Class 4, Problem Set 1

UNIVERSITY OF COPENHAGEN

Introduction to Programming and Numerical Analysis

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from scipy import optimize
```

Plan for today

1. Brush up on important concepts/syntax
    - Creating functions
    - Figure syntax
    - SciPy.optimize
2. Work on PS1

UNIVERSITY OF COPENHAGEN

Functions

Never do something twice: This is where functions come in handy (and for a lot of other reasons)

```
def name_of_function(input):
    **do something with the input**
    return output
```

In [2]:

```
# Example
def f(x):
    fx = np.sin(x)+0.05*x**2
    return fx

print(f(6))
```

```
1.5205845018010742
```

Functions: Best practice

UNIVERSITY OF
COPENHAGEN

In [4]:

```
def f(
    x:float
    ) -> float:

    '''Calculate a function value given input
    Args:
        x (float): Input
    Returns:
        float: Function value
    '''

    fx = np.sin(x)+0.05*x**2

    return fx
```

In [5]:

```
for i in range(-4,5):
    print(f'The function value at {i} is: {f(i)}')
```

```
The function value at -4 is: 1.5568024953079282
The function value at -3 is: 0.3088799919401328
The function value at -2 is: -0.7092974268256818
The function value at -1 is: -0.7914709848078965
The function value at 0 is: 0.0
The function value at 1 is: 0.8914709848078965
The function value at 2 is: 1.1092974268256817
The function value at 3 is: 0.5911200080598672
The function value at 4 is: 0.04319750469207184
```

Figure syntax

Probably different from what you know (SAS, Stata etc.)

... get used to it...

In [6]:

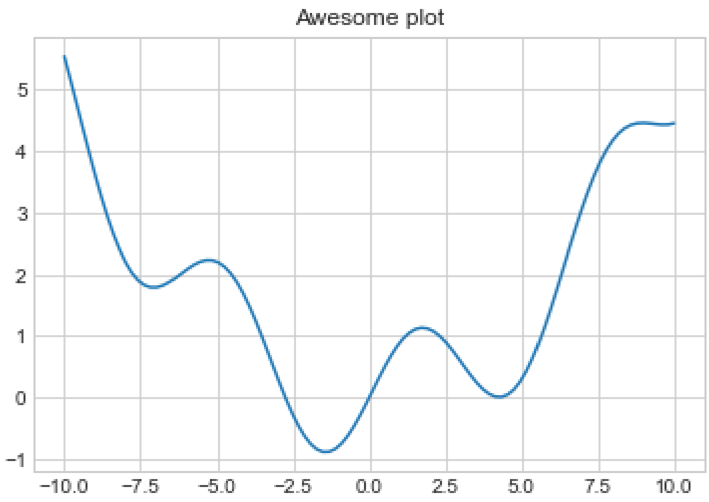
```
# Return evenly spaced numbers over a specified interval
x = np.linspace(-10,10,num=100)

# Initialize canvas - From documentation: "Unique identifier for the figure"
fig1 = plt.figure() # Now fig1 object is a Matplot figure

# From documentation: add_subplot(nrows, ncols, index, **kwargs)
ax1 = fig1.add_subplot(1,1,1)

# Choose method: .plot(), .hist(), .plot_surface() etc.
ax1.plot(x, f(x))

# Customize
ax1.set_title('Awesome plot'); #; suppress print
```

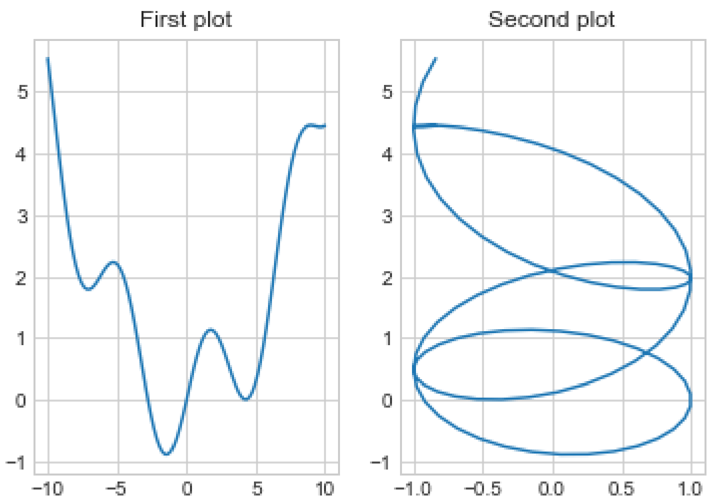


In [7]:

```
fig2 = plt.figure()

ax2_1 = fig2.add_subplot(1,2,1) # set
ax2_1.plot(x, f(x))
ax2_1.set_title('First plot')

ax2_2 = fig2.add_subplot(1,2,2)
ax2_2.plot(np.cos(x), (f(x)))
ax2_2.set_title('Second plot');
```



SciPy.optimize

UNIVERSITY OF COPENHAGEN

- Module for optimizing - more precise than 'just' looping through combinations
- One problem can be solved in different ways - don't let it knock you out
- Remember that we minimize! So if your maximizing your objective should be negative

In [9]:

```python
# Initial guess
x_guess = 0

# Objective function:
objective_function = lambda x: f(x)

list_of_bounds = [(np.min(x),np.max(x)), (np.min(x),0), (0,np.max(x))]

# SciPy
for b in list_of_bounds:
    opt = optimize.minimize_scalar(objective_function
                                   , x_guess
                                   , method='bounded'
                                   , bounds=b
                                  )

    # Unpack results
    x_best_scipy = opt.x
    f_best_scipy = opt.fun
    # Print
    print('minimum', f_best_scipy)
    print(f'Minimum function value is {f_best_scipy:.2f} at x = {x_best_scipy:.8f}\n')
```

```
minimum -0.887862826573322
Minimum function value is -0.89 at x = -1.42755262

minimum -0.887862826573621 9
Minimum function value is -0.89 at x = -1.42755138

minimum 0.007912876341589659
Minimum function value is 0.01 at x = 4.27109533
```

SciPy.optimize

- You will be using `minimize` and **not** `minimize_scalar`
- What method to choose?

SciPy.optimize

- You will be using `minimize` and **not** `minimize_scalar`
- What method to choose?

In [11]:

```
opt = optimize.minimize(objective_function
                        , x_guess
                        #, method='Nelder-Mead'
                        , method='Newton-CG'
                        #, method='BFGS'
                   )
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
<ipython-input-11-9507ed7c4a29> in <module>
----> 1 opt = optimize.minimize(objective_function
      2                         , x_guess
      3                         #, method='Nelder-Mead'
      4                         , method='Newton-CG'
      5                         #, method='BFGS'

~/opt/anaconda3/lib/python3.8/site-packages/scipy/optimize/_minimize.py in
minimize(fun, x0, args, method, jac, hess, hessp, bounds, constraints, to
l, callback, options)
    612         return _minimize_bfgs(fun, x0, args, jac, callback, **opti
ons)
```

```
    613     elif meth == 'newton-cg':
--> 614         return _minimize_newtoncg(fun, x0, args, jac, hess, hessp, callback,
    615                                   **options)
    616     elif meth == 'l-bfgs-b':

~/opt/anaconda3/lib/python3.8/site-packages/scipy/optimize/optimize.py in _minimize_newtoncg(fun, x0, args, jac, hess, hessp, callback, xtol, eps, maxiter, disp, return_all, **unknown_options)
   1673     _check_unknown_options(unknown_options)
   1674     if jac is None:
-> 1675         raise ValueError('Jacobian is required for Newton-CG method')
   1676     fhess_p = hessp
   1677     fhess = hess

ValueError: Jacobian is required for Newton-CG method
```

	Newton	BFGS	BHHH	Nelder-Mead	Steepest Descent
`method`	User written / CG	`BFGS`	CG	`Nelder-Mead`	User written
Option	–	[default]	Provide user-written Hessian		
Gradient used	✓	✓	✓	÷	✓
Hessian used	✓	✓	✓	÷	÷
Step	$f'(\cdot)/f''(\cdot)$	$f'(\cdot)/f''(\cdot)$	$f'(\cdot)/f''(\cdot)$	Heuristic	$\gamma f'(\cdot)$
Hessian	Numeric	Iterative updating	Outer product	Not used	Not used
Best for	Nice f but weird Hessian	Nice f	Likelihood estimation	Nasty f	Non-convex or non-quadratic f
Iterations	Medium	Few	Few	Many	Many
Globalization	Line search	Line search	Line search	n.a.	Line search