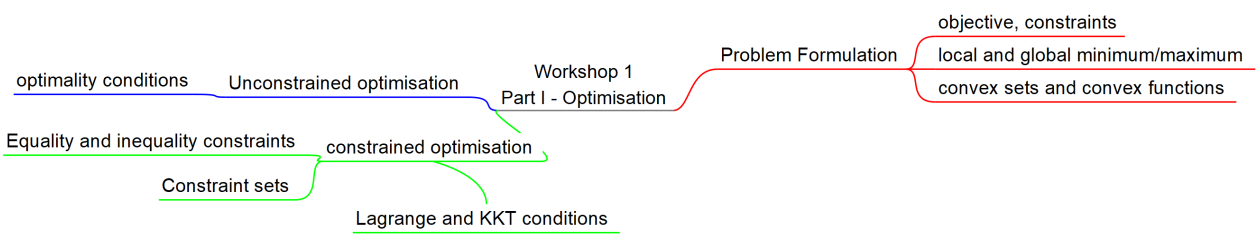
Table of Contents

Workshop 1 – Optimisation and Machine Learning [3 weeks]

1 Part I – Optimisation

1.1 Topics Covered

- Optimisation basics, formulation of optimisation problems
- Local/global minima/maxima
- Convex sets and functions
- Unconstrained optimisation and optimality conditions
- Constrained optimisation, equality and inequality constraints, constraint sets
- Lagrange theory and KKT conditions



1.2 Topic Notes

Another name for the field of “Optimisation” is “Mathematical Optimisation.” As the name indicates, optimisation is an area of applied mathematics. It is possible to study optimisation entirely from a mathematical perspective. However, engineers are interested in solving real-world problems in a principled way. Many engineering problems can be and are formulated as optimisation problems. In those cases, mathematical optimisation provides a solid theoretical foundation for solving them in a principled way.

In this workshop, you will learn how to formulate and solve optimisation problems in practice. This will give you a chance to connect theoretical knowledge and practical usage by doing it yourself. You will familiarise yourself with practical optimisation tools for Python. These are chosen completely for educational reasons (simplicity, accessibility, cost). While the underlying mathematics is timeless, optimisation software evolves with time, and can be diverse. Fortunately, once you learn one or two, it should be rather easy to learn others now and in the future because software designers often try to make it user friendly and take into account what people already know.

> In the future, you should consider learning serious optimisation software (https://en.wikipedia.org/wiki/List_of_optimization_software) for scalability and reliability. They can be complex and/or expensive but they get the job done for serious engineering. Learning such software takes a significant amount of time and is beyond the scope of this subject.

1.3 Workflow and Assessment

This subject follows a problem- and project-oriented approach. In this learning workflow, the focus is on solving practical (engineering) problems, which motivate acquiring theoretical (background) knowledge at the same time.

1.3.1 Objectives:

- Use these problems as a motivation to learn the fundamentals of optimisation covered in lectures.
- Learn how to formulate and solve optimisation problems in practice.
- Familiarise yourself with practical software tools used for optimisation.
- Solve optimisation problems using Python (and/or Matlab).
- Connect theoretical knowledge and practical usage by doing it yourself.

1.3.1.1 Common objectives of all workshops

Gain hands-on experience and learn by doing! Understand how theoretical knowledge discussed in lectures relates to practice. Develop motivation for gaining further theoretical and practical knowledge beyond the subject material.

> **Self-learning** is one of the most important skills that you should acquire as a student. Today, self-learning is much easier than it used to be thanks to a plethora of online resources.

1.3.2 Assessment Process

1. Follow the procedures described below, perform the given tasks, and answer the workshop questions **in this Python notebook itself! The resulting notebook will be your Workshop Report!**
2. Submit the workshop report at the announced deadline
3. Demonstrators will conduct a brief (5min) oral quiz on your submitted report in the subsequent weeks.
4. Your workshop marks will be a combination of the report you submitted and oral quiz results.

> **The goal is to learn**, NOT blindly follow the procedures in the fastest possible way! **Do not simply copy-paste answers (from Internet, friends, etc.). You can and should use all available resources but only to develop your own understanding. If you copy-paste, you will pay the price in the oral quiz!**

1.4 Convex Functions

Remember the definition of convex and concave functions from lecture slides. Functions are mathematical objects but they are used in engineering in very practical ways, for example, to represent the relationship between two quantities. Let's draw a function!

In [ ]:

```
%matplotlib notebook
import numpy as np
import matplotlib.pyplot as plt

# define function f(x)
def f(x):
    return 5*(x-1)**2

# define x and y
x = np.linspace(-20, 20, 100) # 100 equally spaced points on interval [-20,20]
y = f(x) # call function f(x) and set y to the function's return value

# Plot the function y=f(x)
plt.plot(x,y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('$y=5(x-1)^2$')
plt.show()
```

1.4.1 Practice Question

Plot one concave and one non-concave function of your choosing (preferably one in 2 dimensions and the other in 3 dimensions so that it can be visualised, check e.g. this tutorial (https://matplotlib.org/2.0.2/mpl_toolkits/mplot3d/tutorial.html) for hints). Provide their formulas below.

Hint: an interesting and well-known function is [Rosenbrock function].(https://en.wikipedia.org/wiki/Rosenbrock_function (https://en.wikipedia.org/wiki/Rosenbrock_function)) It is already built-in to Scipy optimize (https://docs.scipy.org/doc/scipy/reference/optimize.html#benchmark-problems) as a benchmark. *You can keep your answer simple and don't need to spend too much time on this practice question.*

In [ ]:

```
''' Answer as code here '''
```

Answer as text here

1.4.2 Curiousity Question

How would you determine whether a single or multi-variate continuously differentiable function is convex or not?

> Note that the question becomes very tricky if you have a **parametric** multivariate polynomial of degree four or higher!

> *[Optional]* An interesting paper (for those who wish to go deeper) http://web.mit.edu/~a_a_a/Public/Publications/convexity_nphard.pdf (http://web.mit.edu/~a_a_a/Public/Publications/convexity_nphard.pdf)

Answer as text here

1.4.3 Discussion Question

Why are convex optimisation problems considered to be easy to solve? Consider optimality conditions of unconstrained functions in your answer. Plot the first- and second-order derivative functions for one concave and one non-concave function (this time only in 2 dimensions) to further support your argument.

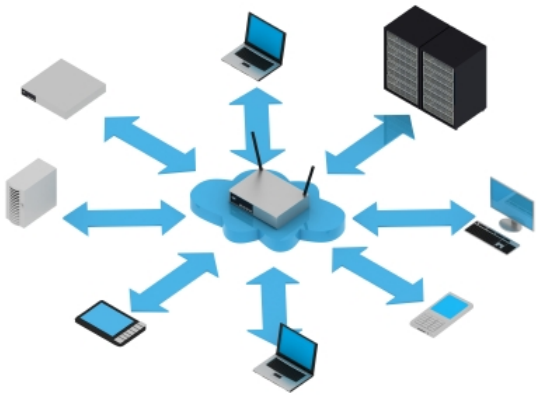
Answer as text here

In [ ]:

```
''' Answer as code here '''
```

1.5 Unconstrained Optimisation

1.5.1 Example: Aloha communication protocol



Aloha is a well-known random access or *MAC* (Media/multiple Access Control) communication protocol. It enables multiple nodes to share a broadcast channel without any additional signaling in a distributed manner. Unlike *FDMA* or *TDMA* (frequency or time-division multiple access), the channel is not divided into segments beforehand and collisions of packets due to simultaneous transmissions by nodes are allowed. In slotted Aloha, the nodes can only transmit at the beginning of time slots, which are kept by a global/shared clock. See Aloha (https://en.wikipedia.org/wiki/ALOHAnet#Slotted_ALOHA) for further background information.

1.5.2 Slotted Aloha Efficiency

For an N-node slotted Aloha system, where each node transmits with a probability p, the throughput of the system is given by

$$S(p) = Np(1-p)^{N-1}$$

1.5.3 Question

Formally define the optimisation problem to find the optimal probability p that maximises the throughput. Clearly identify the objective and decision variable(s). Is the objective convex or concave? Show through derivation. Find the optimality conditions for this problem.

Note that there is the constraint $0 \leq p \leq 1$ on probability p but we will ignore it for now.

Answer as text here

Answer as text here

In [ ]:

```
''' Answer as code here '''
```

1.5.4 Question

Find the optimal probability p for $N = 12$ nodes. Use an appropriate package from Scipy (https://docs.scipy.org/doc/scipy/reference/optimize.html). Cross-check your answer with a mathematical formula that you should derive by hand.

Hint: see examples and documentation for scalar case. (https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize_scalar.html#scipy.optimize.minimize_scalar)

In [ ]:

```
from scipy import optimize
''' Answer as code here '''
```

Answer as text here

1.6 Constrained Optimisation

Pyomo (http://www.pyomo.org/) is a Python-based tool for modeling and solving optimisation problems. Algebraic modeling languages (AMLs) like Pyomo are high-level languages for specifying and solving mathematical optimisation problems. Widely used commercial AMLs include AIMMS, AMPL, and GAMS.

Pyomo uses the following mathematical concepts that are central to modern modeling activities:

- Variables: These represent unknown or changing parts of a model (e.g., which decisions to take, or the characteristic of a system outcome).
- Parameters: These are symbolic representations for real-world data, which might vary for different problem instances or scenarios.

- Relations: These are equations, inequalities, or other mathematical relationships that define how different parts of a model are related to each other.

Pyomo supports an object-oriented design for the definition of optimisation models. A Pyomo model object contains a collection of modeling components that define the optimisation problem. The Pyomo package includes modeling components that are necessary to formulate an optimisation problem: variables, objectives, and constraints, as well as other modeling components that are commonly supported by modern AMLs, including index sets and parameters.

The pyomo online documentation (https://pyomo.readthedocs.io/en/stable/index.html) gives you an excellent starting point. There is also an entire book (https://www.springer.com/gp/book/9783319588193) for those who are interested.

See the instructions to install Pyomo, based on Pyomo's instructions using conda (https://pyomo.readthedocs.io/en/stable/installation.html#using-conda).

> **2021 Installation Problem for Windows:** Ipopt installation is a bit broken under Windows. All you need to do is copy the ipopt.exe we have provided in files (originally from pre-compiled sources, 3.11.1 64 or 32 bit version (https://www.coin-or.org/download/binary/Ipopt/)) and copy that to Anaconda's `\Library\bin` directory. This directory can be found using the command `where conda`. After copying ipopt.exe, close and reopen anaconda (and this notebook) before continuing.

Hint: to process Pyomo results, see working with Pyomo models (https://pyomo.readthedocs.io/en/stable/working_models.html), e.g. you can access Lagrange multipliers (duals) (https://pyomo.readthedocs.io/en/stable/working_models.html#accessing-duals) by passing an argument to solver.

Suggested exercise: verify your answer to Aloha optimisation above by formulating that problem as a Pyomo model.

1.7 Example: Non-Convex Optimisation

The Rosenbrock function (https://en.wikipedia.org/wiki/Rosenbrock_function) is a non-convex function, introduced by Howard H. Rosenbrock in 1960, which is used as a performance test problem for global optimisation algorithms. A two-variable, arbitrarily-constrained variant is

$$\min_{x \in \mathcal{A}} f(x) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2,$$

where the constraint set is defined by

$$\mathcal{A} := \{x \in \mathbb{R}^2 | x_2 \geq x_1 + 1,\ x_1 \in [-2, 3], x_2 \in [-2, 2]\}.$$

Let us use the following abstract Pyomo model (https://pyomo.readthedocs.io/en/stable/pyomo_overview/simple_examples.html) for this problem:

In [ ]:

```
# A Pyomo model for the Rosenbrock problem
import pyomo.environ as pyo
from pyomo.opt import SolverFactory

model = pyo.AbstractModel()
model.name = 'Rosenbrock'

# note boundaries of variables and initial condition $x_0=[-2,2]$
model.x1 = pyo.Var(bounds=(-2,3), initialize=-2)
model.x2 = pyo.Var(bounds=(-2,2), initialize=2)

def rosenbrock(model):
    f = (1.0-model.x1)**2 + 100.0*(model.x2 - model.x1**2)**2
    return f

def ineqconstr(model):
    return model.x2 >= model.x1+1

model.obj = pyo.Objective(rule=rosenbrock, sense=pyo.minimize)
model.constraint = pyo.Constraint(rule=ineqconstr)
```

This model can be solved in multiple different ways. Since this is a Pyomo AbstractModel, we must create a concrete instance of it before solving.

In [ ]:

```
import io

# declare a dummy file
dummy_file = io.StringIO() #we create a in-memory text stream

# create an instance of the problem
arosenbrockproblem = model.create_instance()
# this is to access Lagrange multipliers (dual variables)
arosenbrockproblem.dual = pyo.Suffix(direction=pyo.Suffix.IMPORT)

# define solver
opt = pyo.SolverFactory('ipopt') # we can use other solvers here as well

results = opt.solve(arosenbrockproblem, tee=True, logfile = "name.csv")

# show results

arosenbrockproblem.display(ostream=dummy_file)
#arosenbrockproblem.display(filename="output_file")  #alternatively write output to a file
#help(arosenbrockproblem.display)
```

In [ ]:

```
dummy_file.getvalue()
```

In [ ]:

```
arosenbrockproblem.display()
```

Now, let's see the Lagrange multipliers.

In [ ]:

```
def display_lagrange(instance):
    # display all duals
    print ("Duals")
    for c in instance.component_objects(pyo.Constraint, active=True):
        print ("   Constraint",c)
        for index in c:
            print ("      ", index, instance.dual[c[index]])

display_lagrange(arosenbrockproblem)
```

Display the solution directly

In [ ]:

```
def disp_soln(instance):
    output = []
    for v in instance.component_data_objects(pyo.Var, active=True):
        output.append(pyo.value(v))
        print(v, pyo.value(v))
    print (instance.obj, pyo.value(instance.obj))
    output.append(pyo.value(instance.obj))
    return output


disp_soln(arosenbrockproblem)
```

In [ ]:

```
results
```

Since this is a non-convex optimisation problem, the solver can only find local solutions! What happens if we change the starting point to $x_0 = [1.5, 1.5]$?

In [ ]:

```
arosenbrockproblem.x1 = 1.5
arosenbrockproblem.x2 = 1.5

results = opt.solve(arosenbrockproblem)

arosenbrockproblem.display()
```

In [ ]:

```
display_lagrange(arosenbrockproblem)
```

In [ ]:

```
disp_soln(arosenbrockproblem)
```

In [ ]:

```
dir(arosenbrockproblem)
```

1.8 Using random.seed() for pseudo-random number generation

We use Python or numpy random modules to generate random numbers or vectors/matrices. Random numbers and vectors are used a lot in various context, e.g. as asked above or in machine learning problems. They also cause a *reproducibility* problem. In this case, when your demonstrators run your code, they will get a different α vector each time and the results will be different each time. How can we prevent that? Python *random.seed* addresses this problem.

If you call *random.seed(seedvalue)* every time before calling the random number generator, you will get the same "random" number. Change the seed and the number will change as well. This makes it pseudo-random (https://en.wikipedia.org/wiki/Pseudorandomness) which ensures reproducibility.

See this nice article for basics (https://pynative.com/python-random-seed/). To generate random vectors and matrices numpy (https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.random.seed.html#numpy.random.seed) is very useful.

In [ ]:

```
import random
import numpy as np

print ("Random numbers with a given seed")
random.seed(1215151)
print (random.random())
random.seed(1215151)
print (random.random()) # same seed, same number!
random.seed(1215151)
print (random.random()) # same seed, same number!
random.seed(5298496496)
print (random.random()) # different seed, different number!
random.seed(5298496496)
print (random.random()) # repeat!

print (random.random()) # no seed, purely random number
print (random.random()) # no seed, purely random number

np.random.seed(4198494)
print(np.random.rand(3,2))
np.random.seed(4198494)
print(np.random.rand(3,2))
# now without seed, most probaby it will be different!
print(np.random.rand(3,2))
```

1.8.1 Important Note on Random Number/Vector Generation

Each group has to use a different number seed (which is an arbitrary number as illustrated above) and groups cannot share seeds. The pseudo-randomness is used here to create diversity. Otherwise, if groups use the same seed, you would lose points in assessment

1.9 Example Economic Dispatch in Power Generation

The problem is formulated as

$$\min_{P} \sum_{i=1}^{N} c_i P_i$$

$$\text{subject to } P_{i,max} \geq P_i \geq 0, \ \forall i, \ \text{and} \ \sum_{i=1}^{N} P_i = P_{demand}$$

Here, $P_1, \dots P_N$ are the power generated by Generators $1, \dots, N$, c_i is the per-unit generation cost of the i-*th* generator, and P_{demand} is the instantaneous power demand that needs to be satisfied by aggregate generation. More complex formulations take into account transmission, generator ramp-up and down constraints, and reactive power among other things.

1.9.1 Question: Economic Dispatch

Let us get inspired from generation in Victoria with $N = 12$ biggest generators that have more than 200MW capacity. Choose their maximum generation randomly or from the Victoria generator report if you wish to be more realistic. Generate a random cost vector c varying between $10 - 50$ AUD per MWh (use a group-specific random seed). *(Optionally, you can search and find how much different generation types cost if you are interested and add a bit of random noise to it)*. Let the demand be $P_{demand} = 5000MW$.

Solve this simplified economic dispatch (https://en.wikipedia.org/wiki/Economic_dispatch) problem defined above. The resulting merit order (https://en.wikipedia.org/wiki/Merit_order) is the generation that would have been if there was no NEM (electricity market).

1. Solve the problem using *Pyomo*.
2. What type of an optimisation problem is this? Briefly explain.

> Further information: you can find more about Australian wholesale electricity market and generation at https://www.aemo.com.au/ (https://www.aemo.com.au/) See also this NEM overview introductory document (right click to download) (./files/National_Electricity_Market_Fact_Sheet.pdf) and the Victoria generator report as of January 2019 (files/Generation_Information_VIC_January_2019.xlsx).

Note: *if you are in the minority of people who have problem installing pyomo, then you can use scipy or even Matlab.*

Answer as text here

In [ ]:

```
''' Answer as code here '''
```

1.10 Example: Waterfilling in Communications

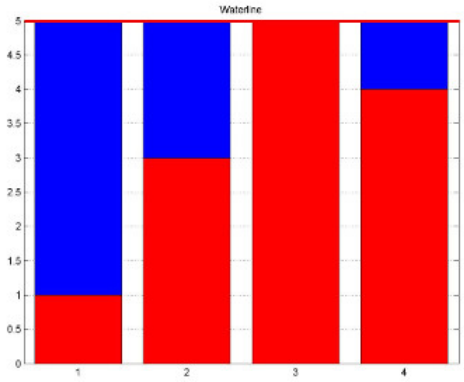
by Robert Gowers, Roger Hill, Sami Al-Izzi, Timothy Pollington and Keith Briggs. From the book by Boyd and Vandenberghe, Convex Optimization, Example 5.2 page 245.

$$\min_x \sum_{i=1}^{N} -\log(\alpha_i + x_i)$$

$$\text{subject to } x_i \geq 0,\ \forall i,\ \text{and } \sum_{i=1}^{N} x_i = P$$

This problem arises in information/communication theory, in allocating power to a set of n communication channels. The variable x_i represents the transmitter power allocated to the *i-th* channel, and $\log(\alpha_i + x_i)$ gives the capacity or communication rate of the channel, where $\alpha_i > 0$ represents the floor above the baseline at which power can be added to the channel. The problem is to allocate a total power of one to the channels, in order to maximize the total communication rate.

This can be solved using a classic water filling algorithm (https://en.wikipedia.org/wiki/Water_filling_algorithm).



1.10.1 Question

1. Is the problem in the Example above convex? Formally explain/argue why or why not. What does this imply regarding the solution?
2. Solve the problem above for $N = 8$ and a randomly chosen α vector (use a group-specific random seed). You *can* use Pyomo for this. Cross-check your answer with another software (package), e.g. Matlab or Scipy.
3. Write the Lagrangian, KKT conditions, and find numerically the Lagrange multipliers associated with the solution (using the software package/function). Which constraints are active? Explain and discuss briefly.

Answer as text here

In [ ]:

```
''' Answer as code here '''
```

1.11 Example: Power Control in Wireless Communication

Adapted from Boyd, Kim, Vandenberghe, and Hassibi, "A Tutorial on Geometric Programming (https://web.stanford.edu/~boyd/papers/pdf/gp_tutorial.pdf)."

The power control problem in wireless communications (http://winlab.rutgers.edu/~narayan/PAPERS/PC%20for%20Wireless%20Data.pdf) aims to minimise the total transmitter power available across N trasmitters while concurrently achieving good (or a pre-defined minimum) performance.

The technical setup is as follows. Each transmitter i transmits with a power level P_i bounded below and above by a minimum and maximum level. The power of the signal received from transmitter j at receiver i is $G_{ij}P_j$, where $G_{ij} > 0$ represents the path gain (often loss) from transmitter j to receiver i. The signal power at the intended receiver i is $G_{ii}P_i$, and the interference power at receiver i from other transmitters is given by $\sum_{k\neq i} G_{ik}P_k$. The (background) noise power at receiver i is σ_i. Thus, the *Signal to Interference and Noise Ratio (SINR)* of the ith receiver-transmitter pair is

$$S_i = \frac{G_{ii}P_i}{\sum_{k\neq i} G_{ik}P_k + \sigma_i}.$$

The minimum SINR represents a performance lower bound for this system, $S^{\min}$.

The resulting optimisation problem is formulated as

$$\begin{aligned} \min_P \quad & \sum_{i=1}^{N} P_i \\ \text{subject to} \quad & P^{min} \leq P_i \leq P^{max}, \ \forall i \\ & \frac{G_{ii}P_i}{\sigma_i + \sum_{k\neq i} G_{ik}P_k} \geq S^{min}, \ \forall i \end{aligned}$$

1.11.1 Question: Wireless Power Control

Let $N = 10$, $P^{min} = 0.1$, $P^{max} = 5$, $\sigma = 0.2$ (same for all). Create a random path loss matrix G, where off-diagonal elements are between 0.1 and 0.9 and the diagonal elements are equal to 1.

1. Write down the Langrangian and KKT conditions of this problem.
2. Solve the problem first with $S^{min} = 0$ using *Pyomo*. Plot the power levels and SINRs that you obtain.
3. What happens if you choose an S^{min} that is larger? Solve the problem again and document your results. What happens if you choose a very large S^{min}? Observe and comment.

Note: *if you are in the minority of people who have problem installing pyomo, then you can use scipy or even Matlab.*

Answer as text here

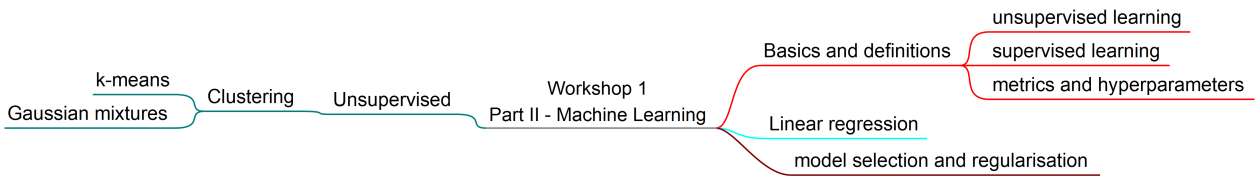
In [ ]:

```
''' Answer as code here '''
```

2 Part II - Machine Learning (ML)

2.1 Topics Covered in Part 2

- ML basics and definitions, supervised and unsupervised learning
- Linear regression, model selection, and regularisation
- Clustering, k-means, and Gaussian Mixture Models
- *Remaining topics will be covered in Workshop 2!*



2.2 Part 2 Notes

Optimisation is widely used in engineering (practice and research) today. That was not always so. I expect that in the future machine learning will be as prevalently used in engineering as optimisation is used today. The arguments in favour of it are (a) increasingly more powerful computing (b) lots of data (c) decreasing storage and computing costs. Machine learning benefits substantially from these trends. We will hopefully see together how engineering world will evolve in this century.

In this workshop, you will learn how to solve machine learning problems in practice and apply common algorithms to various data sets. Doing this yourself will give you a chance to connect theoretical knowledge and practical usage. We will start with simple, easy-to-visualise (2D) data sets so that concepts become clear. More interesting problems and data will be posed as open-ended (and optional) problems.

You will also familiarise yourself with machine learning libraries of Python, which is the de-facto language for ML these days. Still, the tools and data are chosen completely for educational reasons (simplicity, accessibility, cost). There are and will be better ML frameworks and more complex data sets but it is not realistic to cover all. Due to time limitations, we unfortunately do not focus on a big topic in this workshop and subject: data science

(https://study.unimelb.edu.au/find/courses/graduate/master-of-data-science/what-will-i-study/). You should not get the wrong impression from the nice, cleaned-up data sets you are given in this workshop. In real life, data is messy and more than half of data science is about preparing data itself.

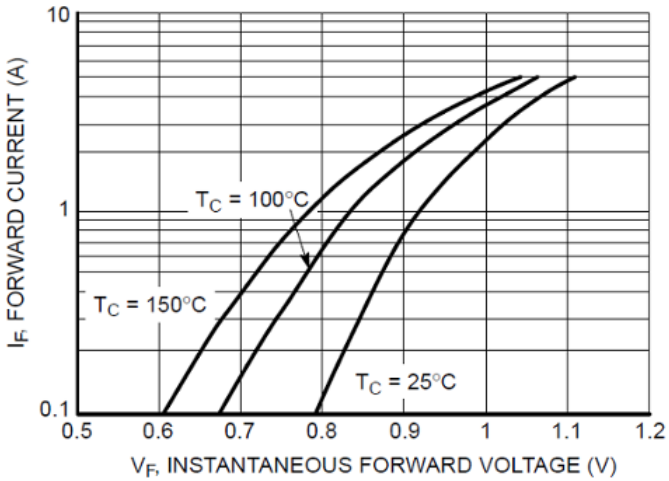
> In the future, you should consider learning additional ML software packages and libraries. Finding the right tool for the right job is an important skill obtained through knowledge and experience. I would also recommend learning more about data preparation and analysis. The popular Pandas (https://pandas.pydata.org/) library, which we briefly use, makes a good starting point.

2.3 Part 2 Objectives

- Use these problems as a motivation to learn the fundamentals of machine learning covered in lectures.
- Gain hands-on experience with basic machine learning paradigms.
- Familiarise yourself with some of the practical software tools used for machine learning.
- Solve basic machine learning problems using Python Scipy and Scikit-learn.
- Connect theoretical knowledge and practical usage by doing it yourself.

2.4 Linear Regression, Overfitting, and Regularisation

2.4.1 Example: Using curve fitting to model Diode characteristics



The diagram above shows the I-V curve of a diode (https://en.wikipedia.org/wiki/Diode) widely used in electronic circuits, see 1N4001-D spec sheet (right click to download) (1N4001-D.pdf).

We can use regression to model the I-V curve of this diode at 25˚C.

Using the nice tool, WebPlotDigitizer (https://automeris.io/WebPlotDigitizer/), a small and clean data set is generated and stored in csv format (https://en.wikipedia.org/wiki/Comma-separated_values). We now use the famous pandas library (https://pandas.pydata.org/) to read the *csv* file.

In [ ]:

```
%matplotlib notebook
import pandas as pd
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt
import matplotlib

dataset=pd.read_csv('files/diode_dataset.csv', names=['Vf', 'If'])
# Note that if you don't put names to csv or into the function as above,
# pandas ignores the first row in calculations!
dataset.head()
```

In [ ]:

```
plt.figure()
plt.plot(dataset.values[:,0], dataset.values[:,1])
plt.xlabel('Voltage, V')
plt.ylabel('Current, I')
plt.title('Diode I-V')
plt.grid()
plt.show()
```

Note that the figure above is convex but the one above was looking concave! Can you see why?

2.4.2 Question: Regression

Now let's consider the linear model $I = f(V) = a + bV$, for $a, b \in \mathbb{R}$ for simplicity.

We can find the best (a, b) that minimises the error between the N data points (I_j, V_j) and this linear model by solving the optimisation problem

$$\min_{a,b} \sum_{j=1}^{N} (I_j - (a + bV_j))^2$$

This is equivalent *in spirit* to what the machine learning libraries such as *scikit-learn (sklearn)* do to solve this problem! As you will repeatedly see, there is a deep and close relationship between optimisation and many learning methods.

1. Find the optimal a, b pair by solving the unconstrained optimisation problem above using *pyomo*. Using the formula and the parameters, a, b, you derived, plot the linear I-V curve with the additional constraint $I \geq 0$ or $\max(I, 0)$.
2. First, fit a linear model (https://scikit-learn.org/stable/modules/linear_model.html) using "linear_model.LinearRegression()". Plot the result, find the coefficients, and calculate the mean squared error (MSE).
3. Next, fit a polynomial model (https://scikit-learn.org/stable/modules/linear_model.html#polynomial-regression-extending-linear-models-with-basis-functions) of second degree, i.e. a quadratic model. Plot the model, find the coefficients, and calculate MSE. Interpret and discuss the results.
4. Add a regularisation term, i.e use ridge regression (https://scikit-learn.org/stable/modules/linear_model.html#ridge-regression). Do you observe an improvement? Discuss.
5. Try a couple of higher order models (e.g. 4, 6) without regularisation, and provide the results as before. What do you observe when you compare the coefficients? Find validation and training errors for the models and discuss/interpret your results.

2.4.2.1 Hints

1. you will need to use pipelines (https://scikit-learn.org/stable/modules/compose.html#pipeline). To access coefficients use `model.named_steps['linearregression'].coef_` or `model.named_steps['ridge'].coef_`
2. the `train_test_split` function provides a very convenient way of shuffling the data and dividing it into training and test sets (3:1 default ratio), see https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html)

In [ ]:

```
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split

# full data in correct form for sklearn
Vfulldata = np.array(dataset.values[:,0]).reshape(-1,1) # reshape needed for sklearn functions
Ifulldata = np.array(dataset.values[:,1]).reshape(-1,1)

# split into training and test sets
Vtrain, Vtest, Itrain, Itest = train_test_split(Vfulldata, Ifulldata)
```

Answer as text here

In [ ]:

```
''' Answer as code here '''
```

2.5 Clustering and Gaussian Mixtures

Unsupervised learning is all about data. We will use first the famous two moon data set to practice a little bit and digest some of the fundemental concepts. Since two moons data inherently has two clusters (each moon as a cluster), we can use this as a ground truth (https://en.wikipedia.org/wiki/Ground_truth). In most real problems, we don't have this luxury of having the ground truth at hand!

Note that Scikit Learn does not have its own global random state but uses the numpy random state (https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.random.seed.html#numpy.random.seed) instead. See the code below.

In [ ]:

```
from sklearn import cluster, datasets, mixture
from sklearn.cluster import KMeans
from sklearn.utils import shuffle
from sklearn import decomposition
from sklearn.mixture import GaussianMixture
from sklearn import metrics

# Set a random seed as you did in optimisation workshop by uncommenting the line below!
#np.random.seed(Put here a group-specific number!)

noisy_moons = datasets.make_moons(n_samples=200, noise=0.05)
X = noisy_moons[0] # data points
y = noisy_moons[1] # 0, 1 labels of class, 50 each - giving us the ground truth

order_ind = np.argsort(y) # order labels, 50 each class
X1 = X[order_ind[0:100]]   # class 1
X2 = X[order_ind[101:200]] # class 2

# Plot data
plt.figure()
plt.scatter(X1[:,0], X1[:,1], color='black')
plt.scatter(X2[:,0], X2[:,1], color='red')
plt.show()
```

2.5.1 Question: K-means clustering

1. Use sklearn's k-means clustering algorithm to divide the two moon data given above (X) into two clusters. Plot the result and show the cluster centres that you found.
2. Experiment with different starting points (`init='random'`) and number of clusters, e.g. 3, 4, 5. Write your observations and interpret them using your theoretical knowledge from lectures and books.

In [ ]:

```
''' Answer as code here '''
```

Answer as text here

2.5.2 Question: GMMs as density estimators

1. Now use a Gaussian Mixture Model (GMM) for clustering the same two moon data. Try two clusters and plot your results. GMMs also provides you probabilities (of a sample belonging to a cluster). Print those of a few samples.
2. Increase the number of components of your GMM model. What do you observe? Use a metric to choose the number of components in a principled way. *Hint: check BIC (https://en.wikipedia.org/wiki/Bayesian_information_criterion) or AIC (https://en.wikipedia.org/wiki/Akaike_information_criterion)*
3. It is maybe better to use GMM as a **generative model**! Generate 200 brand new samples from a trained GMM with your choice of parameters and plot your results. Discuss your findings!

In [ ]:

```
''' Answer as code here '''
```

Answer as text here

3 Workshop Assessment Instructions

You should complete the workshop tasks and answer the questions within the allocated session! **Submission deadline is usually end of the last week of the workshop. Please check Canvas for the exact deadline!**

It is **mandatory to follow all of the submissions guidelines** given below. *Don't forget the Report submission information on top of this notebook!*

1. The completed Jupyter notebook and its Pdf version (you can simply print-preview and then print as pdf from within your browser) should be uploaded to the right place in Canvas. *It is your responsibility to follow the announcements!* **Late submissions will be penalised (up to 100% of the total mark depending on delay amount)!**
2. Filename should be “ELEN90088 Workshop **W: StudentID1-StudentID2** of session **Day-Time**", where **W** refers to the workshop number, **StudentID1-StudentID2** are your student numbers, **Day-Time** is your session day and time, e.g. *Tue-14*.
3. Answers to questions, simulation results and diagrams should be included in the Jupyter notebook as text, code, plots. *If you don't know latex, you can write formulas/text to a paper by hand, scan it and then include as image within Markdown cells.*
4. Please submit your report individually. Partners can submit the same report.

3.1 Workshop Marking

- **Each workshop has 15 points corresponding to 15% of the total subject mark inclusive individual oral examination.** You can find the detailed rubrics on Canvas.

- Individual oral quizzes will be scheduled within the next two weeks following the report submission. They will be during workshop hours. Therefore, it is important that you attend the workshops!
- The individual oral examination will assess your answers to workshop questions, what you have done in that workshop, and your knowledge of the subject material in association with the workshop.

3.1.1 Additional guidelines for your programs:

- Write modular code using functions.
- Properly indent your code. But Python forces you do that anyway ;)
- Heavily comment the code to describe your implementation and to show your understanding. No comments, no credit!
- Make the code your own! It is encouraged to find and get inspired by online examples but you should exactly understand, modify as needed, and explain your code via comments. If you resort to blind copy/paste, you will certainly not do well in the individual oral quizzes.

Report Submission Information (must be completed before submitting report!)

- Student 1 Full Name and Number:
- Student 2 Full Name and Number:
- Workshop day: e.g., Wednesday
- Workshop time: e.g., 12pm

In [ ]:

```
```