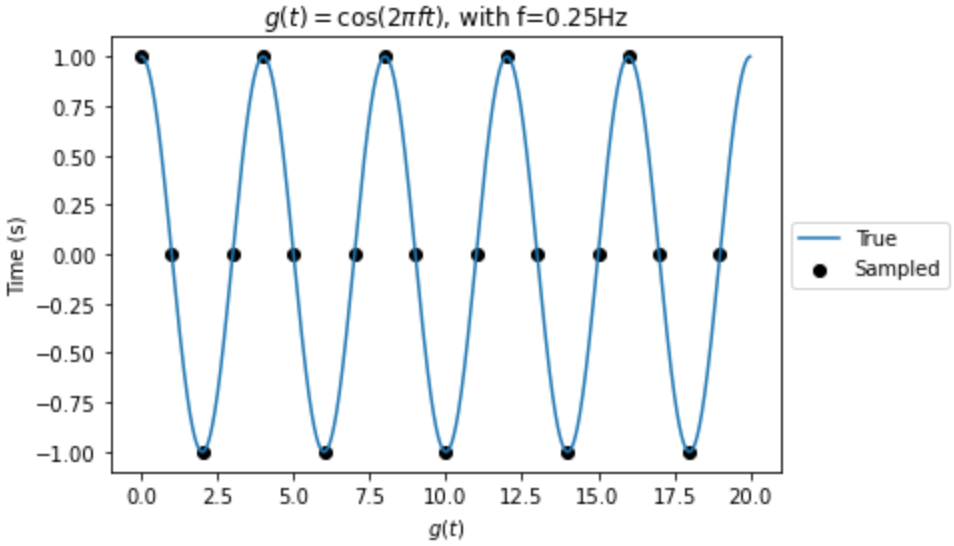
Let us sample the function $g(t) = \cos(2\pi ft)$ at sampling interval $dt = 1$, for frequency values of $f = 0, 0.25, 0.5, 0.75, 1.0$ hertz.

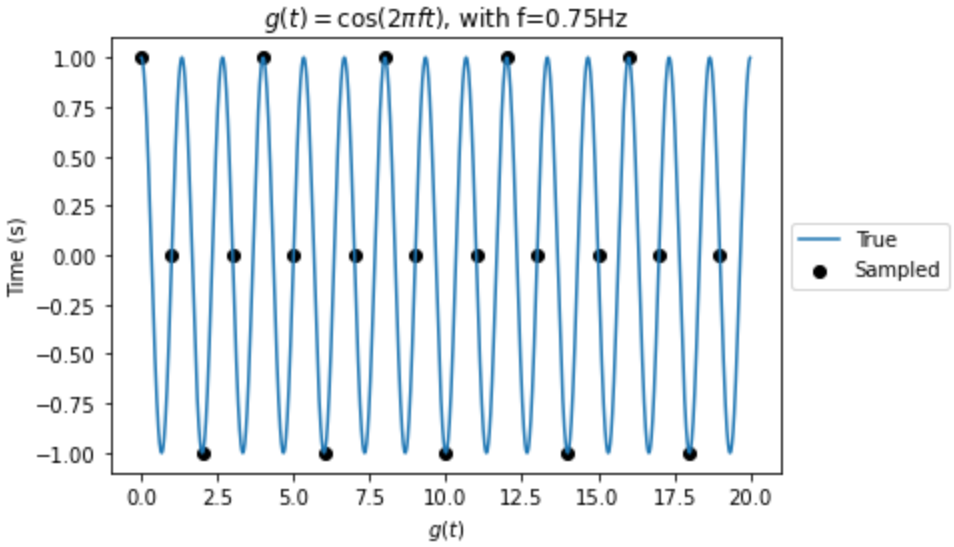
In each case, plot on the screen the points of the resulting time series (as isolated red crosses) to see how well it approximates $g(t)$ (plotted as a blue-dotted line, try a very small dt fine sampling). Submit only plots for frequencies of 0.25 and 0.75 Hertz, use xlabel, ylabel, title commands to annotate each plot. For each frequency that you investigated, do you think the sampling time series is a fair representation of the original time series $g(t)$? What is the apparent frequency for the sampling time series? (Figure out after how many points (N) the series repeats itself, then the apparent frequency = $1/(N * dt)$. You can do this either mathematically or by inspection. A flat time series has apparent frequency = 0.) Can you guess with a sampling interval of $dt = 1$, what is the maximum frequency f of $g(t)$ such that it can be fairly represented by the discrete time series? (Please attach your Python code.)

Answer

In [114…

```
f_list = [0,0.25,0.5,0.75,1.0] #hz
dt = 1

t = np.arange(0, 20, dt)
t_true = np.arange(0,20, 1e-2)

for i in range(len(f_list)):
    f = f_list[i]
    g = np.cos(2*np.pi*f*t)
    g_true = np.cos(2*np.pi*f*t_true)

    if f in [0.25,0.75]:
        plt.plot(t_true, g_true, label='True')
        plt.scatter(t,g, color='k', label='Sampled')
        plt.title('$g(t) = \cos(2 \pi f t)$, with f='+str(f) + 'Hz')
        plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
        plt.ylabel('Time (s)')
        plt.xlabel('$g(t)$')
        plt.show()
```





The sampling time series is not always fair representation of the original time series $g(t)$. The important parameters here is the Nyquist frequency for the highest frequency for f.

We are sampling at $1Hz$, this means that the Nyquist frequecny is 0.5. This means we can only adequatly measure a time-series with a frequency of ~$0.25hz$ and below.

In the case of $f = 0.25Hz$ is lower by a factor of 2 then the Nyquist frequecny, this means our measurements are not distorted.

In the case of $f = 0.75Hz$ is lower then the Nyquist frequecny;however, its not low enough (a factor of 2) this means our measurements are distorted.

The apparent frequecny for $0.25Hz$, is $0.25Hz$, while for $f = 0.75Hz$ the apparent frequecny is $0.25Hz$

In [ ]: