1 Titanic

Introduction. On April 15, 1912, the largest passenger liner ever made collided with an iceberg during her maiden voyage. When the Titanic sank it killed 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships. One of the reasons that the shipwreck resulted in such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others.

Data. The `titanic.csv` file contains data for 887 of the real Titanic passengers. Each row represents one person. The columns describe different attributes about the person including whether they survived (S), their passenger-class (C), their gender (G), their age (A), and the fare they paid (F).

Column S encodes survival by 1 and death by 0; column G encodes male by 0 and female by 1. Passenger classes C are 1 (upper), 2 (middle), and 3 (lower).

Task. Write a Python program that reads the data file and answers the following questions:

1. What was the average fare by passenger class?
2. What was the chance of survival?
3. What was the chance of survival given the gender?
4. What was the chance of survival given the passenger class?
5. What was the chance of survival given the gender and the passenger class?
6. What was the chance of survival by age group?

Hint. Useful packages:

- NumPy Routines (many NumPy packages)
- NumPy: Loading data from a text file
- NumPy: Statistics (mean, std, min, max, ...)
- NumPy: Sorting and Searching

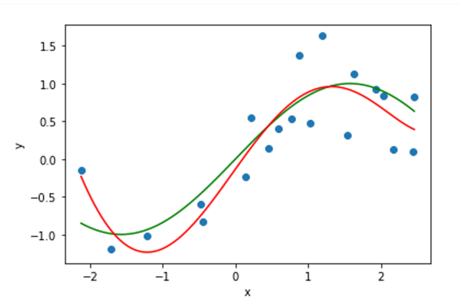
2 Fitting a Polynomial

The goal is to approximate the function $f(x) = \sin(x)$ defined on the interval $[-\pi, +\pi]$ using `polyfit` from the numpy.polynomial.polynomial module.

We randomly draw n input-output examples $(x_1, y_1), \ldots, (x_n, y_n)$ such that

$$y_i = sin(x) + \varepsilon_i$$

where ε_i is drawn from a normal distribution with mean zero and standard deviation 0.5. The following plot shows a sample of $n = 20$ randomly drawn input-output examples (blue dots), the sine-function to be approximated (green), and the least squares fit of a polynomial of degree 5 (red).



We conduct three experiments using the sceleton provided by the Jupyter notebook `polyfit_ex.ipynb`.

Experiment 1 Randomly draw a sample of size $n = 15$. Fit polynomials of degree $1, 3, 5, 7, 9$, report the coefficients, the residual sum of squares, and plot the result as in the above figure. What do you observe?

Experiment 2 Fit a polynomial of degree 10 to samples of size $n = 15$, 30, 50, 100, 250. For each sample, report the results as in the first experiments. What do you observe?

Experiment 3 Generate three samples:

- training sample of size 20
- validation sample of size 10
- test sample of size 100

Fit polynimals of degree $1, 2, .., 10$ on the training sample. Compute the residual sum of squares of the trained polynomials on the validation sample by using the function `linalg.norm` from the numpy.linalg module. Select the model with the lowest residual sum of squares on the validation sample. Apply the model on the test sample and compare the resulting residual sum of squares with the one on the validation set (after normalizing each by the sample size).

3 Gradient Descent

We want to minimize a function $f(x)$ using gradient descent. Starting at some initial solution x, gradient descent iteratively updates the solution according to the rule

$$x \leftarrow x - \eta \nabla f(x),$$

where $\eta \in [0, 1]$ is the learning rate (step size) and $\nabla f(x)$ is the gradient of f at x.

Example. Consider the function $f(x) = x^2$. The gradient (derivative) of f at x is $\nabla f(x) = f'(x) = 2x$. Thus, the update rule becomes

$$x \leftarrow x - 2\eta x$$

Consider the function

$$f : \mathbb{R}^n \to \mathbb{R}, \quad x \mapsto x^t x = \sum_{i=1}^{n} x_i^2.$$

The gradient of f at x is $\nabla f(x) = 2x$ giving the same update rule

$$x \leftarrow x - 2\eta x$$

but in higher dimension.

Task. Implement the basic gradient descent algorithm as a function:

```
1. randomly draw initial solution x
2. for i = 1 to max_iter
      2.2 compute derivative df at x
      2.1 x -= eta * df
      2.2 print f(x)
3. return x
```

Minimize the univariate functions $f(x) = x^2$, $f(x) = x^4$ on $\mathbb{R}$ and the multivariate $f(x) = x^t x$ on $\mathbb{R}^{10}$. Try to find suitable learning rates.