Week 1 (29/09 - 05/10)

30 September 2025 16:07

30/09/2025

11:00 - Met up with Yvonne & Ethan for our first meeting to discuss the project!

Discussions included:

- Initial Admin about getting an ATLAS account
- Information about what to get to work on in Week 1
- What we're going to be looking at:
  - Classifying Processes such as g -> tt or g-> HH etc.
  - Extending this to multi-class classification (i.e. comparing g -> tt and g-> HH to everything else as background)
  - Trying to reconstruct collision kinematics from final state information using ML models
  - Other extensions towards spin or using some angle matrix thing that Yvonne was very interested in (will probably learn more about later)

Work for the week ahead:

1. Read through slides "Intro2ML for Particle Physics"
2. Work through the exercises in "Intro to ML 4 Physicists"
3. Consult Ethan if finished before the end of the week

13:00 - Getting VSCode set up

- Re-installed Python due to Pathing issue
- Installed Pytorch
- Reinstalled Pytorch due to issue with CUDA after confirming information about laptop GPU
- Set up MPhys Folder and Repository in Github (Still needs to be linked)
- Installed Numpy again

15:00 - Starting work on the Exercises

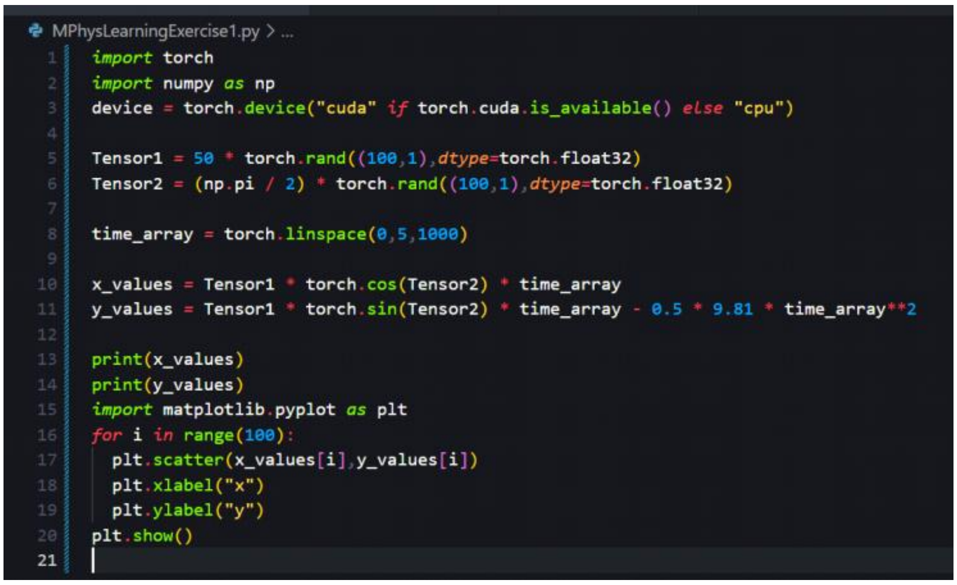
15:05 - Exercise 1 (Filename: 1_Tensors.ipynb)

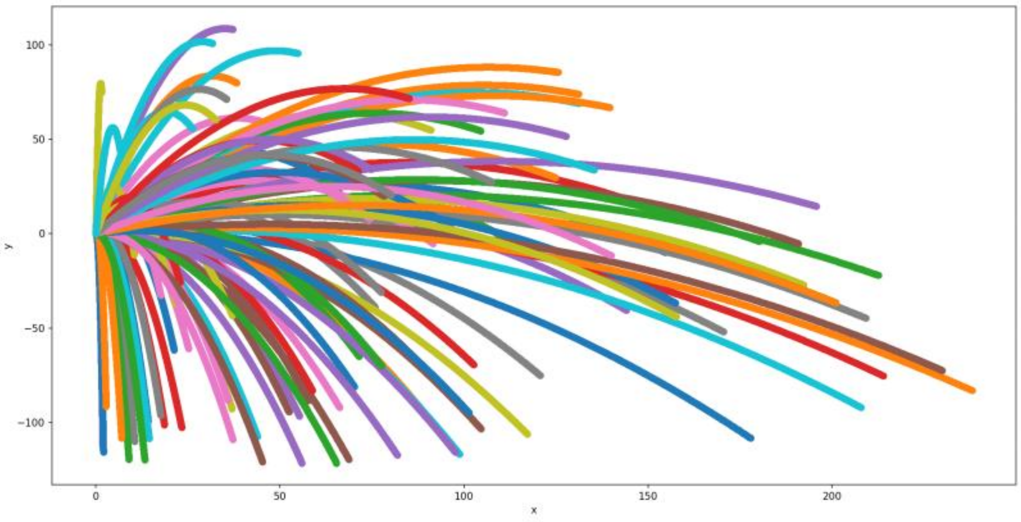
Remember that for more mathematical operations on elements in a tensor, you have to use `torch.sin()` rather than something such as `np.sin()`

Note for self: remember to create tensors with the correct dimensions; I was stuck on an issue with multiplying the tensors in the x and y values section as instead of creating a 2D tensor by multiplying the velocities by the linspace, it was attempting to multiply each linspace value by each corresponding velocity. By switching from (1,100) to (100,1) this problem was trivially fixed.

16:05 - Completed Task 1

C:\Users\Ian Standard\Documents\Manchester\Physics\MPhys_Project
MPhysLearningExercise1Section1.py


```
MPhysLearningExercise1.py > ...
import torch
import numpy as np
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

Tensor1 = 50 * torch.rand((100,1),dtype=torch.float32)
Tensor2 = (np.pi / 2) * torch.rand((100,1),dtype=torch.float32)

time_array = torch.linspace(0,5,1000)

x_values = Tensor1 * torch.cos(Tensor2) * time_array
y_values = Tensor1 * torch.sin(Tensor2) * time_array - 0.5 * 9.81 * time_array**2

print(x_values)
print(y_values)
import matplotlib.pyplot as plt
for i in range(100):
  plt.scatter(x_values[i],y_values[i])
  plt.xlabel("x")
  plt.ylabel("y")
plt.show()

```



16:30 - Exercise 1, Section 2

<u>Stack, concatenate & reshaping operations</u>

Dimension 0 stacks vertically, dimension 1 stacks horizontally, presumably higher level dimensions are for higher order tensors (i.e. dimension 2 would equate to value on the x axis at a given point if a 3D tensor was used to represent field values at different points in space determined by position in an array)

So using dimension 0 on [1,2,3] and [4,5,6] gives:

1 2 3
4 5 6

Using dimension 1 would yield:

1 4
2 5
3 6

Concatenate combines tensors along an existing direction. Given that these tensors have only got 1 dimension, you would just use dimension 0 as it's the only dimension they have. If instead, they were 2 dimensional row vectors (1,3), then you would have to concatenate with dimension 1 to get them to be "stacked" horizontally so that it became 1 long (1,6) row vector.

17:14 - End of day

Had a few teething issues with uploading to a Github repository but have it all sorted out now. Will continue from the point of reshaping when I get back to work as it seems a little confusing and I need some more time to digest it.

23:10 - Decided to do a bit more work whilst it's still fresh in the head

Reshaping makes a bit more sense now having looked at it again. It preserves the number of elements and by using "-1" as one of your dimensions, the program automatically calculates how long to make that dimension so that the element number stays the same. E.g. if you had a 2x3 array and wanted it to become a 6x1 array, you could just put in `Tensor1.reshape(-1,1)` which would automatically calculate the needed size of the first dimension to maintain element number (6).

Filtering seems to make sense for 1 dimensional tensors, but I'm a little confused as to how it would work at higher dimensions - might not be possible? As element number is not preserved.

A few important Git commands just to make it easier:

1. Check what changed:
```
git status
```

2. Stage everything:
```
git add .
```

3. Commit with a message:
```
git commit -m "describe what you changed"
```

4. Push to Github:
```
git push origin master
```

23:52 - A bit confused in Task 2

It seems like the Z-score normalisation seems a bit useless here as the means are already defined to be zero so you're just dividing by the standard deviation? Will ask in teams at a more sociable hour. Will continue anyways assuming a different mean value (just so it's more general).

Nevermind, having done the coding, it turns out `randn` just takes values randomly from a normalised data set with mean zero, rather than the data values having a mean of zero. Whoops. Basically, it is useful for generating data but won't make it perfect with a mean of zero.

Using `torch.min(filtered_info, dim=0).values` returns simply the values without having to print out the indices too.

01/10/2025

00:17 - Completed Task 2

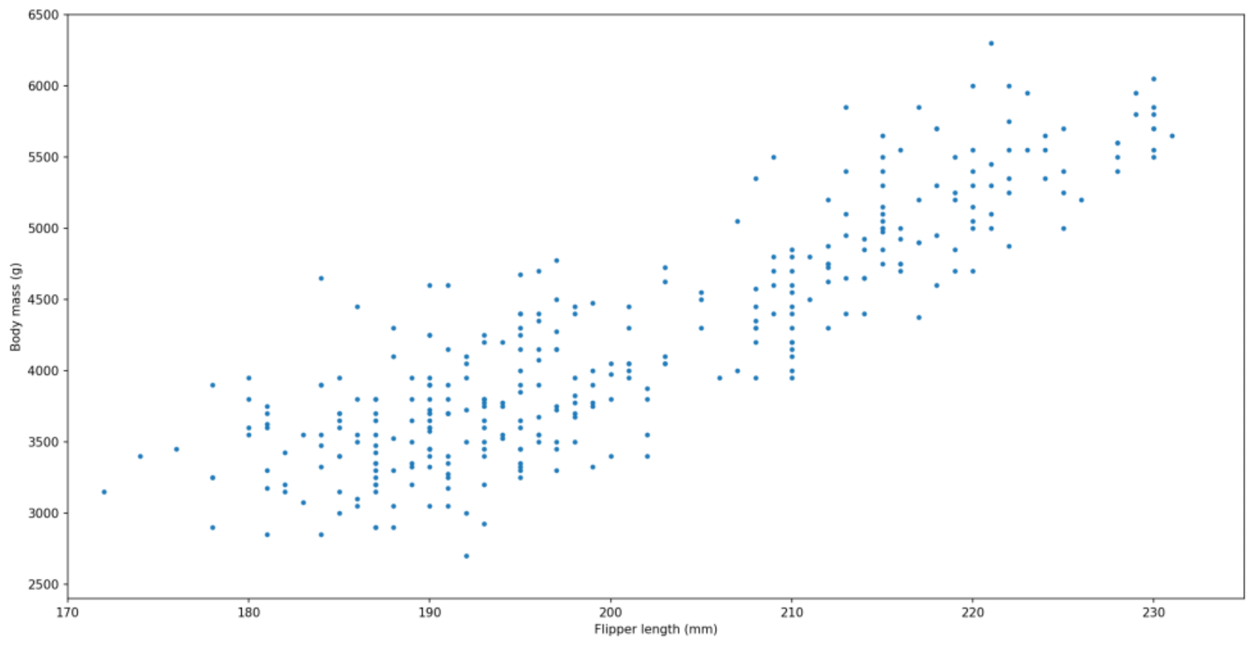
C:\Users\Ian Standard\Documents\Manchester\Physics\MPhys_Project
MPhysLearningExercise1Task2.py

Was quite fun to code - now completed with Exercise 1, on to Exercise 2 next!

11:30 - Exercise 2 (Filename: 2_Regression_Penguins_Sklearn.ipynb)

Need to spend some time looking into pandas further: the .iloc function will likely be important.

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iloc.html



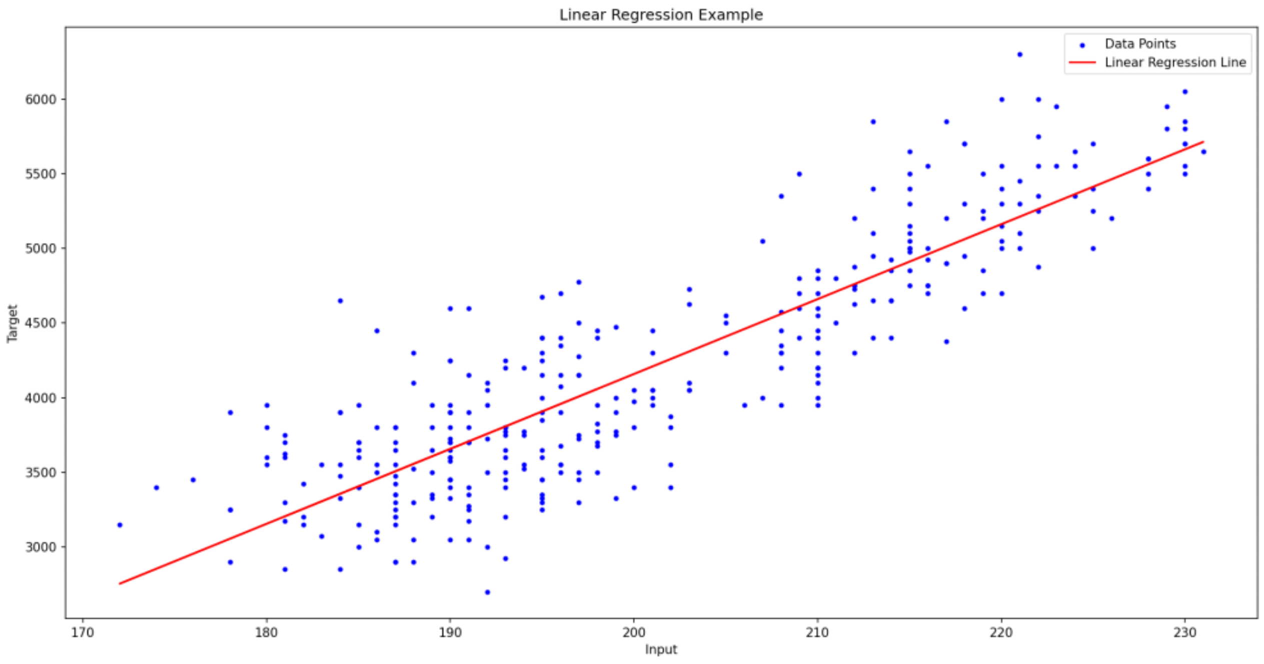
Quickly installed scikit-learn via pip

02/10/2025

10:49 - Exercise 2 continued

Linear regression from scikit is an ordinary least squares solution:

- Imported as:
  ```
  from sklearn.linear_model import LinearRegression
  ```
- Required data shape of Data × Features
- If given 2 variables to fit between, need to turn the Features (X in ML convention) into a column vector using reshape
- Then, define and fit the model using:
  ```
  model = LinearRegression()
  model.fit(X, y_true)
  ```
- Can then create example features to predict:
  ```
  example_flipper_length = np.asarray([300, 500])
  example_body_mass = model.predict(example_flipper_length.reshape(-1, 1))
  print(example_body_mass)
  ```
- Note the reshaping of the features to ensure that X is a column vector
- The same applies to predicting all the data:
  ```
  y_pred = model.predict(X)
  ```
- Can then use this predicted data to plot a regression line against a scatter plot using:
  ```
  plt.scatter(X, y_true, color='blue', label='Data Points', marker='.')
  plt.plot(X, y_pred, color='red', label='Linear Regression Line')
  plt.xlabel('Input')
  plt.ylabel('Target')
  plt.title('Linear Regression Example')
  plt.legend()
  plt.show()
  ```



- The slope and y-intercept are then found using:
  `model.coef_[0]` and `model.intercept_`
- Goodness of fit is calculated using the coefficient of determination:
  - Perfect model, $R^2 = 1$
  - Completely Imperfect model, $R^2 = 0$
- Found using:

```
r_squared = model.score(X, y_true)
print(f"R-squared: {r_squared}")
```