Python for Machine Learning

Basic Optimization

CE 5331 Machine Learning for Civil Engineers

Venki Uddameri, Ph.D. , P.E.

Recap and Goals

- Installed Python and Anaconda Environments
- Introduction to Python
  - Setting working directory
  - Adding comment lines
    - Docstrings
- Introduction to Pandas
  - Reading a csv
  - Extracting columns (attributes)
  - Extracting rows
  - Obtaining summary measures
- Basic graphing and charting in Python
  - Matplotlib package
- Scipy
  - Interpolation
  - Kernel Density Functions
  - Integration (1D)
  - Integration (2D)
- Control Statements
  - If, if-elif-else, if-else
  - For loop
  - While loop
  - Use of Boolean operators
- Functions
  - Passing inputs
  - Lambda functions
  - Pass by object reference
- Numpy
  - Matrix Calculations
  - Vectorization

Goal of this module is to explore Optimization Methods in Python

Optimization

Optimization is the cornerstone of machine learning models

Model parameters have to be obtained via optimization

The minimization of residual sum of square (RSS) or maximization of the log-likelihood functions are two basic parameter estimation procedures

Python has several optimization routines

Unconstrained and constrained optimization problems

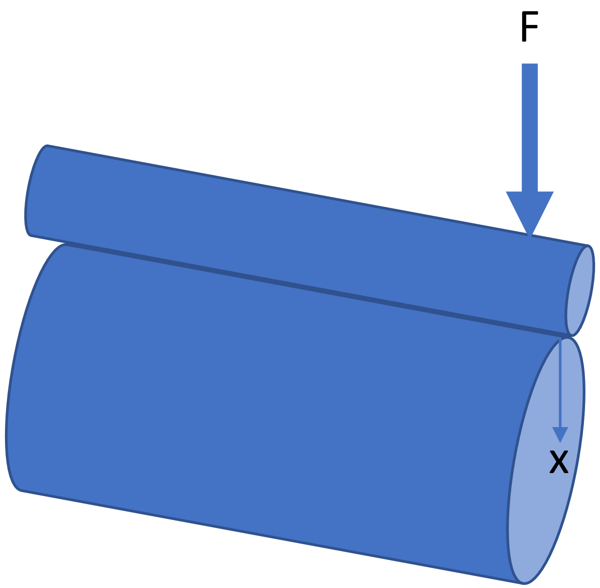
Evolutionary algorithms

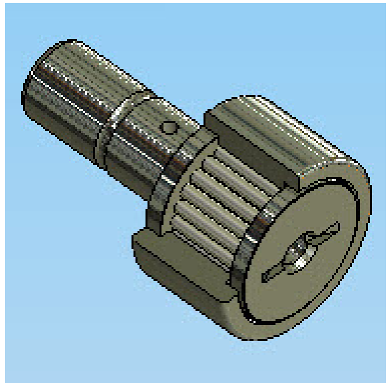
In this lecture, I will introduce you to some basic optimization routines in Python using **scipy** package
Optimization routines can be found in other packages too (e.g., scikit learn skopt module)

Unconstrained Minimization – Single Variable

- Roller bearings are subject to fatigue failure cause by large contact loads
- The location of the maximum stress along the x axis can be obtained by maximizing the function:

$$f(x) = \frac{0.4}{\sqrt{1+x^2}} - \sqrt{1+x^2}\left(1 - \frac{0.4}{\sqrt{1+x^2}}\right) + x$$





Answer
1.05 units from the top of the bearing

Note: Maximizing a f(x) is the same as minimizing –f(x)

Unconstrained Minimization – Single Variable

- Steps:
  - Import the minimize function from scipy.optimize module
  - Write the function to be minimized
  - Create a vector of initial guesses
  - Pass the function and the initial guesses to the minimize function
  - Check for convergence and accuracy of the results

```
# Scipy Optimization Examples
# Venki Uddameri 1/18/2020
import os
import numpy as np
from scipy.optimize import minimize # generic module for minimization
os.chdir('D:\\Dropbox\\000CE5333Machine Learning\\Module9\\Codes')

# Define function to be mininized
def funcx(x):
    """ function to minimize to find bearing failure"""
    a = 1 + x**2
    b = np.sqrt(a)
    fx = 0.4/b - b*(1-0.4/a) + x
    fx = -1*fx
    return(fx)

# Minimize the function
x0 = 1 # initial guess
res = minimize(funcx, x0, method='nelder-mead',  # call minimize function
               options={'xatol': 1e-8, 'disp': True})
res.x # Wrie the result to the console
```

Unconstrained Optimization

- The minimize function provides several choices
  - Nelder-Mead simplex is a fairly simple approach that works for well-behaved functions
- There are several other more sophisticated algorithms
  - Broyden-Fletcher-Goldfarb-Shanno algorithm (method='BFGS’)
  - Newton-Conjugate-Gradient algorithm (method='Newton-CG’)
  - Trust-Region Newton-Conjugate-Gradient Algorithm (method='trust-ncg’)
- Unconstrained minimization of a single variable can also be carried out using ‘Brent’ method
  - Unconstrained minimization - scalar (method='brent’)

Some methods require Hessian Matrix to be supplied which adds complexity but improves convergence

For Additional Details Refer to: https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html

Optimization - Regression

- The Greenshields model provides a relationship between mean traffic speed and density in an uninterrupted section as follows:

Free Speed

Density

$$v = v_f - \left(\frac{v_f}{k_j} \right) k$$

Mean Speed

Jam Density

Empirical Data is used to calibrate the Greenshields model

Notice the linear relationship between speed and density

Linear Regression is used when there are more data than unknowns

The unknowns are obtained in a best-fit sense

The sum of squared residuals (SSR) is minimized to obtain the best fit parameters

Fit the Greenshields Model using the rural traffic dataset provided to you (ruraldensityspeed.csv)

Modeling Approach

- Read the data in
  - Pandas library
- Write a function to calculate the SSE
- Specify initial guesses for A and B
- Minimize the SSE function to find optimal values of A and B

$$v = v_f - \left(\frac{v_f}{k_j}\right) k$$

Original Model

$$v_{pred} = A + Bk + e$$

Regression Form
(A and B are unknown coefficients)

$$e_i = v_{obs,i} - v_{pred,i}$$

Error term

Objective Function minimizes the sum of squared error term

$$\boxed{SSE = \sum_{i=1}^{N} e_i^2 = \sum_{i=1}^{N} (v_{obs,i} - v_{pred,i})^2 = \sum_{i=1}^{N} (v_{obs,i} - [A + Bx_i])^2}$$

This is also referred to as the loss function in Machine Learning Literature

Linear Regression using Unconstrained Optimization

Slope = -0.53
Intercept = 62.56

$$v = v_f - \left(\frac{v_f}{k_j}\right) k$$

Free Speed
Density
Mean Speed
Jam Density

Therefore – Free speed = 62.65 mph and Jam Density is 118.47 vehicles/mile/lane

You can also use linregress function in scipy stats module to perform linear regression

```
# Use scipy stats model to perform linear regression
# Now you can extract statistics as well
from scipy import stats
slope, intercept, r_value, p_value, std_err =
stats.linregress(k,vobs)
round(slope,2), round(intercept,2)
```

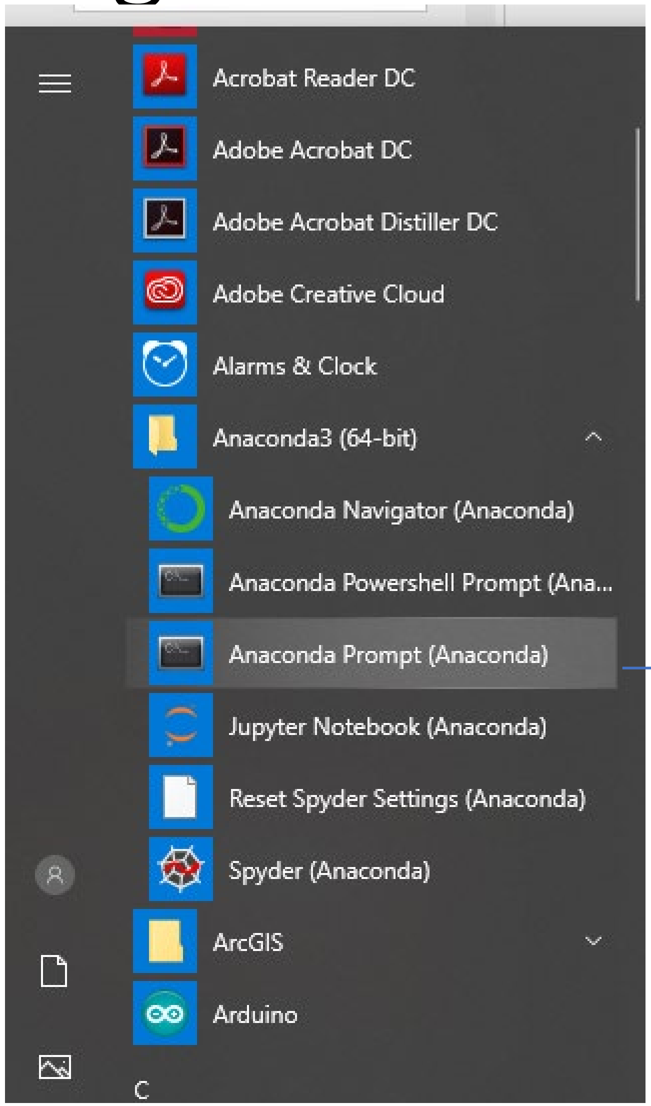
We will study linear regression in greater depth in the course

```
# Linear Regression using Unconstrained Optimization
# Venki Uddameri, TTU
# Step 1: Load Libraries
import os
import numpy as np
import pandas as pd
from scipy.optimize import minimize

# Set working directory
os.chdir('D:\\Dropbox\\000CE5333Machine Learning\\Module9\\Codes')

# Read data from csv file and extract variables
a = pd.read_csv('ruraldensityspeed.csv')
vobs = a['Speed']
k = a['Density']

# Define function for computing SSE
def funsse(A,k,vobs):
    pred = A[0] + A[1]*k
    err = (vobs-pred)**2
    sse = np.sum(err)
    return(sse)
# Call minimize functions
init = (1,1) # Starting values for slope and intercept
res = minimize(funsse,init,method='Nelder-Mead',args=(k,vobs,))
res.x # Write slope and intercept to the console
```

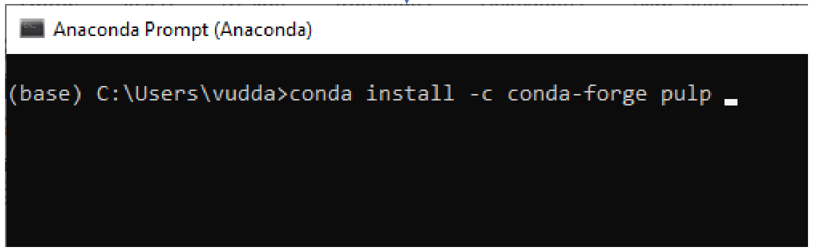
Linear Programming

- Python has a couple of options for solving linear programs
  - scipy
  - puLP
    - This library does not come preinstalled with conda
    - But you can easily install it

PuLP is an LP modeler written in python. PuLP can generate MPS or LP files and call GLPK, COIN CLP/CBC, CPLEX, and GUROBI to solve linear problems.



Installing pulp package



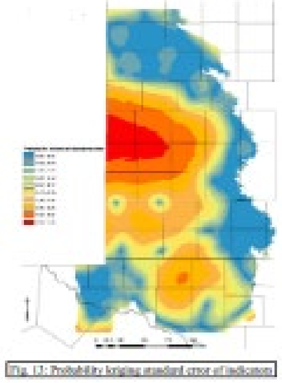
Accept Defaults

Constrained Linear Programming

- To prolong the useful life of Ogallala Aquifer a farmer is Texas is seeking to blend ‘brackish water’ from the underlying Dockum HSG to grow corn
- What is the **minimum** amount of water that the Farmer needs to draw from the Ogallala Aquifer during a season?

Parameters	Ogallala	Dockum	Crop Requirement
Well Yield (gpm)	200	75	
TDS (mg/L)	350	1800	700
SAR	5	12	10
Growing Season			6 month
Irrigation Water Requirement			234 gpm

Dockum Hydrostratigraphic Unit



Optimization Model

Governing Equations

$$Min : Q_o$$

Minimize Production from the Ogallala Aquifer

$$Q_o \leq Q_{o,wc}$$
$$Q_d \leq Q_{d,wc}$$

Well Yield Constraints

$$Q_d\,(TDS_d - TDS_s) + Q_o\,(TDS_o - TDS_s) \leq 0$$
$$Q_d\,(SAR_d - SAR_s) + Q_o\,(SAR_o - SAR_s) \leq 0$$

Crop Water Quality Constraints

$$Q_o + Q_d \geq Q_{irr}$$

Irrigation Water Requirements

$$Q_o,\ Q_d \geq 0$$

Non-Negativity Constraints

Parameterized Equations

$$Min : Q_o$$

Min Ogallala Production

$$Q_o \leq 200$$
$$Q_d \leq 75$$

Well Yield Constraints

$$1100Q_d - 350Q_o \leq 0$$
$$2Q_d - 5Q_o \leq 0$$

Crop Water Quality Constraints

$$Q_d + Q_r \geq 234$$

Irrigation Water Requirements

$$Q_o,\ Q_d \geq 0$$

Non-Negativity Constraints

We shall use the puLP modeler for solving the LP problem

puLP modeling Steps

- Import pulp library
- Create a problem variable
  - Specify if the problem is of minimization/maximization
- Define decision variables
  - Specify lower and upper bounds
- Add objective function
- Add constraints
- Write the problem data into '.lp' file
- Solve the problem
- Write results

Qd = 56.48 gpm and Qo = 177.52 gpm

Irrigation and TDS requirements are binding constraints

```
"""
Blending Ogallala and Dockum Waters for Corn Production
Authors: Venki Uddameri
"""
# Import PuLP modeler functions
from pulp import *

# Create the 'prob' variable to contain the problem data
prob = LpProblem("Ogllala Blend",LpMinimize)

# The 2 variables Qd (dockum) and Qo (ogallala) are created
# Lower bound = 0 and Upper bound = well capacities
Qd=LpVariable("DockumPumping",0,75)
Qo=LpVariable("OggPumping",0,200)

# The objective function is added to 'prob' first
prob += Qo, "Minimize Ogallala Pumping"

# The five constraints are entered
prob += Qd + Qo >= 234, "Irrigation Requirement"
prob += 1100*Qd - 350*Qo <= 0.0, "TDSRequirement"
prob += 2*Qd - 5*Qo <= 0.0, "SARRequirement"

# The problem data is written to an .lp file
prob.writeLP("oggdockum.lp")

# The problem is solved using PuLP's choice of Solver
prob.solve()

# The status of the solution is printed to the screen
print ("Status:", LpStatus[prob.status])

# Each of the variables is printed with it's optimum value
for v in prob.variables():
    print(v.name, "=", v.varValue)
```

Other Optimization Routines

- Python's scipy optimization module has several functions
- Python has other libraries for linear and nonlinear optimization

» PythonForOperationsResearch

» PythonForOp...onsResearch

This page attempts to collect information and links pertaining to the field of Operations Research, which includes problems in Linear Programming, Integer Programming, Stochastic Programming, and other Optimization methods in python.

- **APM Python** - APM Python is free optimization software through a web service. Nonlinear Programming problem are sent to the APMonitor server and results are returned to the local Python script. A web-interface automatically loads to help visualize solutions, in particular dynamic optimization problems that include differential and algebraic equations. Default solvers include APOPT, BPOPT, and IPOPT. Pre-configured modes include optimization, parameter estimation, dynamic simulation, and nonlinear control.
- **Coopr** - The Coopr software project integrates a variety of Python optimization-related packages.
- **CVOXPT** - CVXOPT is a free software package for convex optimization based on the Python programming language. It can be used with the interactive Python interpreter, on the command line by executing Python scripts, or integrated in other software via Python extension modules. Its main purpose is to make the development of software for convex optimization applications straightforward by building on Python's extensive standard library and on the strengths of Python as a high-level programming language.
- OpenOpt (license: BSD) contains connections to tens of solvers and has some own Python-written ones, e.g. nonlinear solver with specifiable accuracy: interalg, graphic output of convergence and some more numerical optimization "MUST HAVE" features. Also OpenOpt can solve FuncDesigner problems with automatic differentiation, that usually work faster and gives more precise results than finite-differences derivatives approximation.
- **prodyn** - a generic implementation of the dynamic programming algorithm for optimal system control.
- **PuLP** - PuLP is an LP modeler written in python. PuLP can generate MPS or LP files and call GLPK, COIN CLP/CBC, CPLEX, and GUROBI to solve linear problems.
- **Pyomo** - The Python Optimization Modeling Objects (Pyomo) package is an open source tool for modeling optimization applications in Python. Pyomo can be used to define symbolic problems, create concrete problem instances, and solve these instances with standard solvers. Pyomo provides a capability that is commonly associated with algebraic modeling languages such as AMPL, AIMMS, and GAMS, but Pyomo's modeling objects are embedded within a full-featured high-level programming language with a rich set of supporting libraries. Pyomo leverages the capabilities of the Coopr software library, which integrates Python packages for defining optimizers, modeling optimization applications, and managing computational experiments.
- **pyOpt** - pyOpt is a package for formulating and solving nonlinear constrained optimization problems in an efficient, reusable and portable manner (license: LGPL).
- **PySCIPOpt** - PySCIPOpt provides an interface from Python to the SCIP Optimization Suite.
- **scipy.optimize** - some solvers written or connected by SciPy developers.'
- **ticdat** - ticdat simplifies the process of developing modular mathematical engines to read from one schema and write to another. Specifically designed with Mixed Integer Programming problems in mind, it can be used for rapidly developing a wide variety of mathematical engines.

https://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html

SciPy.org

SciPy.org | Docs | SciPy v1.4.1 Reference Guide | SciPy Tutorial

Optimization (scipy.optimize)

The scipy.optimize package provides several commonly used optimization algorithms. A detailed listing is available: scipy.optimize (can also be found by `help(scipy.optimize)`).

The module contains:

1. Unconstrained and constrained minimization of multivariate scalar functions (**minimize**) using a variety of algorithms (e.g., BFGS, Nelder-Mead simplex, Newton Conjugate Gradient, COBYLA or SLSQP).
2. Global optimization routines (e.g., **basinhopping**, **differential_evolution**, **shgo**, **dual_annealing**).
3. Least-squares minimization (**least_squares**) and curve fitting (**curve_fit**) algorithms.
4. Scalar univariate functions minimizers (**minimize_scalar**) and root finders (**root_scalar**).
5. Multivariate equation system solvers (**root**) using a variety of algorithms (e.g., hybrid Powell, Levenberg-Marquardt or large-scale methods such as Newton-Krylov [KK]).

Below, several examples demonstrate their basic usage.

Unconstrained minimization of multivariate scalar functions (minimize)

The **minimize** function provides a common interface to unconstrained and constrained minimization algorithms for multivariate scalar functions in **scipy.optimize**. To demonstrate the minimization function, consider the problem of minimizing the Rosenbrock function of N variables:

$$f(\mathbf{x}) = \sum_{i=2}^{N} 100\left(x_{i+1} - x_i^2\right)^2 + (1 - x_i)^2.$$

The minimum value of this function is 0 which is achieved when $x_i = 1$.

Note that the Rosenbrock function and its derivatives are included in **scipy.optimize**. The implementations shown in the following sections provide examples of how to define an objective function as well as its jacobian and hessian functions.

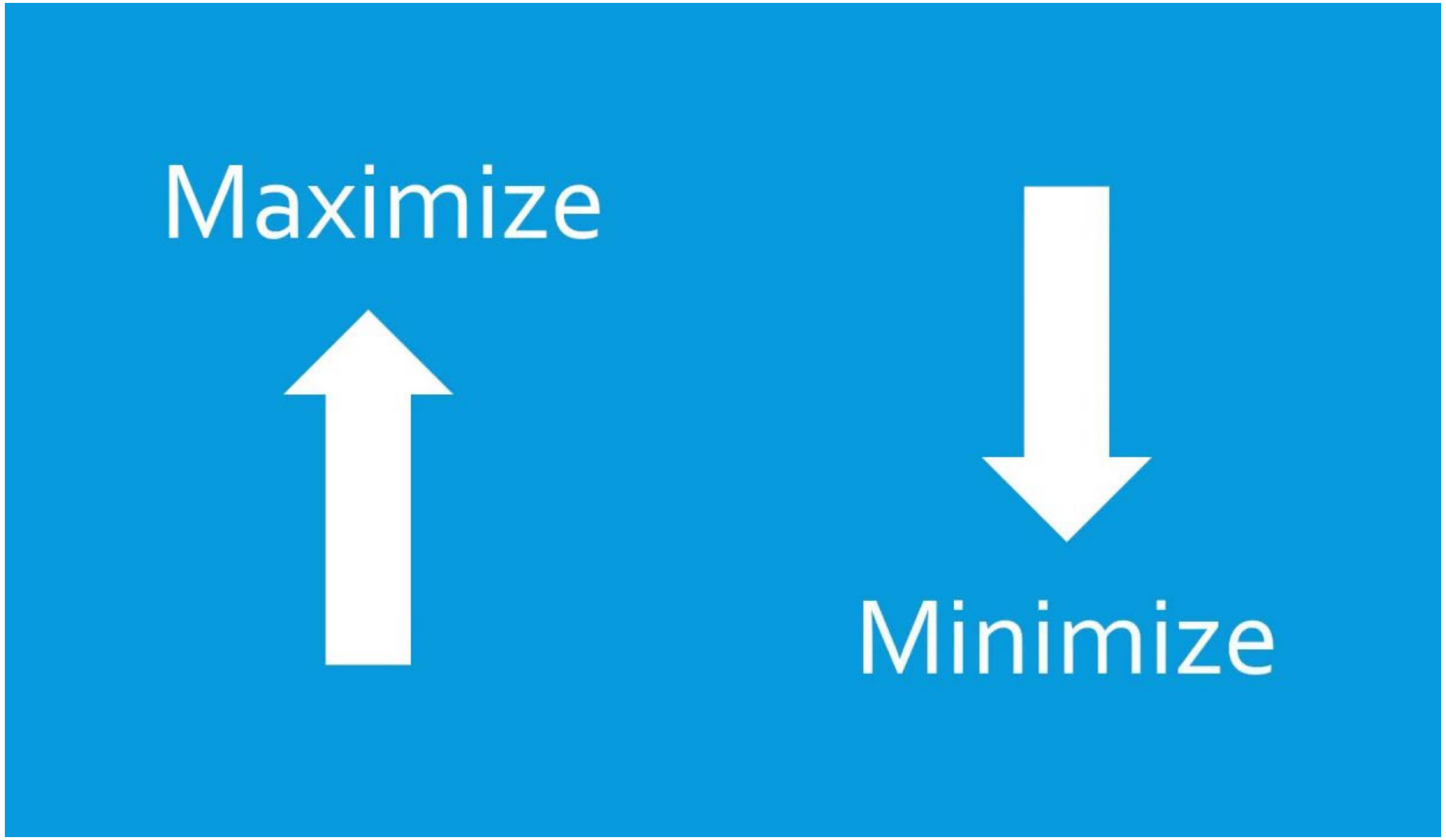
Nelder-Mead Simplex algorithm (`method='Nelder-Mead'`)

In the example below, the **minimize** routine is used with the *Nelder-Mead* simplex algorithm (selected through the `method` parameter):

```
>>> import numpy as np
>>> from scipy.optimize import minimize
```

```
>>> def rosen(x):
...     """The Rosenbrock function"""
...     return sum(100.0*(x[1:]-x[:-1]**2.0)**2.0 + (1-x[:-1])**2.0)
```

https://wiki.python.org/moin/PythonForOperationsResearch



You Should Know

- How to perform unconstrained minimization using python scipy library
- Use of unconstrained minimization for linear regression problems
- Solving linear programming models using python
    - Installing puLP modeling library in anaconda
    - Using puLP for solving linear programming models