Lab_1_data_numpy PDF

October 11, 2023

```
[ ]: # Initialize Otter
     import otter
     grader = otter.Notebook("Lab_1_data_numpy.ipynb")
```

1 Lab 1: Statistical analysis of data using numpy

Resources: What you're doing laid out visually, see miro https://miro.com/app/board/uXjVOWs8R4Y=/?share_link_id=567398312390
Lab 1: frame, mapping between task and code and the Lab slides https://docs.google.com/presentation/d/1lVYGqoStt0ZdnRAYMfF9Km6f0NgMNkuYgINsRhXASwI/edit?usp=s

Motivation: Whether you're in engineering or business or health care - almost any field nowadays - you need to be able to work with data. Just about every thing that touches a computer now has the ability to store data. Most of this data will be numbers, but sometimes it will be qualitative data (think 3 people like this, 10 people don't).

You can do a lot of data analysis with spreadsheets, but at some point it's almost always easier to write some code to either *put* data into a spread sheet in a form that's useful, to *pull* specific data from one (or more) spreadsheets, or to automate some processes (like creating six custom plots from this month's data showing price trends). Being able to write a bit of code to clean up or re-purpose data is really useful, and not too difficult.

- Lab week 1: Read in data, re-arrange it, and use it to do (text-based) statistical analysis
- Lab week 2: Plot the data you worked with in lab week 1
- Homework weeks 1 & 2: – Make the code more general, so you can look at different data channels – Make nicer plots

Some notes on the data you'll be working with. This is real data captured by a robotic hand designed to pick fruit. The hand is instrumented with a couple sensors (IMUs in each of the three fingers, force and torque information at the wrist and information from the motors driving the three fingers). Each of these sensors outputs a data stream, which we've stored in a csv file.

Big picture: We want to know if we can detect if the apple was picked or not from the sensor data. Each row of the Data/proxy_pick_data.csv file is data from a single picking trial. Each group of n columns represents one time step. We want to plot/analyze data from different data channels to see if there is a difference between the successful and unsuccessful picks.

For this lab the goal is to pull out one data channel (the wrist torque sensor, z value) and print out statistics for failed versus successful picks. Yes, you could do all of this by manually going into the spreadsheet, sorting columns, and setting up some spreadsheet formulas. That works for one data channel… but what if you want to do a different one? Or the data file format changes because someone added another sensor? Or you're asked to throw out the biggest n samples?

Yes, this is going to be frustrating/seem like a lot of work for nothing the first time you do it. The point is not to do this particularly task, but to learn how to access data in dictionaries, lists, and numpy arrays to "pull out" data that you're interested in. Yes, I could just tell you to use numpy slicing to pull out every 15th column, starting with the 3rd column, and sort by the last column, but where would the fun be in that?

```
[ ]: # Libraries that we need to import - numpy and json (for loading the␣
      ↪description file)
     import numpy as np
     import json as json
```

1.1 Reading in data

TODO First step, read in the data from Data/proxy_pick_data.csv and put it in a numpy array pick_data. Don't forget to set the delimiter. - see numpy loadtxt and a_tutorial_numpy.py

```
[ ]: pick_data = np.loadtxt("Data/proxy_pick_data.csv", delimiter=",")
```

```
[ ]: # TODO - set the n_picks variable, then print it out. Do NOT just put in a␣
      ↪number - use the variable pick_data to calculate this
     n_picks = pick_data.shape[0]
     print(f"Number of picks: {n_picks}")
```

```
Number of picks: 660
```

```
[ ]: # TODO - set the variable n_successful and print it out. Do NOT just put in a␣
      ↪number - use the variable pick_data
     #   Calculating the number of successful picks: The number of values in the␣
      ↪last column that are 1 (use np.sum)
     n_successful = np.sum(pick_data[:, -1] == 1)

     print(f"Number of successful picks: {n_successful}")
```

```
Number of successful picks: 355
```

```
[ ]: grader.check("count_rows")
```

```
[ ]: count_rows results: All test cases passed!
```

1.2 JSON, lists, and dictionaries: Getting information from a file

The format of the spreadsheet data is given in Data/data_format.json. TODO: Open up the file using any text editor and look through it to see if it makes sense. Also open up proxy_pick_data.csv in a spread sheet editor and make sure you understand the data format.

```
[ ]: # This reads in the json data
     try:
```

```
    with open("data/proxy_data_description.json", "r") as fp:
        pick_data_description = json.load(fp)
except FileNotFoundError:
    print(f"The file was not found; check that the data directory is in the␣
 ↪current one and the file is in that directory")
```

1.3 How many sensor data channels, how many time steps?

TODO: Figure out how many data channels there are, total, for one time step.

1.3.1 Step 1: Figure out how to get the "Data channels" list out of pick_data_description

Note: pick_data_description is a dictionary.

```
[ ]: # TODO - use the key "Data channels" to get out the list of data channels from␣
      ↪pick_data_description
     data_channels = pick_data_description
```

```
[ ]: n_total_dims = 0
     # TODO 1: turn this pseudo code into real code.
     num_data_channels = len(data_channels)

     #. for each item in data channels
     for chl in data_channels:
     #.     Get the number of dimmensions in that element and add it to n_total_dims
         num = len(chl)
         n_total_dims += num
     #.   Note that each item in data channels is a dictionary
     # TODO: Fill in the number of data channels - again, use a variable, not just a␣
      ↪number
     print(f"Number of data channels items in list: {num_data_channels}, total␣
      ↪summed number of dimensions: {n_total_dims}")
```

```
Number of data channels items in list: 3, total summed number of dimensions: 33
```

1.3.2 Number of time steps

Now do a bit of math to figure out the total number of time steps (number of columns / number of dimensions)

Make sure this is an integer

And remember that there is one extra column (the last one) that stores if the pick was successful or not

Extra help: Google python 3 integer divide for a bit more information on how to get an integer back out

```
[ ]: # TODO: Calculate number of time steps
     n_time_steps = 40
     print(f"Number of time steps: {n_time_steps}")
```

```
Number of time steps: 40
```

```
[ ]: grader.check("read_json")
```

```
[ ]: read_json results: All test cases passed!
```

Data slicing to get out the Wrist torque data Practice slicing - pull out the Z value of the Wrist torque for all picks

You are free to use the fact that the name of the data channel you want is Wrist torque, and that z is the 3rd one, but you should get the actual offset index value from the dictionary, not just do index_wrist_torque_offset = 5 (suppose someone changed the order of the data…).

There are several ways to do this; the simplest is to loop through all of the data channels looking for the one that is called “Wrist torque” and then set the index offset value from that. It would be a good idea to check that you actually found the right starting index by looking at the .json file. Don’t forget that numpy indexes from 0.

Note: Use ==, not **is**, for the string comparison.

The **z** channel is the 3rd one (the starting index field in the json file gives the index for the x channel).

```
[ ]: channel_name = "Wrist torque"
     index_wrist_torque_offset = 5  #  Set it to a value that is NOT a valid index
     # TODO: Turn this pseudo code into real code
     for index, chl in enumerate(data_channels):
         if chl == channel_name:
             index_wrist_torque_offset = index
             break
     # for each channel in data channels
     #     if this channel's name is the one I'm looking for
     #.        Set index_wrist_torque_offset (don't forget the z offset)


     # Check that you actually set the value somewhere in the loop - this is␣
      ↪"defensive coding"
     if index_wrist_torque_offset == 5:
         print(f"Error: No channel {chl} found")

     print(f"Offset for wrist torque: {index_wrist_torque_offset}")
```

```
Error: No channel Data channels found
Offset for wrist torque: 5
```

```
[ ]: grader.check("channel_index")
[ ]: channel_index results: All test cases passed!
```

1.4 Compute stats: Now use slicing to get out all of the wrist torque data and calculate the min and the max

This is a pretty complicated slice. Steps to get there:

- First, use the slice operator, selecting all rows and columns, **data**[**:, :**] to calculate the minimum across the entire matrix of data. This should be the same as **np.min(pick_data)**
- Now change the column slice from all columns to starting at the offset value.
- Now change the slice to take a step/stride of **n_total_dims** instead of 1
- Reminder: slicing is **start:end:step**

Note: You don't need to put an end value in - just leave it blank if you want to go all the way to the end

Remember: The data is in **pick_data**, not **pick_data_description**

Do NOT use a **for** loop for this - use slicing.

```
[ ]: # TODO: Use slicing to get the columns of data you want, and then use min and␣
       ↪max to get the min and max values
     wrist_torque_data = pick_data[:, index_wrist_torque_offset::n_total_dims]
     min_wrist_torque = np.min(wrist_torque_data)
     max_wrist_torque = np.max(wrist_torque_data)
     print(f"Minimum {min_wrist_torque} and maximum {max_wrist_torque} value of␣
       ↪wrist torque z channel")
```

```
Minimum -0.62552044 and maximum 0.340460618 value of wrist torque z channel
```

```
[ ]: grader.check("slicing")
```

```
[ ]: slicing results: All test cases passed!
```

1.5 Boolean slicing to get successful versus unsuccessful picks out

Now for the fun one - do the same slicing but ONLY for successful picks versus unsuccessful. This data is stored in the last column - if the last column is 1 the pick was successful (it is 0 if not)

Note: Use **== 1**, not **is 1**

It may be helpful here to explicitly create a boolean array that is **(number of rows) X 1** to use as the row index for the pick_data, rather than trying to do it "in-place". Then you can check that the array is correct (print it out - it should have Trues and Falses in the same order as the 1s and 0s in the spreadsheet's last column).

This boolean array replaces the **:** row index: **data**[**boolean_array, start:end:step**]

The boolean array is created by doing a comparison, for example, **data == 1**. In this case, you want to get out all rows, but only the LAST column of the data, and compare it to 1.

```
[ ]: # TODO: Create a boolean array to pick out the successful columns, and use that␣
     ↪to pick out the rows. Then
     #. calculate min and max
     successful_data = pick_data[:, -1] == 1
     unsuccessful_data = pick_data[:, -1] == 0

     min_wrist_torque_successful = np.min(pick_data[successful_data,␣
      ↪index_wrist_torque_offset::n_total_dims])
     max_wrist_torque_successful = np.max(pick_data[successful_data,␣
      ↪index_wrist_torque_offset::n_total_dims])


     # TODO: Same thing, but this time pick out the unsuccessful columns
     min_wrist_torque_unsuccessful = np.min(pick_data[unsuccessful_data,␣
      ↪index_wrist_torque_offset::n_total_dims])
     max_wrist_torque_unsuccessful = np.max(pick_data[unsuccessful_data,␣
      ↪index_wrist_torque_offset::n_total_dims])


     print(f"Successful: Minimum {min_wrist_torque_successful} and maximum␣
      ↪{max_wrist_torque_successful} value of wrist torque z channel")
     print(f"Unsuccessful: Minimum {min_wrist_torque_unsuccessful} and maximum␣
      ↪{max_wrist_torque_unsuccessful} value of wrist torque z channel")
```

```
Successful: Minimum -0.293665094 and maximum 0.340460618 value of wrist torque z
channel
Unsuccessful: Minimum -0.62552044 and maximum 0.326538637 value of wrist torque
z channel
```

```
[ ]: grader.check("boolean_slicing")
```

```
[ ]: boolean_slicing results: All test cases passed!
```

1.6 Optional: print out all of the indices where the maximum value for the successful pick was reached

np.where is the method you want; it returns a tuple (think list) with two lists, one for each dimension. To get out the pairs of indices, you want the first element of the first list and the first element of the second list, and so on. You can do this with a for loop, using **zip** to zip together the two arrays.

- for r, c in zip( ):

Some gotchas: it's easiest to do where on the entire pick_data set, but then you'll get indices that are NOT the ones you want (other data channels). If you do max on the sliced matrix, then you'll get indices on the sliced matrix, not the original... So either you need to filter out indices that are not the channel you want with an **if** statement (the modulo operator **%** is useful here) OR adjust the column index by doing **offset + index * n_total_dims**.

See the slides for help on converting to and fro from row column indexing, and see the Miro diagram for more info on re-configuring the output of **where**.

This is OPTIONAL for the lab, but we will be coming back to this in the homework.

[ ]:
```
# Use where to get out the indices. You can use == OR np.isclose() here; either␣
↪works. In general, use .isclose for
#. floating point comparisons.
# Append the row number of any matches to the max into this list
all_rows_with_max = []


all_indices_from_where = np.where(np.isclose(pick_data,␣
↪max_wrist_torque_successful))
# for all row, column in all_indices_from_where
for r, c in zip(all_indices_from_where[0], all_indices_from_where[1]):
    if c // n_time_steps == index_wrist_torque_offset:
#.   if this is the column for wrist torque
        print(f"Row: {r}, Time step: {c // n_time_steps} Successful y/n:␣
↪{pick_data[r, -1] == 1}, value: {pick_data[r, c]}")
        all_rows_with_max.append(r)

if len(all_rows_with_max) > 0:
    assert all_rows_with_max[0] == 82
else:
    print("No suitable conditions")
```

```
No suitable conditions
```

[ ]:
```
grader.check("optional_where")
```

[ ]:
```
optional_where results:
    optional_where - 1 result:
         Test case failed
        Trying:
            assert all_rows_with_max[0] == 82
        Expecting nothing
        **********************************************************************
        Line 1, in optional_where 0
        Failed example:
            assert all_rows_with_max[0] == 82
        Exception raised:
            Traceback (most recent call last):
              File "c:\Users\user10\anaconda3\Lib\doctest.py", line 1351, in
__run
                exec(compile(example.source, filename, "single",
              File "<doctest optional_where 0[0]>", line 1, in <module>
                assert all_rows_with_max[0] == 82
```

```
                     ~~~~~~~~~~~~~~~~~^^^
                IndexError: list index out of range
```

1.7 Hours and collaborators

Required for every assignment - fill out before you hand-in.

Listing names and websites helps you to document who you worked with and what internet help you received in the case of any plagiarism issues. You should list names of anyone (in class or not) who has substantially helped you with an assignment - or anyone you have *helped*. You do not need to list TAs.

Listing hours helps us track if the assignments are too long.

```
[ ]: # List of names (creates a set)
     worked_with_names = {}
     # List of URLS (creates a set)
     websites = {}
     # Approximate number of hours, including lab/in-class time
     hours = 10
     your_column_for_wrist_torque = any
     # for all row, column in all_indices_from_where
     #.    if this is the column for wrist torque
     #.       print(f"Row: {r}, Time step: {c // n_time_steps} Successful y/n:␣
      ↪{pick_data[r, -1] == 1}, value: {pick_data[r, c]}")
     for r in range(len(pick_data)):
         for c in range(len(pick_data[0])):
             if c == your_column_for_wrist_torque:
                 # Assuming 'your_column_for_wrist_torque' is the column index you␣
      ↪are interested in
                 print(f"Row: {r}, Column: {c // n_time_steps}, Successful y/n:␣
      ↪{pick_data[r, -1] == 1}, Value: {pick_data[r][c]}")
```

```
[ ]: grader.check("hours_collaborators")
```

```
[ ]: hours_collaborators results: All test cases passed!
```

1.8 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

Submit just the .ipynb file to Gradescope (Lab 1 Arrays). You do not need to submit the data files. Don't change the provided variable names or autograding will fail. Look at the Gradescope grading rubric for code-quality checks.

```
[ ]: # Save your notebook first, then run this cell to export your submission.
     grader.export(run_tests=True)
```

```
c:\Users\user10\anaconda3\Lib\site-packages\nbconvert\utils\pandoc.py:51:
RuntimeWarning: You are using an unsupported version of pandoc (3.1.8).
Your version must be at least (1.12.1) but less than (3.0.0).
Refer to https://pandoc.org/installing.html.
Continuing with doubts…
  check_pandoc_version()
```

```
Running your submission against local test cases…
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
c:\Users\user10\Desktop\ME␣
↪203\IntroPythonProgramming\IntroPythonProgramming\Week_1_arrays\Lab_1_data_nu py.
↪ipynb Cell 33 line 2
      <a href='vscode-notebook-cell:/c%3A/Users/user10/Desktop/ME%20203/
↪IntroPythonProgramming/IntroPythonProgramming/Week_1_arrays/Lab_1_data_numpy.
↪ipynb#X44sZmlsZQ%3D%3D?line=0'>1</a> # Save your notebook first, then run thi ␣
↪cell to export your submission.
----> <a href='vscode-notebook-cell:/c%3A/Users/user10/Desktop/ME%20203/
↪IntroPythonProgramming/IntroPythonProgramming/Week_1_arrays/Lab_1_data_numpy.
↪ipynb#X44sZmlsZQ%3D%3D?line=1'>2</a> grader.export(run_tests=True)

File c:\Users\user10\anaconda3\Lib\site-packages\otter\check\utils.py:184, in␣
↪grading_mode_disabled(wrapped, self, args, kwargs)
    182 if type(self)._grading_mode:
    183     return
--> 184 return wrapped(*args, **kwargs)

File c:\Users\user10\anaconda3\Lib\site-packages\otter\check\utils.py:166, in␣
↪incompatible_with.<locals>.incompatible(wrapped, self, args, kwargs)
    164     else:
    165         return
--> 166 return wrapped(*args, **kwargs)

File c:\Users\user10\anaconda3\Lib\site-packages\otter\check\utils.py:217, in␣
↪logs_event.<locals>.event_logger(wrapped, self, args, kwargs)
    215 except Exception as e:
    216     self._log_event(event_type, success=False, error=e)
--> 217     raise e
    219 if ret is None:
    220     ret = LoggedEventReturnValue(None)

File c:\Users\user10\anaconda3\Lib\site-packages\otter\check\utils.py:213, in␣
↪logs_event.<locals>.event_logger(wrapped, self, args, kwargs)
    208 """
    209 Runs a method, catching any errors and logging the call. Returns the␣
↪unwrapped return value
```

```
    210 of the wrapped function.
    211 """
    212 try:
--> 213     ret: Optional[LoggedEventReturnValue[T]] = wrapped(*args, **kwargs)
    215 except Exception as e:
    216     self._log_event(event_type, success=False, error=e)

File c:\Users\user10\anaconda3\Lib\site-packages\otter\check\notebook.py:523, i␣
  ↪Notebook.export(self, nb_path, export_path, pdf, filtering, pagebreaks, files ␣
  ↪display_link, force_save, run_tests)
    520         display(HTML(out_html))
    522 if pdf_created or not self._nbmeta_config.require_no_pdf_confirmation:
--> 523     continue_export()
    524 else:
    525     display_pdf_confirmation_widget(
    526         self._nbmeta_config.export_pdf_failure_message, continue_export

File c:\Users\user10\anaconda3\Lib\site-packages\otter\check\notebook.py:505, i␣
  ↪Notebook.export.<locals>.continue_export()
    503 if run_tests:
    504     print("Running your submission against local test cases…\n")
--> 505     results = grade_zip_file(zip_path, nb_path, self._tests_dir)
    506     print(
    507         "Your submission received the following results when run agains␣
  ↪" + \
    508         "available test cases:\n\n" + indent(results.summary(), "    ")
    510 if display_link:
    511     # create and display output HTML

File c:\Users\user10\anaconda3\Lib\site-packages\otter\check\utils.py:103, in␣
  ↪grade_zip_file(zip_path, nb_arcname, tests_dir)
    100 print(results.stdout.decode("utf-8"))
    102 if results.stderr:
--> 103     raise RuntimeError(results.stderr.decode("utf-8"))
    105 with open(results_path, "rb") as f:
    106     results = dill.load(f)

RuntimeError: C:
  ↪\Users\user10\AppData\Roaming\Python\Python311\site-packages\zmq\_future.py:
  ↪693: RuntimeWarning: Proactor event loop does not implement add_reader family␣
  ↪of methods required for zmq. Registering an additional selector thread for␣
  ↪add_reader support via tornado. Use `asyncio.
  ↪set_event_loop_policy(WindowsSelectorEventLoopPolicy())` to avoid this warnin␣.

  self._get_loop()
```