Tutorial12-numericalDifferentiation_Solutions

June 29, 2022

Due on Thursday June 30, at 11:59pm.

1 Make a new notebook

Create a new jupyter notebook and edit its name from "`Untitled`" to "`tutorial##_{identikey}`".

[ ]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

2 think+type: numerical derivatives of arrays (6 pts)

*In this **think+type**, please test out the following code and think carefully about what it's doing. The concepts covered here will help you in the following section(s) and on Homework F!*

During lecture, we learned how to take numerical derivatives using a forward difference scheme:

$$f'(x) = \frac{f(x+h) - f(x)}{h}$$

where h is a small step away from x. If we have an array of values we'd like to differentiate, the value of `h` will be set by the spacing between our x values.

For example, consider a function for a line, $y = 2x$, sampled by an array:

[ ]:
```
x = np.linspace(0, 1, 11)
y = 2 * x
```

We can estimate the derivative from the arrays using "rise over run", e.g. computing the difference between y-points and dividing by the difference in the corresponding x-points. To compute *diff*erences between elements and their neighbors, we can use `np.diff`.

[ ]:
```
# equivalently: dx = x[1:] - x[:-1]
dx = np.diff(x)
dy = np.diff(y)
```

Let's see how this worked.

[ ]:
```
print("x  =",x)
print("dx =",dx)
```

The first element in `np.diff(x)` is `x[1] - x[0]`, the second element `x[2] - x[1]`, and so on. Of course, it worked the same for `y` too:

[ ]:
```
print("y  =",y)
print("dy =",dy)
```

We know the slope of this line is 2, so that's what the derivative, `dydx`, should be. Let's check:

[ ]:
```
dydx = dy / dx       # "rise over run"
print(dydx)          # this should be 2 everywhere!
```

Note that our derivatve `dydx` has a length that is exactly 1 shorter than `x` or `y`, because we've constructed it from differences in their values. Therefore, if we want to plot it against `x`, we need to cut one element from `x`, as shown below.

[ ]:
```
plt.plot(x[:-1], dydx);
```

3 think+type+pair+code+write: calculate dT/dz in Earth's atmosphere (9 pts)

*In this **think+pair+code**, you will work with your neighbor to apply your array derivative skills. Please discuss the problem collaboratively!*

Earth's atmosphere is different temperatures at different altitudes. You can load two arrays that represent `altitude` (in km) and `temperature` (in K) within Earth's atmosphere by running the commands below:

(Anaconda users, you'll need to copy `earth-standard-atmosphere.txt` from Canvas (Tutorial module) to your tutorial directory and adjust the path below as you've done in previous tutorials. Remember, the easiest way is to use the Jupyter File Browser to "Upload" the txt file into the same directory where your tutorial file is located. Then you can eliminate the `files` variable.)

[ ]:
```
# JupyterHub users
files = "/home/hama2717/astr2600/shared/"
files=""
import pandas as pd
table = pd.read_csv(files + "earth-standard-atmosphere.txt",
                    sep='\s+')
altitude = table['altitude'].values
temperature = table['temperature'].values
```

- Plot the `altitude` (z, on the y-axis) vs `temperature` (T, on the x-axis) in Earth's atmosphere.
  - Make **sure** you've got the right variables on the right axes. Double check this!
  - You should see the standard Earth atmospheric temperature profile (like this one).

- Calculate the rate of change of temperature ($\frac{dT}{dz}$) (in units of K/km) from T and z.
- (In a second plot) Plot the `altitude` (z, on the y-axis) vs the rate of change of temperature ($\frac{dT}{dz}$) as we move upward in Earth's atmosphere.
  - Take a moment to make sure you understand this plot. Effectively, you are seeing how much the temperature changes (dT/dz) at every altitude point (z) in the atmosphere.
- Green Mountain, just behind the Flatirons, is about 1km higher in altitude than we are in Boulder right now. ***Based on your second plot***, estimate about how much colder do you think it is up there and explain how you derived this.

[ ]:
```
plt.plot(temperature, altitude)
plt.xlabel('Temperature (K)')
plt.ylabel('Altitude (km)');
```

[ ]:
```
# calculate the numerical derivative of the temperature
dTdz = np.diff(temperature)/np.diff(altitude)

# plot the derivative
plt.plot(dTdz, altitude[:-1])
plt.xlabel('dT/dz (K/km)')
plt.ylabel('Altitude (km)');
```

In the lower atmosphere, our plot says that dT/dz is about -6 K/km (and pretty constant up to about 10km). That means that if we move upward by 1 km in altitude, we should feel the temperature drop by about 6 K, or about 11°F.

4 think+type: differentiating (mathematical/python) functions (6 pts)

*In this **think+type**, you will calculate numerical derivatives and experiment with the limitation of how precise Python can be!*

What if we have a function that can accept any values, instead of an array already sampled at discrete intervals? We could use any step size h we want. For example, we could define the function for calculating derivatives:

[ ]:
```
# You can either cut and paste docstrings or skip them entirely.
def forwardDerivative(f, x, h):
    """
    Estimate the numerical derivative of
    function f at positions x using a step size of h
    using  the forward difference method.

    Parameters
    ----------
    f : func
       the function you want to differentiate
    x : np.array
```

```
        the x positions at which you want the derivative
    h : float
        the stepsize to use for the numerical derivative
    """
    y2 = f(x+h)
    y1 = f(x)
    return (y2 - y1) / h
```

Let's test `forwardDerivative` on $y = x^3$. We know the derivative analytically: it is $y' = dy/dx = 3x^2$. At $x = 2$, the derivative should be $y'(x = 2) = 12$. Let's check:

[ ]:
```
def y(x):
    return x**3
```

[ ]:
```
forwardDerivative(y, 2, 0.1)
# Wait, did we just pass a function as an argument
# to another function? Cool!!!
```

Our estimate of the derivative isn't great; let's try again with a smaller value of h.

[ ]:
```
forwardDerivative(y, 2, 0.01)
```

and even smaller yet:

[ ]:
```
forwardDerivative(y, 2, 0.001)
```

It seems the error is of order h.

Let's look at how the accuracy of our numerical derivative changes as we decrease `h`:

[ ]:
```
# array of h-values from 10**-12 to 0.1
h_array = np.logspace(-12, -1, 1000)

dydxOfH = forwardDerivative(y, 2.0, h_array)

# Calculate the relative (fractional) error:
# (Estimate - Exact) / Exact
error = np.abs(dydxOfH - 12.0) / 12.0

plt.loglog(h_array, error)
plt.xlabel('h')
plt.ylabel('Fractional Error')
plt.xlim(1,1e-13);
# Note this is an inverted x-axis. Numbers get smaller to the right!
```

As h (the step size) gets smaller and smaller, our fractional error gets smaller and smaller (as expected). At $h \approx 10^{-7}$, our numerical estimate of the derivative is accurate to better than a part in 10 million! But then weird things start happening for even smaller `h`. The errors go back *up!*

This is caused by *round-off error*, a result of small (and necessary) errors in the way numbers are stored in a computer's memory. We'll come back to this soon.

5 think(+pair)+code: backward derivative (8 pts)

*In this **think+code**, you will create a new function to use the backward difference method. If there are still other folks around, you may want to work with your neighbor(s). It can be helpful to talk through the concepts with another person!*

Create a function called `backwardDerivative` that computes a derivative using a *backward* difference scheme, i.e.

$$f'(x) = \frac{f(x) - f(x-h)}{h}$$

Now, you will apply this new function to the $y = x^3$ function (that you already defined in your notebook as `y`). Compute an estimate of the derivative at $x = 2$ using $h = 0.1$ and compare to the true value of 12 and your earlier result from `forwardDerivative` using the same parameters.

In terms of the function's known derivative $y' = 3x^2$, think about why the backward and forward estimates differ as they do.

[ ]:
```
def backwardDerivative(f, x, h):
    """
    Estimate the numerical derivative of
    function f at positions x using a step size of h
    using the backward difference method.

    Parameters
    ----------
    f : func
       the function you want to differentiate
    x : np.array
       the x positions at which you want the derivative
    h : float
       the stepsize to use for the numerical derivative
    """

    return (f(x) - f(x - h)) / h
```

[ ]:
```
backwardDerivative(y, 2, 0.1)
```

With the known derivative of $3x^2$, we should have gotten an answer of 12. The forward difference gave us an overestimate of this value; the backward difference gave us an underestimate of this value. That's because the forward difference is weighted toward x values a little bit above $x = 2$, and the backward difference is weighted toward values a little below $x = 2$, so we're seeing derivatives estimated closer to those locations.

6 Save and upload

Be sure to save the final version of your notebook. To submit, click "File|Download As >|HTML (html)" inside jupyter notebook to convert your notebook into an HTML web page file. It should have a name like `tutorial##_{identikey}.html`. Please upload this HTML file as your submission to "Tutorial ##" on `Canvas`.