

Exercise_6_Problems

November 2, 2018

1 Exercise 6: Arbitrary distributions, moving averages, and Monte-Carlo

```
In [ ]: import numpy as np
import pandas as pd
import datetime
import scipy.stats as stats
import matplotlib.pyplot as plt
```

1.1 1. Sampling from an arbitrary distribution

As seen in exercise 4, you can use uniformly distributed random variables, which are in principle themselves simple to generate, to draw samples from the normal distribution via the Box-Muller transform. A more general approach is to sample according to the inverse of the cumulative distribution function (CDF).

A simple example is to generate numbers from the exponential distribution.

$$f(t; \lambda) = \lambda e^{-\lambda t}$$

- Write the CDF $F(T, \lambda)$ and find its inverse ($T = \dots$)
- Write a function to compute this, and compare your result to that from scipy (hint: sometimes called percent-point function or quantile function)

```
In [ ]: # Quantile function
def exp_quantile(p, l):
    ...

    p = np.linspace(0, 1, 100)
    l = 0.2
    ...
```

- Now draw N samples from the uniform distribution $[0, 1]$. For each sample, calculate $F^{-1}(u, \lambda)$
- Plot a histogram and compare the distribution of points to the exponential pdf

```
In [ ]: N = 1000
l = 0.2
x = ...
```

```

y = ...

...

print('Actual lambda:', 1)
print('Estimated lambda: ', ...)

...

# Check the fit
...

# Plot histogram, fit, and calculated pdf
...

```

2 2. Smoothing data

2.1 2.1 Moving average

The moving average, or rolling mean, is a simple technique which can be used to remove short term or periodic (e.g. seasonal) variations in time series data, for example. It can be viewed as a "smoothing", and can ease trend spotting, for instance. One has to be careful when interpreting and using the result; for instance, it is generally improper to fit on such data.

The simplest moving average can be computed using a "sliding window" of length N , with all weights equal. For example, for a 3 point moving average, the window would be $\frac{1}{3}[1, 1, 1]$.

- Write a function to compute the N point moving average of a data series

```

In [ ]: def moving_average(y, length):
        ...

```

The following line of code loads a dataset (into a pandas DataFrame) containing monthly measurements of variation in the global surface temperature, stretching back as far as 1750. (More data like this can be found on <http://berkeleyearth.org>).

```

In [ ]: df = pd.read_csv('Material/Complete_TAVG_complete.txt', skipinitialspace=True, delimiter=';',
                        df['Date'] = df.apply(lambda row: datetime.datetime(
                                                int(row['Year']), int(row['Month']), 15), axis=1)
                        df

```

- Plot the data. To plot the monthly differences, for example, you can directly write `df2['MDiff'].plot()`

```

In [ ]: # For example...
df.query('Year>1980 & Year<2000').plot(x='Date', y='MDiff')
plt.show()

```

- Apply your moving average filter to the monthly data MDiff. Try (for example) 6 months, 5 years, 10 years. Plot these on top of cuts of the original data to compare.

```

In [ ]: ...

```

2.1.1 2.2 Electronic response of RC circuit

In general, the response of a linearly time invariant system is found to be the convolution of the its impulse response $h(t)$ and the input voltage. Consider a resistor and capacitor connected in series, driven by a time-varying voltage $u(t)$. The impulse response for such a circuit is:

$$h_c(t) = \frac{1}{RC} e^{-t/RC} u(t)$$

- Write a function to calculate the impulse response as a function of time, the resistance, and the capacitance, and input. Take care to normalise the integral.
- Now consider a noisy sinusoidal input voltage $u_N(t) = u(t) + \epsilon(t)$, where $u(t) = \sin(2\pi f_1 t) + \cos(2\pi f_2 t)$, and ϵ is a vector comprising samples draw from $N(0, 1)$. f_1 should be a lower frequency (~factor 10) than (f_2), where the cosine represents a faster ripple riding the fundamental tone. Plot the noisy signal and superimpose the clean signal.
- Calculate the circuit response for your signal and compare the result to the noisy signal and the clean, original signal
- Play with the RC time constant and see the effect on the signal.

Note: this first order low pass filter is exactly equivalent to an exponential moving average. The "memory" of the output is effectively determined by the time constant.

```
In [ ]: def rc_impulse(t, R, C):
        # Impulse response
        ...

        def rc_response(t, u, R, C):
            # Cumulative response
            ...

        t = np.linspace(0, 0.1, 5000)
        dt = t[1]-t[0]
        R = 5e3
        C = 100e-9
        tc = R*C

        f1 = 200
        f2 = 0.1 * f1
        u = ...
        un = ...

        print('Cutoff: ', tc)

        ...

        # Try different cutoffs (remove noise, fast ripple, then whole thing)
        ...
```

2.2 3. Monte Carlo methods

2.2.1 3.1. Particle propagation

The elementary processes of particle absorption and scattering are random in their nature. Propagation of particles through a slab of material with multiple scattering events may be impossible to calculate analytically, but can easily be simulated with Monte Carlo methods.

- Consider a beam of photons propagating through an absorbing medium with absorption coefficient $\alpha = 0.2$ per unit length. What is the probability of a photon being absorbed in a unit length slab of material?
- Now take a piece of 1D material made up of 100 slices, each unit length. Starting at $x=0$, propagate a beam of 1000 photons through the material, slice-by-slice. At each interface, you should "measure" each photon to determine whether it has been transmitted or absorbed (hint: uniform distribution, $P(abs)$)
- Plot the number of photons which are transmitted at the end of each slice, and compare that to the Beer-Lambert-Bouger law
- Plot a histogram of the distance travelled before absorption for each photon (free paths).

$I(x) = I_0 e^{-\alpha x}$, where α is absorption coefficient

```
In [ ]: N_slices = 100 # Slices of material
        N_particles = 1000 # Number of particles to simulate
        alpha = 0.2 # absorption coefficient
        P_abs = ...

        # Generate N_slices x N_particles matrix of uniformly distributed random numbers.
        # Transform it into a matrix of absorption events, where True = absorption, False = no
        # mean(Abs_events) = P_abs
        Abs_events = ...

        ...

        print('Generated absorption probability (mean) = ', np.mean(Abs_events))
        print('Fraction of escaped particles = ', N_escaped_final/N_particles)

        # Plots
        ...
```

2.2.2 3.2. Monte-Carlo integration: estimate π

In a so-called 'hit-and-miss' approach, or 'simple sampling', one can estimate the integral of an arbitrary, well-behaved function over some interval by scattering many points over some rectangular area A . The probability of a point landing below the curve is proportional to the function's integral. A classic problem is to determine the value of .

- Uniformly distribute N points over a unit area. Plot these on top of a unit circle (or quarter circle)

- Calculate the proportion that are within the bounds of your shape for some number of samples N (for large N , it would be unwise to plot)
- Repeat the exercise for increasing N . For each run, you should compute and store the error $\epsilon = \bar{\pi} - \pi$
- Plot log-log the convergence of your estimate to the actual value (to machine precision) of π , i.e. ϵ vs the number of points N . Compare this to the expected rate of convergence ($1/\sqrt{N}$).