DAT110: Exercise Sheet 6

Alexander Johannes Stasik

8st of March, 2024

Exercises are due on Friday, 15th of March 2024, or if marked, to be done during the exercise sessions during the week.

1 Summarizing

The goal of this exercise is to formalize and collect things we did in the previous lectures, so that you see more of a connection and a red thread. For the following, we assume we are working with a 1 dimensional continuous probability distribution, defined between $x_{\min}$ and $x_{\max}$.
Complete the table below

name	abbreviation	equation
Probability density function	*pdf*	$p(x)$
Normalization	norm	$E(1) = \int_{x_{\min}}^{x_{\max}} 1 \cdot p(x) \mathrm{d}x$
Cumulative density function	cdf	...
Mean	μ	$E(x) = \ldots$
Variance	...	...
Skewness	...	...
Kurtosis	...	...
General expectation value	...	$E(g(x)) = \ldots$
Inverse cumulative density function	icdf	
Median		
Percentile function		

There might not be clear answers for everything, and nomenclature might be inconsistent. The goal here is that you ill out as much as you can. Put special attention to the last column, I want you to realize how those quantities are all connected.

2 Interpolation (coding exercise, voluntary)

The goal of this exercise is to write your own linear interpolation function. As a reminder, the logic of linear interpolation is the following:

1. Provided an x value, find the closest data points left and right of x, we call them x_l and x_r for now, likewise their y values y_l and y_r.

2. If x is outside the range of data, throw an error message, a print, or whatever to notify the user

3. Compute the parameters of a linear function $f(x) = mx + b$ that connects the two points linearly, such that $f(x_l) = y_l$ and $f(x_r) = y_r$.

4. Evaluate the function $f(x) = y_{\text{interpolate}}$ and return it.

The function you are writing should receive a list of pre-computed data points, two 1-d `numpy` arrays, and an x value to evaluate. The pre-computed data points do not need to be equally samples. The two challenges are to find the left and right data point, and to do the linear function. Finding the linear function is easy once you write down the math, and the local interpolation function can be computed on the fly. Getting the left and right neibour is easy if you know the right `numpy` functions:

- `np.argsort(x)` return the indices which sort the array. You need those, because you want to sort both your x and y coordinates. Basically, it can be used like this
  - `id_sorted = np.argsort(x)`
  - `x_sorted = x[id_sorted]`
  - `y_sorted = y[id_sorted]`

The second trick is to use `np.digitize`. Read the documentation, but it does basically everything you already wrote yourself for the histogram exercise. Either use your old code if you trust it, or `np.digitize`.
You can test your code by comparing with numpys interpolation function, and by plotting it visually.

3 Numerical integration (coding exercise)

In the lecture, we used numerical integration. While this is a complicated topic, we will code up the basic idea and will see that it actually worked pretty well. Recall that an integral is the area under a curve. You have all seen that this can be approximated by narrow bars, as seen in figure 1

So if we make the bars tighter and tighter, we will approximate the integral. Mathematically speaking, this means

$$\int_a^b f(x)\mathrm{d}x \approx \sum_{i=0}^{N} A_i.$$

So we want use a computer and compute the sum of all the bars. If we split up the interval $[a, b]$ into N small pieces, they all have length $\Delta x = \frac{b-a}{N}$. As in

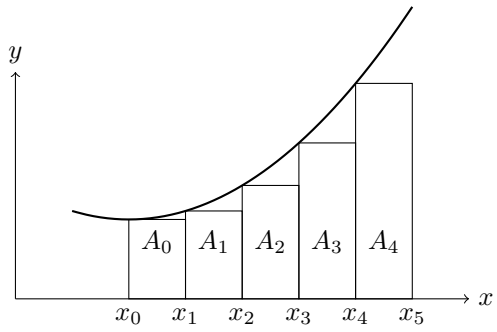

Figure 1: Riemann sum for the function $f(x) = 0.3x^2 - 0.6x + 1$

figure 1, for the height, we just use the left edge of the bar[1], which is $f(x_i) = f(a + i \cdot \Delta x)$, so we have $A_i = f(a + i \cdot \Delta x) \cdot \Delta x$.

3.1 Implementation

For the exercise, we want to write a function that takes as input another function to be integrated, left integration border, right integration boarder and number of bars. Use numpy's linspace method to create the bars. Use a for-loop to iterate over all x_i, compute the A_i and sum them up. That's our integral.

3.2 Testing

Test your function with some known integrals, e.g. $\int_0^1 x\mathrm{d}x$, $\int_0^1 x^2\mathrm{d}x$, and so on, that you can easily verify. Play with number of support points and see how accuracy increases as well as computation time.

3.3 Testing on the uniform distribution

We have worked with the uniform distribution a lot. Let us for now consider the uniform distribution between 0 and 1. Given the analytical results, compute mean, variance and skewness given $a = 0$ and $b = 1$. If you do not recall the formulas, look it up on the old exercises, lecture slides or online. Now compute the same quantities (mean, variance and skewness) using your own integration function. Given a sufficient N, you should get very accurate results.

[1]Note that there are better methods to do this, but they all rely on the same idea, which is this one.

4 Hypercube and Hypersphere (Coding Exercise)

In this exercise, we will perform some random sampling to study volumes of higher-dimensional objects, similar to what we did in the lecture to approximate π. Let's first define a hypercube and a hypersphere.

A cube is a three-dimensional object that includes all points in the interval $[a, b] \times [a, b] \times [a, b]$, and its volume is $V = (b - a)^3$. We want to generalize this to n dimensions. This is simply all points $x \in \mathbb{R}^n$ that lie in the interval $[a, b] \times [a, b] \times \cdots \times [a, b]$, and this obviously has a volume of $V = (b - a)^n$. To check that this definition makes sense, we can investigate the 2-dimensional hypercube, typically known as a square. It is obvious to check that the hypervolume of it (pedestrians might refer to it as the area) is actually $A = (b - a)^2$. Note that there is also a 1-dimensional hypercube, which has a *volume* of $b - a$.

Now, the hypersphere: A sphere is an object where all points lie within a certain radius r. If we say that $x \in \mathbb{R}^n$, then

$$r = |x|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

and $x = (x_1, x_2, \ldots, x_n)^T$. So our hypercube is defined as all x that have $r < R$, and we are interested in the volume of a hypercube. As a mental exercise, check that our hypercube definition makes sense for 1, 2, and 3 dimensions.

4.1 Coding Exercises

We want to estimate the volume of an n-dimensional hypersphere with radius 1. Note that every hypersphere with radius $R = 1$ is fully contained in a hypercube $[-1, 1]^n$ with volume $V = 2^n$. So, you should do the following steps:

1. Uniformly sample N points from the hypercube $[-1, 1]^n$. Don't use a for loop, but use numpy with a shape argument, so you can draw a lot of numbers at the same time.

2. Check how many of your sampled points lie within the hypersphere of radius one. This can be done on an array by using `np.linalg.norm(X, axis=)` where you need to specify the axis. Read the documentation on `np.linalg.norm` if you are not familiar with it.

3. Voluntary exercise: Code up the same, but without numpy, just using for loops and standard Python. You will need to do two nested loops, one for each dimension and one for each data point. If you make N large enough, you will see a significant performance reduction and see why we want to use numpy whenever possible.

4. Do this experiment with 10^6 data points for all dimensions between 1 and 10 and store the ratio of total points and points in the hypersphere.

5. The volume of a hypersphere is actually $V_n = \frac{\pi^{n/2}}{\Gamma\left(\frac{n}{2}+1\right)} r^n$ where Γ is the gamma function. You can evaluate it using `from scipy.special import gamma`. Compare your findings with the analytical solutions graphically by making a plot with the x-axis for n and y-axis for the volume, one line for the analytic solution, one for your numerical result. Can you run the code for 10^3 data points as well and plot this, too?