1 Getting started

In today's session we will be looking at the basics of evolutionary algorithms. The topic of evolutionary robotics covers three weeks. To learn to work with these algorithms, in this first week we will focus on one of the most simple applications, namely *simple linear regression*. Your lab report on evolutionary robotics includes results from all three weeks, so keep detailed notes!

1.1 The problem of simple linear regression

As you may remember from your statistics courses, a regression model deals with predicting a real-valued output from a real-valued input, and is trained on a correctly labeled dataset. This makes it a *supervised learning* problem.

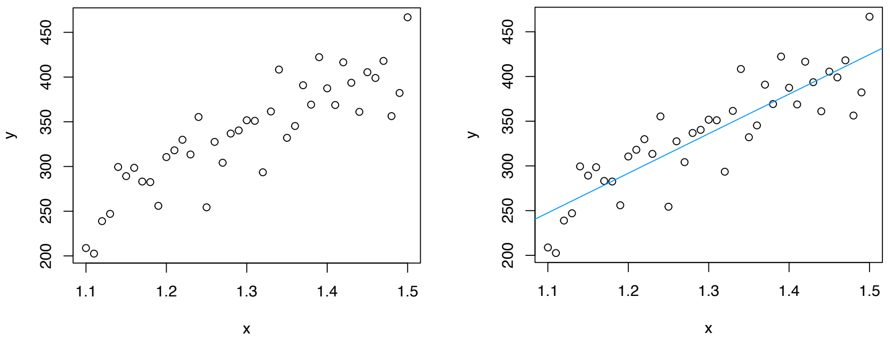

Figure 1: The original dataset is displayed on the left. Using linear regression, we determine a line of best fit, displayed in blue on the right. How do we find the line of best fit?

1.2 Solving a simple linear regression

What we actually want to know when we “solve” a simple linear regression problem is the parameters of the *line of best fit*. When we then get a novel x-value we can predict its y-value by line intersection. A straight line is parameterized by two parameters, which we will call α (intercept) and β (slope) so that $y = \alpha + \beta x$. The intercept represents the line's y-intercept and the slope represents the change in y divided by the change in x (rise over run, or *steepness*). We also need some way to define what a *best fit* is. In other words, we want to find a parameterization (α, β) that minimizes some error function (or maximizes some *best-fit* function). Think about this for a while. What would be a way to quantify the fitness of a line of best fit? Hint: solving a simple linear regression analytically (closed-form) is often done using a method known as *ordinary least squares* (OLS). In fact, solving a simple linear regression analytically for data points $x_i y_i$ and number of observations n can be done using:

$$\beta = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{n \sum x_i^2 - (\sum x_i)^2}$$
$$\alpha = \overline{y} - \beta \overline{x}$$

Can you think of other ways of defining *best fit?* Anyway, in this class we will—of course—not be using the analytic way of determining our parameters, but we will use an evolutionary algorithm. Let's get to work!

1.3 Prerequisites

Make sure you have the *scipy*, *pandas*, and *matplotlib* libraries for Python installed. If not, install them using *pip install [library]*. Also download the datafile associated with your group number from Brightspace. Each group has a specific dataset assigned to it. Make sure you find your group number on Brightspace and work with the correct dataset, as your code will be graded against that specific dataset.

Make sure you load these specific libraries in your files:

```
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
```

2 Getting a feel for the data

Load your dataset as a `pandas.DataFrame`. This is a data structure with labeled rows and columns. Check how many data points your dataset contains, and visualize it using a scatter plot. Now imagine a line of best fit. Estimate its slope and intercept.

3 Quantifying error

3.1 The error function

We start by defining an error function. Our error function will take three arguments: the dataset as a `pandas.DataFrame`, the intercept parameter, and the slope parameter. It will return a float quantifying the error for the given parameterization, so lower numbers correspond to a better fit. Make sure your function is implemented as follows:

```
def get_error(dataset, intercept, slope):
    # Your code here
    return error
```

3.2 The error landscape

The error landscape is a surface in three-dimensional space, where we define the x and y coordinates as the intercept and slope parameters, and the z coordinate as the error. This way, we can visualize the effect our parameters have on the error. Consult this website for information on creating a surface plot. Include your most informative version of this plot in your lab report.

Investigate the error landscape. Is it smooth and convex? Is this well-suited to regular optimization algorithms? How about evolutionary algorithms? Discuss this in terms of local minima and convexity.

4 Optimizing our parameters

4.1 Implementing our evolutionary algorithm

Now that we have an error function, and some idea of the error landscape, we can implement our evolutionary algorithm. The high-level description of a simple evolutionary algorithm is:

Algorithm 4.1: EvoAlg(dataset, population size, generations, threshold)

```
generate initial population
repeat
  evaluate each individual's fitness
  generate new generation
until best individual has fitness higher than threshold
return (parameters)
```

You will need to implement an evolutionary algorithm to find the optimal parameterization given your specific dataset. Make sure your implementation conforms to the following prototype:

```python
def run_evo_alg(dataset, pop_size, gens, epsilon):
    tic = time.perf_counter()
    # Your code here
    toc = time.perf_counter()
    print(f"Evolution completed in {toc - tic:0.1f} seconds.")
    params = [intercept, slope]
    return params
```

This is necessary for automated code checking. Your implementation will be tested on several datasets and compared against known parameters.

As your code runs, after each generation generate this specific output to console:

```python
print(f"Generation {cur_gen} of size {pop_size}; best error {best_error:.0f}; \
intercept {best_intercept:.2f}; slope {best_slope:.2f}")
```

4.2 Things to investigate

As you saw in section 4.1, there are several hyperparameters implemented in our code. We can manipulate population size (how many candidate solutions do I have in each generation?), the number of generations, and some threshold that represents our required accuracy.

4.2.1 Population size

Manipulate population size to investigate its effects. Set a fixed threshold. Create two figures: 1) showing population size on the x-axis and run time on the y-axis, and 2) showing population size on the x-axis and generations until threshold reached on the y-axis. Discuss.

4.2.2 Number of generations

Given a population size, show in a figure how the error decreases over generations and when it plateaus. Now create two more lines in the same figure showing the same for two other population sizes.

4.2.3 Other hyperparameters

Of course, there other factors that are important. Think of the elitism parameter: what percentage of best solutions are copied to the next generation? Or mutation rate: how much noise do I want to add to each subsequent generation? Discuss these and give examples!

4.2.4 Comparison to ground truth

Perform an ordinary least squares (OLS) regression. using the `stats.linregress()` function. Compare the results from your evolutionary algorithm to the OLS output. Discuss any differences and their causes.

4.3 Submission

You will submit your lab report after the third week of evolutionary robotics. It will consist of your work from all three weeks. Submit your code to a separate assignment that is able to accept .py files. Instructions will be given after the third week.

Good luck with this week's assignment, and remember: we are here to help you if you get stuck.