|  |  |
| --- | --- |
| **Name: Caspar Hendrickson** |  |

|  |
| --- |
| **Names of people you worked with:** |
| * Jonathan Kaminski |

|  |
| --- |
| **Websites you used:** |
| * <https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp1d.html> * <https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.subplots.html> * https://docs.scipy.org/doc/numpy/reference/generated/numpy.digitize.html |

|  |  |
| --- | --- |
| **Approximately how many hours did it take you to complete this assignment (to nearest whole number)?** | 4 |

The Rules: Everything you do for this lab should be your own work. Don't look up the answers on the web, or copy them from any other source. You can look up general information about Python on the web, but no copying code you find there. Read the code, close the browser, then write your own code.

By writing or typing your name below you affirm that all of the work contained herein is your own, and was not copied or copied and altered.

Caspar Hendrickson

**Note: Failure to sign this page will result in a 50 percent penalty. Failure to list people you worked with may result in no grade for this lab. Failure to fill out hours approximation will result in a 10-percent penalty.**

**Turn .zip files of Python code to Canvas or your assignment will not be graded**

**BEFORE YOU BEGIN**

1. Please read “ROB456 Fall 2019 Pycharm Installation.pdf” in the Homework 1 assignment folder
2. Install the [PyCharm](https://www.jetbrains.com/pycharm/) Integrated Development Environment (IDE) on your computer, following the above instructions
3. Download probabilities\_studient.py from the Homework 1 assignment folder.
4. Make sure you can run it

**Learning Objectives:**

You should be able to do the following:

* Run Python on your laptop
* Generate and re-sample from a probability distribution function
* Understand the difference between a Gaussian probability distribution and a non-linear one
* Understand the difference between a probability density function and a probability mass function – i.e., why we usually use histograms in code for non-Gaussian distributions
* How to represent a probability density mass as a set of *samples* (particles)

**Homework Guidelines:**

1. Fill in the Homework 1 “fill in here” sections in the python code
2. Follow the Python style guide
   1. Write an English description for each function
   2. Add comments where needed to clarify your code.
3. Copy solutions into the spaces outlined below
4. We’ve provided code to do the plotting

Remember to hand in the python file to Canvas and this file (with solutions) to Gradescope

**Problem 1: Gaussian PDF**

Goals:

1. Create a probability distribution function from a Guassian
2. Use samples from that pdf to create a discretized version of that pdf (i.e., a probability mass function)
   1. X can take on a finite number of values (in this case, evenly spaced x values)
   2. The sum of the probabilities for all values should be 1

Fill in plot\_gauss\_sampled.

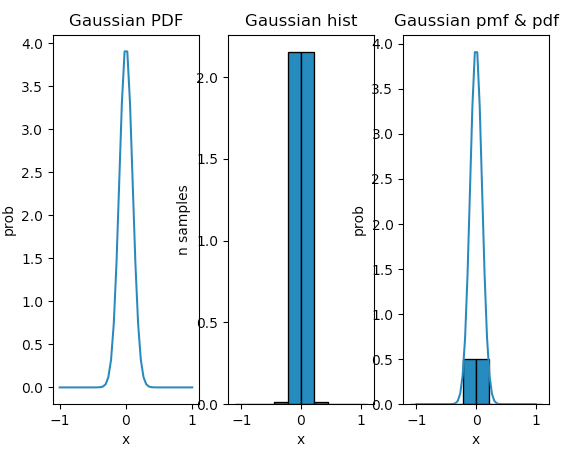
Step 1: Using the mu, sig, and x\_lim values provided, fill in the functionality in plot\_gauss\_sampled.

* Using the provided Gaussian function, create a lambda function that takes in one parameter (x) and returns the Gaussian with the provided mu & sig values
  + Look up python lambda functions if you’re unfamiliar with these
* Using numpy’s quad function, calculate the area under the curve between the provided x\_lim values
  + You’ll want to just get the first return value:
    - area = quad(f, x\_lim[0], x\_lim[1])[0]
* Create a new lambda function that is normalized (divide by the area)
  + This is a probability distribution function because the area under the curve sums to one
* Create n samples from you PDF function. The samples should be evenly distributed between the x\_lim values
  + This is your histogram (hist)
* Normalize the histogram values by dividing by the area given by the bins
  + This is your PMF
* Call create\_plot to make the plot
* Print out the pre and post area values (see below)

Step 2: Determine mu, sig values that give you a reasonable pdf with x\_lim = [0,1], similar to the figure shown below

Suggested reading: numpy.linspace, matplotlib.plot, Normal distributions

Target plot for Step 1:



Area values for Step 1 (using 10 samples)

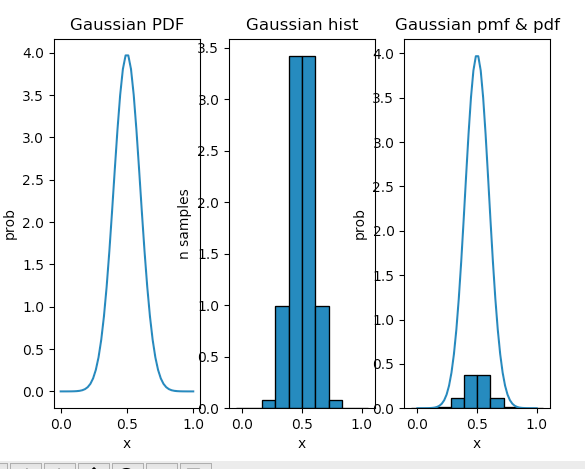
PDF area for Gausian is 0.25

PDF area for normal distribution is 1.0, should be 1

Histogram area is 4.3

PMF area is 1.0, should be 1

Plot for Step 2: This can vary slightly, depending on what mu/sig values you chose



|  |
| --- |
| **Answer the questions:** |
| What mean and standard deviation did you use: mu=.5, sig=.1 |
|  |

|  |
| --- |
| **Figure output** |
|  |

|  |
| --- |
| plot\_gauss\_sampled |
| def plot\_gauss\_sampled(mu=0.0, sig=0.1, x\_lim=[-1, 1], n=10):  *"""  Plot a Gaussian with the given parameters, then normalize and sample to create a pdf and a pmf* ***:param*** *mu: Center of the gaussian* ***:param*** *sig: Sigma - related to standard deviation/width* ***:param*** *x\_lim: Start and stop values* ***:param*** *n: Number of samples to use* ***:return****: None  """* function\_one = lambda x: gaussian(x, mu, sig)  area = quad(function\_one, x\_lim[0], x\_lim[1])[0]  pdf = lambda x: gaussian(x, mu, sig) / area  area\_pdf = quad(pdf, x\_lim[0], x\_lim[1])[0]  samp = np.linspace(x\_lim[0], x\_lim[1], n)  hist = pdf(samp)  area\_bins = sum(hist)  pmf = hist / area\_bins  pmf\_area = sum(pmf)    print('PDF Area Gaussian:')  print(area)  print('PDF Area Normal Distribution:')  print(area\_pdf)  print('Histogram Area')  print(area\_bins)  print('PMF Area')  print(pmf\_area)  print('End of current step')  create\_plot(pdf, hist, pmf, xlim\_pf=[x\_lim[0], x\_lim[1]], title="Gaussian")  # begin homework 1 - Problem 1  # Use a lambda function to create the unnormalized pdf  # calculate area under curve for pdf  # Create a normalized pdf by dividing by the area  # calculate area under curve for normalized pdf (should be 1)  # Sample to create the histogram  # Sum of the histogram values  # Create normalized histogram/pmf  # Plot  # Print area answers |

|  |
| --- |
| **Console output** |
| Copy your step 2 area print outs here  Begin homework 1, problem 1  PDF Area Gaussian:  0.2506628274631001  PDF Area Normal Distribution:  1.0  Histogram Area  4.334696384386622  PMF Area  1.0  End of current step  PDF Area Gaussian:  0.25066268375731304  PDF Area Normal Distribution:  1.0  Histogram Area  9.000003049155621  PMF Area  0.9999999999999999  End of current step |

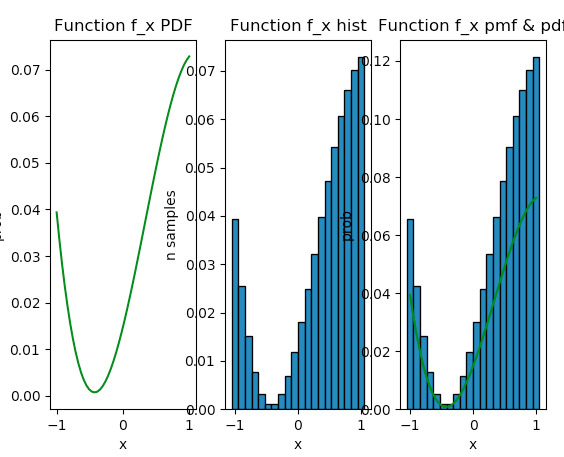
**Problem 2: Creating a PMF from a random function**

Goal: Learn how to make a PMF from any random (positive) function.

Fill in plot\_f\_sampled.

Repeat problem one but with the supplied f\_x function. One small change: evaluate f\_x from -10 to 25, but create your pdf so it goes from -1 to 1. As before, your pdf function and your pmf should have the area under the curve/sum of probabilities be 1.

Example final plot (with 20 samples – use 15 for yours):



|  |
| --- |
| **Comments for grader/additional information (if any)** |
| Had issues with getting the area print outs. I know that it looks messy. Sorry, I had matching graphs but my area was funky.  15 or 100 samples? I’ll include both. |

|  |
| --- |
| plot\_f\_sampled. |
| def plot\_f\_sampled(n=100):  *"""  Plot the f\_x function above, then normalize and shift to 0,1 to create a pdf and a pmf  :param n - number of samples  :return: The PMF  """* # x limits we're using to evaluate f\_xq  x\_lim\_fx = [-10, 25]   # x limits we want to use for pdf  x\_lim\_pdf = [-1, 1]   # for the return - you should be creating this  pmf = [0.1, 0.8, 0.1]   # begin homework 1 - Problem 2  # calculate area under curve for f\_x  # Create a normalized, shifted pdf by dividing by the area and shifting x  # so x=-1 goes to -10, x=1 goes to 25  # Create a function that maps x in x\_lim\_pdf to x in x\_lim  # Build a lambda function that uses the mapping function and normalizes for area  # calculate area under curve for normalized pdf (should be 1)  # Sample to create the histogram  # Summed values of the histogram  # Create normalized histogram/pmf  # Plot  # Print area answers  # end homework 1 - Problem 2    def mapping\_limits(x, lim\_old, lim\_new):  mapped = interp1d(lim\_new, lim\_old)  return mapped(x)   area = quad(lambda x:(f\_x(mapping\_limits(x, x\_lim\_fx, x\_lim\_pdf))), x\_lim\_pdf[0], x\_lim\_pdf[1])[0]  samp = np.linspace(x\_lim\_pdf[0], x\_lim\_pdf[1], n)  pdf\_maybe = lambda x: (f\_x(mapping\_limits(x, x\_lim\_fx, x\_lim\_pdf)))/area  area\_pdf = quad(pdf\_maybe, x\_lim\_pdf[0], x\_lim\_pdf[1])[0]  hist = (pdf\_maybe(samp))  area\_bins = sum(hist)  pmf = hist / area\_bins  pmf\_area = sum(pmf)     print('PDF Area Normal Distribution:')  print(area\_pdf)  print('Histogram Area')  print(area\_bins)  print('PMF Area')  print(pmf\_area)  print('End of current step')  create\_plot(pdf\_maybe, hist, pmf, title="F\_X")  return pmf |

|  |
| --- |
| **Figure** |
| Put your plot here. Use 100 samples  15 Samples  100 Samples |

|  |
| --- |
| **Console Output** |
| Begin homework 1, problem 2  PDF Area Normal Distribution:  1.0000000000000002  Histogram Area  0.45807575587276605  PMF Area  1.0  End of current step |

**Problem 3: Sampling from a PMF**

The assumption here is that you have a function f (or a physical process) that you can sample in some way (i.e., take an action and see what happens, like rolling a dice) but you don’t have code for it. So all you can do is generate samples and count how many times each you “pick” an x value.

Your input is the PMF from problem 2. You already have the probability values, the x values the variable can take on are evenly spaced between 0 and 1 (see the definition of x\_vals).

Fill in plot\_pmf\_samples.

The math/code: Remember that the sum of your PMF values should be 1 (i.e., probability values sum to 1). We can use numpy’s uniform function to generate random numbers evenly distributed between 0 and 1. Think of the PMF values as “occupying” a bit of the range 0 to 1 corresponding to their probability values. For example, if your PMF was [P(x=0) is 0.1, P(x=1) is 0.8], then any sample less than or equal to 0.1 goes in the x=0 bin, while all others go in the x=1 bin.

Step 1: Create an array that represents the **boundaries** of your PMF values in the range 0 to 1. I.e., the boundary of the first bin is 0 and the value of the first PMF. The second bin starts where the previous one left off, and is as wide as the second PMF value. Etc.

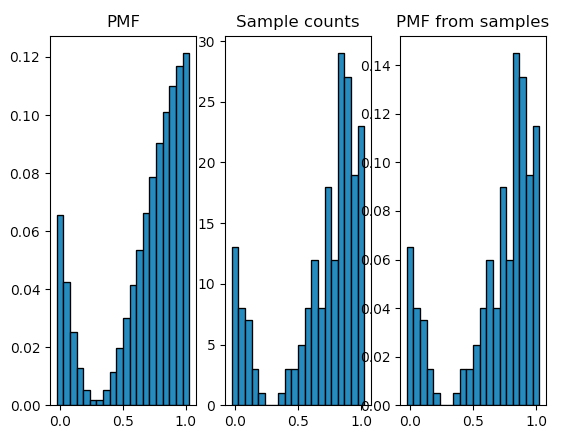
Step 2: Generate n uniform samples. For each sample, figure out which bin it fell into, and add one to that bin’s count.

Step 3: Plot the counts. If you’re using n > 100 or so you should get something that looks (approximately) like the original PMF.

Step 4: Normalize the counts so you get a PMF back.

Step 5: Find a number of samples that gives you (roughly) the same PMF back.

Example plot, using only 200 samples



|  |
| --- |
| **Comments for grader/additional information (if any)** |
| My plots are in a different order. Just to let you know. |

|  |
| --- |
| **how many samples did you decide to use?** |
| 5000 creates a good model of the original PMF  Probably too many for an actual robot but my PC is fast. |

|  |
| --- |
| **plot\_pmf\_samples** |
| def plot\_pmf\_samples(pmf=[0.1, 0.8, 0.1], x\_lim=[0, 1], n=10):  *"""  Generate samples from the PMF then plot how many samples were found in each bin* ***:param*** *n - number of samples* ***:return****: None  """* # Make the subplots  f, (ax1, ax2, ax3) = plt.subplots(1, 3)   # Plot the pmf using uniform samples spaced in x  x\_vals = np.linspace(x\_lim[0], x\_lim[1], len(pmf))  plot\_pmf(ax= ax1, pmf=pmf, x\_vals=x\_vals, title="PMF")   # begin homework 1 - Problem 3  # Generate uniform samples in the range of x\_lim (use numpy's uniform function)  uniform\_samples = np.random.uniform(x\_lim[0], x\_lim[1], n)  boundaries = []  sum = 0  for i in range(len(pmf)):  boundaries.append(sum + pmf[i])  sum = sum + pmf[i]  # print(boundaries)  bin\_counting = list(np.digitize(uniform\_samples, boundaries))  # print(bin\_counting)  counters\_list = []  for n in range(len(pmf)):  counters\_list.append(bin\_counting.count(n))  # print(counters\_list)  area = np.sum(counters\_list)   normalized\_pmf = counters\_list/area   # Make the boundaries of the bins by taking a running sum   # Figure out which bin to put each sample in  # plot the counts using plot\_pmf  plot\_pmf(ax2, counters\_list, x\_vals, "Counters")  plot\_pmf(ax3, normalized\_pmf, x\_vals, "PMF from Samples")  plt.show()  # normalize the bin counts and plot  # plot the normalized bin counts  # end homework 1 - problem 3 |

|  |
| --- |
| **Figure** |
|  |