

Learning By Examples with
ANACONDA3/2019 – 7
BOOKLET



PHOTO:WWW.UNSPLASH.COM

Ahmad A.M. Allawi

Learning By Examples with
ANACONDA 3 / 2019 –
JULY
BOOKLET

Ahmad A.M. Allawi

Learning By Examples with
ANACONDA 3 / 2019 –
JULY
BOOKLET

**Registration Number (834) - 2019 at the General
Directorate of Libraries in Erbil**

– Kurdistan Region – Iraq -

Copyright HAVAL ART -- Printing Press Company -

Iraq - Kurdistan Region - Erbil

www.havalpress.com

Acknowledgment

I am grateful to my sister Nowf - Senior Architect, for her kind help in the general setting of this booklet.

Ahmad A. M. Allawi

September / 2019

Anaconda (Python distribution)

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system Conda. Anaconda distribution is used by over 15 million users and includes more than 1500 popular data-science packages suitable for Windows, Linux, and MacOS. Anaconda can be downloaded from www.anaconda.org.

The core packages of Anaconda are:

Anaconda Platform Document: 5.2.0-July/2019 (211 pages)

SciPy: Reference Guide Release 1.3.0-May/2019 (2535 pages).

Matplotlib: Release 3.1.1-July/2019 (2354 pages).

Pandas: Powerful Python data analysis toolkit Release 0.25.0-July/2019 (2827 pages).

NumPy: Reference Release 1.16.1-January/2019 (1372 pages).

Anaconda Navigator:

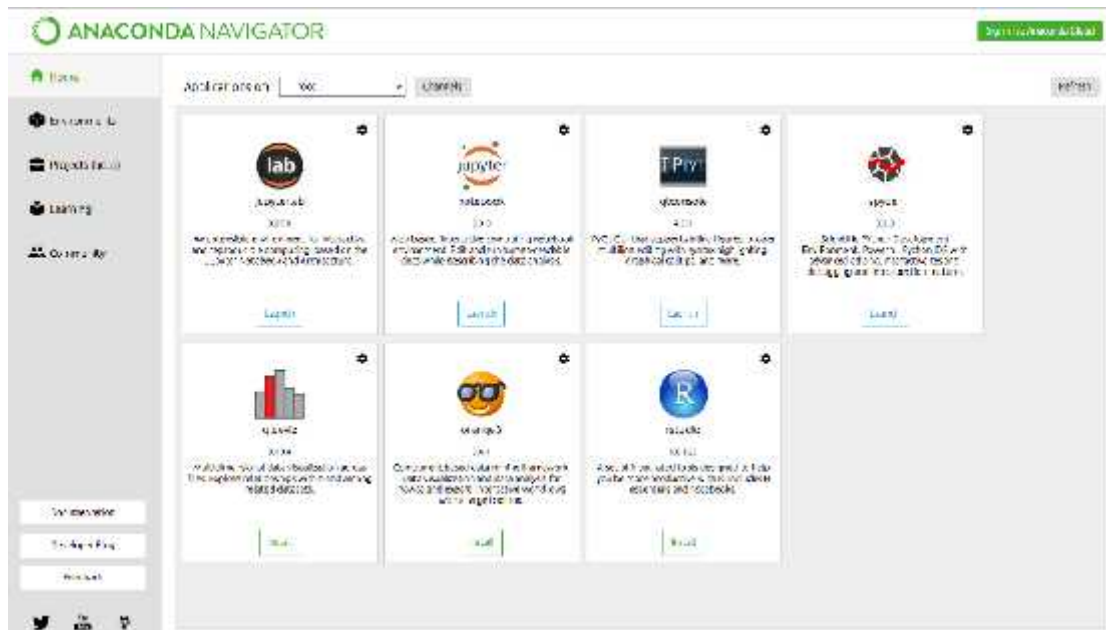
Is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.

The following applications are available by default in the Navigator:

Rstudio, Jupyter Notebook, JupyterLab, Spyder, Glue, and Orange.

The default environment is Python 3.6, but you can easily install Python 3.5, Python 2.7 or R. The above documentation is incredibly detailed and there is an excellent community of users for additional support.

Anaconda Navigator on Startup



Python is the programming language which will be installed on the machine and on top of that different IDEs and packages can be installed. Python on it's own is not going to be very useful unless an IDE is installed. This is where Anaconda comes in the picture. Anaconda installs IDEs and several important packages like NumPy, Pandas, and so on, and this is a very convenient package which can be downloaded and installed.

Introduction to This Booklet

Python applications are very huge and it is very widely used in Engineering and Non-Engineering fields.

The Python programming language is widely used by companies around the world to build web, analyze data, automate operations via DevOps and create reliable scalable enterprise applications.

Many companies do not even realize they are using Python across their organizations. For example, if a company is a "Java-only shop" but they use IBM WebSphere as a web application server then they have to use Python to script the server's configuration!. Python has a habit of getting in everywhere regardless of whether the usage is intentional.

The 8 World-Class Software Companies That Use Python:

- Industrial Light and Magic
- Google
- Facebook
- Instagram
- Spotify
- Quora
- Netflix
- Dropbox
- Reddit

This booklet is not a textbook or a solution manual covering a certain subject with hundreds of pages. It is a booklet with few selected topics and application examples, the goal of which is to make a short cut to the subjects covered. The readers can modify the examples to suit their own application without digging much in Python, if they want.

However, the readers can easily refer to the very huge number of text books, tutorials and research papers etc available elsewhere.

This booklet is not intended for the beginners, but to be a guide line for the methodology of the applications of the programs listed in the articles. I assume that the readers should have some basic knowledge in using Anaconda/Jupyter and especially in Python programming language.

The articles listed below that I chose in this booklet, though few, were selected among the unlimited number of subjects covered in the various textbooks, tutorials and research papers and these are:

- **Pint for unit conversion.**
- **SchemDraw.**
- **Linear regression and machine learning.**
- **Symbolic Math (SymPy) and ODE.**
- **Linear Optimization using Pulp.**
- **Linear and Non-Linear Optimization using SciPy.**

Throughout this booklet, we will use Jupyter Notebook for programming.

Ahmad A.M. Allawi

Consultant Electrical Engineer-Lecturer

Parametric Technology Corporation (PTC) Community Member

No.4009681 – USA

Articles

1- Pint for unit conversion.....	Page-8
Units Conversion Program	
2- SchemDraw.....	Page-9
Schem Draw Program	
3- Linear regression and machine learning.....	Page-10
Multiple Linear Regression Program	
(Tkinter Window for MLR – Figure)	
Machine Learning Program	
Random Forest Program	
4- Symbolic Math (SymPy) and ODE.....	Page-11
Algebra and Symbolic Math with SymP Program	
5- Linear optimization using Pulp.....	Page-12
Simple Example Program	
Company X Problem Program	
Power Company	
Power Company Program	
Optimization of Boiler-Turbo Generators.....	Page-15
Boiler Turbo Generators Program	
6- Linear / Non-Linear optimization using SciPy.....	Page-20
Scipy (Scientific Python)	
Linear / Non-Linear Example with SciPy Program	
Appendix.....	Page-23
References.....	Page-24

1 - Pint for Unit Conversion

Units Conversion

Pint For Units Conversion

Pint library is used for unit conversion.

Pint has to be downloaded using Anaconda Prompt:

With the internet connected, type after the prompt sign `> pip install pint`

This will install Pint Version 0.9 or later.

Working with Units in Python using the pint library

There are a number of python libraries that incorporate units to be used with python calculations.

Among these, pint is a relatively new library that builds on experience with earlier attempts.

The core concept in pint is to work with a unit registry (ur), which is created as shown below:

In [1]:

```
# Example-1: Chemical Engineering

# The unit registry provides a simple means to assign units
# using the multiplication operator.

# For example, here's how to compute the molarity of a sodium chloride solution
# in 58.44 grams of NaCl (mw = 58.44)

# has been dissolved in water to form 3 liters of solution.

from pint import UnitRegistry

ur = UnitRegistry()

# problem data
V = 3.0 * ur.liters
m = 58.44 * ur.grams
mw = 58.44 * ur.grams/ur.mol

# compute molarity
C = m/(mw*V)

print(C)
```

0.3333333333333333 mole / liter

In [3]:

```
# Example-2

# In Unit Conversion Each variable with units has to() and ito() methods
# for converting the quantity to a
# desired set of units.

# The to() method is used to create a new variable by converting
# an existing variable to the indicated units.

x = 0.5 * ur.kilograms/ur.gallon
y = x.to(ur.grams/ur.liter)

print(x)

print(y)
```

```
0.5 kilogram / gallon
132.08602617907425 gram / liter
```

In [4]:

```
# The ito() method converts an existing variable 'in-place'.

x = 0.5 * ur.kilograms/ur.gallon
x.ito(ur.grams/ur.liter)

print(x)
```

```
132.08602617907425 gram / liter
```

In [5]:

```
# Example-3

# Here's how to compute the molarity of a sodium chloride solution in which
# a 0.5 pounds of NaCl (mw = 58.44)
# has been dissolved in water to form 2 gallons of solution.

# problem data

V = 3.0 * ur.gallons
m = 0.5 * ur.lbs
mw = 58.44 * ur.grams/ur.mol

# compute concentration

C = m/(mw*V)
print(C)

# convert to desired units and print

C = C.to(ur.mol/ur.liter)
print(C)

# convert to moles per gallon

C.ito(ur.mol/ur.gallon)

print(C)
```

```
0.0028519279032626055 mole * pound / gallon / gram
0.3417363316133261 mole / liter
1.2936127367100163 mole / gallon
```

In [8]:

```
# Example-4 (Simple One)

distance = 24.0 * ur.meter

time = 8.0 * ur.second

# Unit Check

speed = distance / time

print(speed)
```

```
24.0 meter
8.0 second
<Quantity(8.0, 'second')>
```

In [11]:

```
# Example-5

wavelength = 1550 * ur.nm
frequency = (ur.speed_of_light / wavelength).to('Hz')
print(frequency)
print(frequency.to_compact())
```

```
193414489032258.03 hertz
193.41448903225802 terahertz
```

In [16]:

```
# Example-6 ( Converting Deg.C to Deg.F)

home = Q_(25.4, ur.degC)

print(home.to('degF'))
```

```
77.72000039999993 degF
```

Units and Conversions for Home Heating

This will demonstrates unit conversions for several typical calculations of energy consumption.

Heating with Electricity:

A typical energy efficient home requires 50 million BTUs to heat the home for the winter. If the price of electricity is \$0.08 per kilowatt hour, what would be the cost to heat a typical home with electricity?

Solution

In [17]:

```
Q_btu = 50e6          # BTU

price = 0.08          # USD per kwh
btu_per_joule = 9.486e-4 # conversion factor
kwh_per_joule = 2.778e-7 # conversion factor

cost = price*kwh_per_joule*Q_btu/btu_per_joule

print("Winter heating cost =", round(cost,2), "USD")
```

```
Winter heating cost = 1171.41 USD
```

Heating with Natural Gas

Natural gas is transported by pipeline from producing areas of the country to the midwest where it stored prior to distribution. For heating purposes, the energy content of natural gas is 1000 BTU per cubic foot at 1 atm and 15 °C.

If the natural gas is stored at 1000 psia and 15 °C, how large a storage tank is required to store natural gas for the winter?

Give your answer in cubic meters.

Solution

The volume of natural gas required is determined by the heating requirement.

In [18]:

```
V_ft3 = Q_btu/1000    # ft^3

# Next compute the amount of natural gas required in lb moles

R = 10.73            # ft^3 psia/(Lbmol R)
T_degC = 15         # deg C

# convert temperature to absolute

T_degR = 9.0*T_degC/5.0 + 491.67    # deg R

# compute lb moles

n_lbmol = 14.696*V_ft3/(R*T_degR)

print(round(T_degR, 1), "degrees Rankine")

print(round(n_lbmol, 1), "lb-mols")
```

```
518.7 degrees Rankine
132.0 lb-mols
```

In [19]:

```
# Now calculate the volume in cubic meters at the storage conditions.

V_ft3 = n_lbmol*R*T_degR/1000.0

m3_per_ft3 = 0.028317

V_m3 = V_ft3*m3_per_ft3

print("Storage volume of natural gas at 1000 psia and 15 deg C =", round(V_m3,1), "cubic me
```

```
Storage volume of natural gas at 1000 psia and 15 deg C = 20.8 cubic meters
```

Sizing a Propane Storage Tank

Liquid propane has a heating value of 46.3 MJ/kg, and a specific gravity of 0.493 at ambient temperatures.

How large a storage tank will be required, in cubic meters.

Solution

In [20]:

```
btu_per_joule = 9.486e-4

Q_joule = Q_btu/btu_per_joule
print("Heat requirement =", round(Q_joule/1e6,1), "Megajoules")

m_kg = Q_joule/46.3e6
print("Mass of propane required = {:.2f} kg".format(m_kg))

V_m3 = m_kg/0.493/1000.0

print("Volume of propane required = {:.2f} cubic meters".format(V_m3))
```

```
Heat requirement = 52709.3 Megajoules
Mass of propane required = 1138.43 kg
Volume of propane required = 2.31 cubic meters
```

In []:

2 – SchemDraw

Schem Draw

Drawing quality electrical schematics is one of those tasks that always takes too long. Most software for the job focuses on fancy circuit simulations and doesn't care about appearance. Constantly facing the dilemma of how to draw simple schematic diagrams.

SchemDraw is electrical engineering package and is not part of Anaconda library and should be downloaded and installed using Anaconda prompt:

With the internet connected, type `pip install schemdraw`.

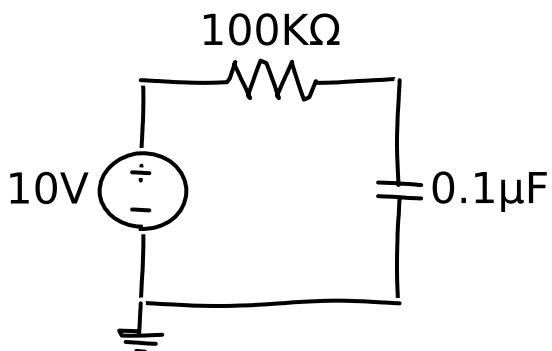
The latest version I have downloaded is 0.4.0.

SchemDraw covers electrical components such as, Resistors, Capacitors, Inductors, Transformers, Diodes, Transistors etc.

Here I will demonstrate how to use schemdraw through some examples.

In [24]:

```
# Example-1
# Resistor and Capacitor connected in series to a DC power supply
# Importing required Libraries
import SchemDraw as schem
import SchemDraw.elements as e # e, stands for elements or components
d = schem.Drawing()
# Drawing the Circuit
# We need a voltage source defined by Label and a SOURCE
V1 = d.add(e.SOURCE_V, label='10V')
# Adding a horizontal resistor of value 100 KOhm
d.add(e.RES, d='right', label='100K$\Omega$')
# Add vertical capacitor
d.add(e.CAP, d='down', botlabel='0.1$\mu$F')
# Connect a Line to the Source voltage
d.add(e.LINE, to=V1.start)
# Add a Ground
d.add(e.GND)
# Draw the circuit
d.draw(showplot=False)
# Save the circuit
d.save('testschematic.eps')
```



In [2]:

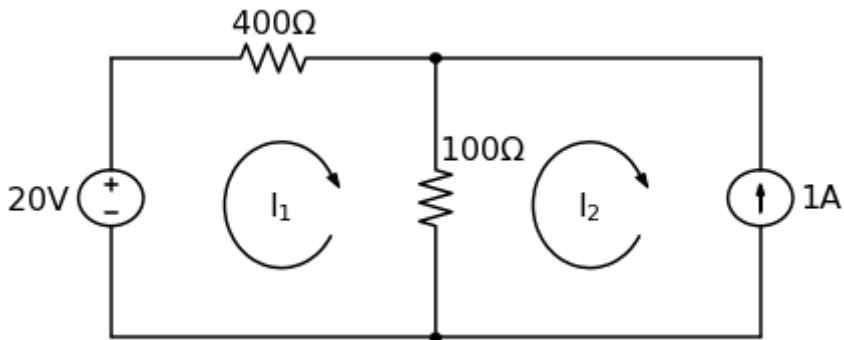
```

# Example-2

import SchemDraw as schem
import SchemDraw.elements as e

d = schem.Drawing(unit=5) # Unit will affect circuit size
V1 = d.add(e.SOURCE_V, label='$20V$')
R1 = d.add(e.RES, d='right', label='400$\Omega$')
d.add(e.DOT)
d.push()
R2 = d.add(e.RES, d='down')
R2.add_label('100$\Omega$', loc='center', ofst=[-.9,.05])
d.add(e.DOT)
d.pop()
L1 = d.add(e.LINE)
I1 = d.add(e.SOURCE_I, d='down', botlabel='1A')
L2 = d.add(e.LINE, d='left', tox=V1.start)
d.loopI([R1,R2,L2,V1], '$I_1$', pad=1.25)
# Use R1 as top element for both so they get the same height
d.loopI([R1,I1,L2,R2], '$I_2$', pad=1.25)
d.draw(showplot=False)
d.save('loop_current.svg')

```

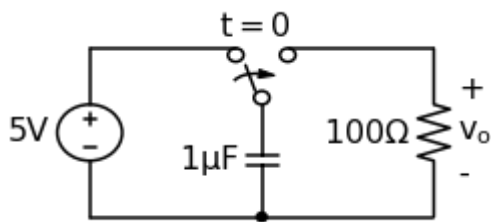


In [3]:

```
# Example-3
```

```
import SchemDraw as schem
import SchemDraw.elements as e

d = schem.Drawing()
V1 = d.add(e.SOURCE_V, label='5V')
d.add(e.LINE, d='right', l=d.unit*.75)
S1 = d.add(e.SWITCH_SPDT2_CLOSE, d='up', anchor='b', rgtlabel='$t=0$')
d.add(e.LINE, d='right', xy=S1.c, l=d.unit*.75)
d.add(e.RES, d='down', label='$100\Omega$', botlabel=['+', '$v_o$', '-'])
d.add(e.LINE, to=V1.start)
d.add(e.CAP, xy=S1.a, d='down', toy=V1.start, label='1$\mu$F')
d.add(e.DOT)
d.draw(showplot=False)
d.save('cap_charge.svg')
```



In [4]:

```
# Example-4 : A Power Supply
```

```
import SchemDraw as schem
import SchemDraw.elements as e

d = schem.Drawing(inches_per_unit=.5, unit=3)
D1 = d.add(e.DIODE, theta=-45)
d.add(e.DOT)
D2 = d.add(e.DIODE, theta=225, reverse=True)
d.add(e.DOT)
D3 = d.add(e.DIODE, theta=135, reverse=True)
d.add(e.DOT)
D4 = d.add(e.DIODE, theta=45)
d.add(e.DOT)

d.add(e.LINE, xy=D3.end, d='left', l=d.unit*1.5)
d.add(e.DOT_OPEN)
d.add(e.GAP, d='up', toy=D1.start, label='AC IN')
d.add(e.LINE, xy=D4.end, d='left', l=d.unit*1.5)
d.add(e.DOT_OPEN)

top = d.add(e.LINE, xy=D2.end, d='right', l=d.unit*3)
Q2 = d.add(e.BJT_NPN_C, anchor='collector', d='up', label='Q2\n2n3055')
d.add(e.LINE, xy=Q2.base, d='down', l=d.unit/2)
Q2b = d.add(e.DOT)
d.add(e.LINE, d='left', l=d.unit/3)
Q1 = d.add(e.BJT_NPN_C, anchor='emitter', d='up', label='Q1\n2n3054')
d.add(e.LINE, d='up', xy=Q1.collector, toy=top.center)
d.add(e.DOT)

d.add(e.LINE, d='down', xy=Q1.base, l=d.unit/2)
d.add(e.DOT)
d.add(e.ZENER, d='down', reverse=True, botlabel='D2\n500mA')
d.add(e.DOT)
G = d.add(e.GND)
d.add(e.LINE, d='left')
d.add(e.DOT)
d.add(e.CAP_P, botlabel='C2\n100$\mu$F\n50V', d='up', reverse=True)
d.add(e.DOT)
d.push()
d.add(e.LINE, d='right')
d.pop()
d.add(e.RES, d='up', toy=top.end, botlabel='R1\n2.2K\n50V')
d.add(e.DOT)

d.here = [d.here[0]-d.unit, d.here[1]]
d.add(e.DOT)
d.add(e.CAP_P, d='down', toy=G.start, label='C1\n100$\mu$F\n50V', flip=True)
d.add(e.DOT)
d.add(e.LINE, xy=G.start, tox=D4.start, d='left')
d.add(e.LINE, d='up', toy=D4.start)

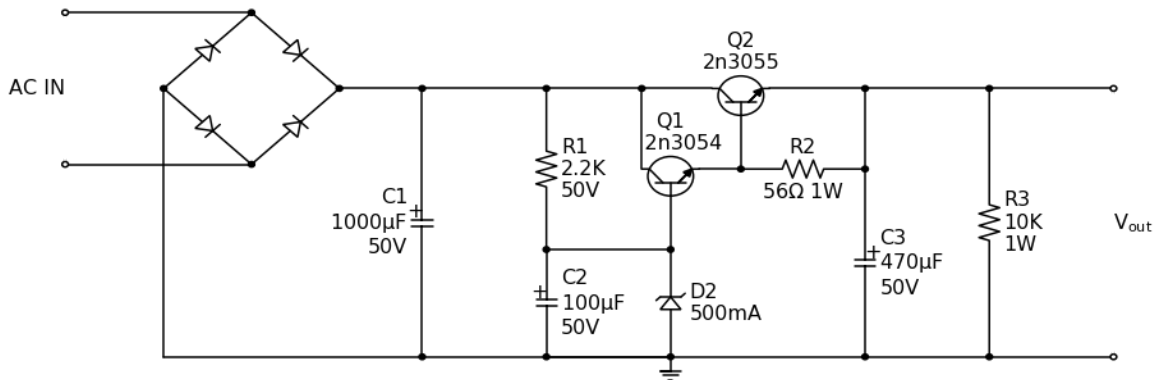
d.add(e.RES, d='right', xy=Q2b.center, label='R2', botlabel='56$\Omega$ 1W')
d.add(e.DOT)
d.push()
d.add(e.LINE, d='up', toy=top.start)
d.add(e.DOT)
d.add(e.LINE, d='left', tox=Q2.emitter)
d.pop()
```

```

d.add(e.CAP_P, d='down', toy=G.start, botlabel='C3\n470$\mu$F\n50V')
d.add(e.DOT)
d.add(e.LINE, d='left', tox=G.start, move_cur=False)
d.add(e.LINE, d='right')
d.add(e.DOT)
d.add(e.RES, d='up', toy=top.center, botlabel='R3\n10K\n1W')
d.add(e.DOT)
d.add(e.LINE, d='left', move_cur=False)
d.add(e.LINE, d='right')
d.add(e.DOT_OPEN)
d.add(e.GAP, d='down', toy=G.start, label='$V_{out}$')
d.add(e.DOT_OPEN)
d.add(e.LINE, d='left')

d.draw(showplot=False)
d.save('powersupply.svg')

```



In [5]:

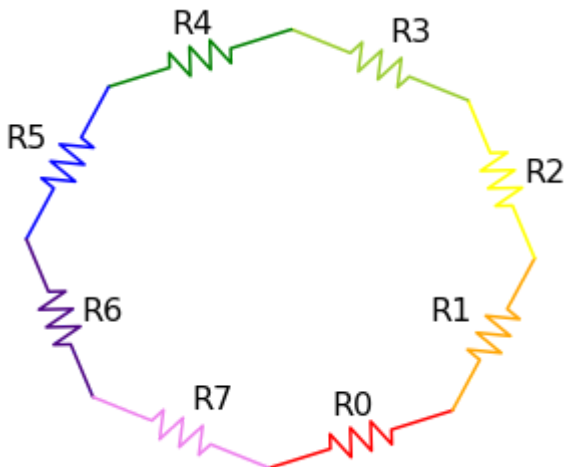
Example-5

```

import SchemDraw as schem
import SchemDraw.elements as e

colors = ['red', 'orange', 'yellow', 'yellowgreen', 'green', 'blue', 'indigo', 'violet']
d = schem.Drawing()
for i in range(8):
    d.add(e.RES, label='R%d'%i, theta=45*i+20, color=colors[i])
d.draw(showplot=False)
d.save('Rcircle.svg')

```



In [6]:

```

# Example-6: An OP Amp

import matplotlib.pyplot as plt

import SchemDraw as schem
import SchemDraw.elements as e

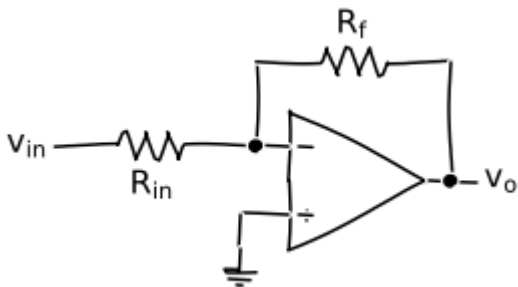
plt.xkcd()

d = schem.Drawing(inches_per_unit=.5)
op = d.add(e.OPAMP)
d.add(e.LINE, d='left', xy=op.in2, l=d.unit/4)
d.add(e.LINE, d='down', l=d.unit/5)
d.add(e.GND)
d.add(e.LINE, d='left', xy=op.in1, l=d.unit/6)
d.add(e.DOT)
d.push()
Rin = d.add(e.RES, d='left', xy=op.in1-[d.unit/5,0], botlabel='$R_{in}$', lftlabel='$v_{in}$')
d.pop()
d.add(e.LINE, d='up', l=d.unit/2)
Rf = d.add(e.RES, d='right', l=d.unit*1, label='$R_f$')
d.add(e.LINE, d='down', toy=op.out)
d.add(e.DOT)
d.add(e.LINE, d='left', tox=op.out)
d.add(e.LINE, d='right', l=d.unit/4, rgtlabel='$v_{o}$')

d.draw()

d.save('ex_xkcd.svg')

```



Solving Circuit Diagram Problem with Python and SchemDraw

In [7]:

```

import matplotlib.pyplot as plt
%matplotlib inline
# if high-resolution images are desired:
# include %config InlineBackend.figure_format = 'svg'
%config InlineBackend.figure_format = 'svg'
import SchemDraw as schem
import SchemDraw.elements as e

```


In [8]:

```
# Here we will Draw the circui diagram
```

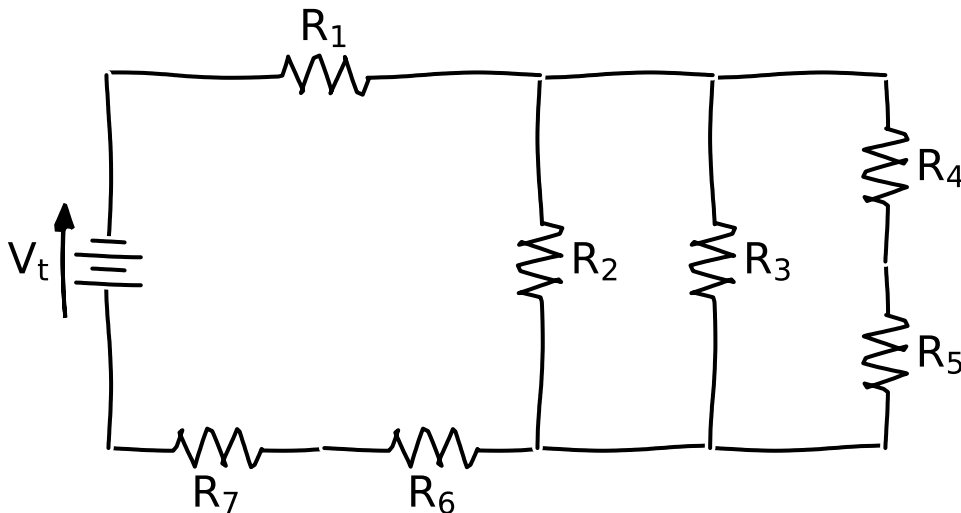
```
d = schem.Drawing(unit=2.5)
R7 = d.add(e.RES, d='right', botlabel='$R_7$')
R6 = d.add(e.RES, d='right', botlabel='$R_6$')
d.add(e.LINE, d='right', l=2)
d.add(e.LINE, d='right', l=2)
R5 = d.add(e.RES, d='up' , botlabel='$R_5$')
R4 = d.add(e.RES, d='up', botlabel='$R_4$')
d.add(e.LINE, d='left', l=2)
d.push()
R3 = d.add(e.RES, d='down', toy=R6.end, botlabel='$R_3$')
d.pop()
d.add(e.LINE, d='left', l=2)
d.push()
R2 = d.add(e.RES, d='down', toy=R6.end, botlabel='$R_2$')
d.pop()
R1 = d.add(e.RES, d='left', tox=R7.start, label='$R_1$')
Vt = d.add(e.BATTERY, d='up', xy=R7.start, toy=R1.end, label='$V_t$', lbfst=0.3)
d.labelI(Vt, arrowlen=1.5, arrowfst=0.5)
d.draw()
d.save('7_resistors_3_loops.png')
#d.save('7_resistors_3_loops.pdf')
```

```
# Find:
```

```
# V6 and V7, the voltage drop across resistors R6 and R7
```

```
# I3 and I6, the current running through resistors R3 and R6
```

```
# P4 and P7, the power dissipated by resistors R4 and R7
```



In [9]:

```
# Circuit Values
```

```
Vt = 5.2
```

```
R1 = 0.0132
```

```
R2 = 0.021
```

```
R3 = 0.00360
```

```
R4 = 0.0152
```

```
R5 = 0.0119
```

```
R6 = 0.0022
```

```
R7 = 0.00740
```

In [10]:

```
# Simplify the circuit
```

```
R45 = R4 + R5      # Series values of R4 and R5
```

```
R67 = R6 + R7      # Series values of R6 and R7
```

```
# Print Results
```

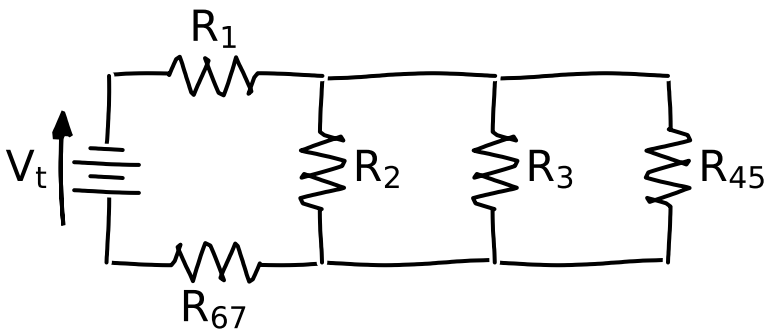
```
print(f'R45 = {round(R45,7)} Ohm, R67 = {round(R67,5)} Ohm')
```

```
R45 = 0.0271 Ohm, R67 = 0.0096 Ohm
```

In [11]:

```
# Redraw the circuit showing the combined resistors

d = schem.Drawing(unit=2.5)
R67 = d.add(e.RES, d='right', botlabel='$R_{67}$')
d.add(e.LINE, d='right', l=2)
d.add(e.LINE, d='right', l=2)
R45 = d.add(e.RES, d='up', botlabel='$R_{45}$')
d.add(e.LINE, d='left', l=2)
d.push()
R3 = d.add(e.RES, d='down', toy=R67.end, botlabel='$R_3$')
d.pop()
d.add(e.LINE, d='left', l=2)
d.push()
R2 = d.add(e.RES, d='down', toy=R67.end, botlabel='$R_2$')
d.pop()
R1 = d.add(e.RES, d='left', tox=R67.start, label='$R_1$')
Vt = d.add(e.BATTERY, d='up', xy=R67.start, toy=R1.end, label='$V_t$', lblofst=0.3)
d.labelI(Vt, arrowlen=1.5, arrowofst=0.5)
d.draw()
d.save('5_resistors_3_loops.png')
#d.save('5_resistors_3_loops.pdf')
```



In [12]:

```
# Finding parallel combination of resistors R2, R3 and R45

Vt = 5.2

R1 = 0.0132
R2 = 0.021
R3 = 0.00360
R4 = 0.0152
R5 = 0.0119
R6 = 0.0022
R7 = 0.00740

R45 = R4 + R5
R67 = R6 + R7

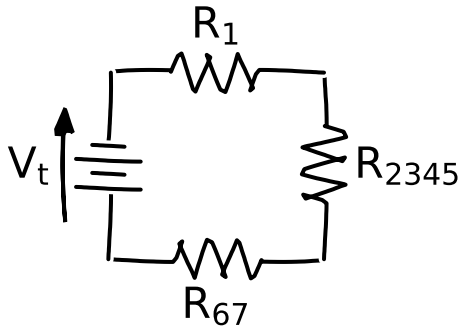
R2345 = ((1/R2)+(1/R3)+(1/R45))*(-1)
print(f'R2345 = {round(R2345,7)} Ohm')
```

R2345 = 0.0027602 Ohm

In [13]:

```
# Further simplifying the circuit by combining resistors

d = schem.Drawing(unit=2.5)
R67 = d.add(e.RES, d='right', botlabel='$R_{67}$')
R345 = d.add(e.RES, d='up', botlabel='$R_{2345}$')
R1 = d.add(e.RES, d='left', tox=R67.start, label='$R_1$')
Vt = d.add(e.BATTERY, d='up', xy=R67.start, toy=R1.end, label='$V_t$', lblofst=0.3)
d.labelI(Vt, arrowlen=1.5, arrowofst=0.5)
d.draw()
d.save('3_resistors_1_loop.png')
#d.save('3_resistors_1_loop.pdf')
```



In [14]:

```
# Finding the total resistance value

Vt = 5.2

R1 = 0.0132
R2 = 0.021
R3 = 0.00360
R4 = 0.0152
R5 = 0.0119
R6 = 0.0022
R7 = 0.00740

R45 = R4 + R5
R67 = R6 + R7

R2345 = ((1/R2)+(1/R3)+(1/R45))**(-1)

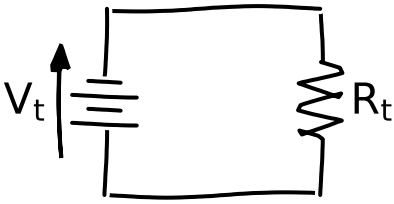
Rt = R1 + R2345 + R67
print(f'Rt = {round(Rt,7)} Ohm')
```

Rt = 0.0255602 Ohm

In [15]:

```
# Last circuit diagram including Vt and Rt

d = schem.Drawing(unit=2.5)
L2 = d.add(e.LINE, d='right')
Rt = d.add(e.RES, d='up', botlabel='$R_{t}$')
L1 = d.add(e.LINE, d='left', tox=L2.start)
Vt = d.add(e.BATTERY, d='up', xy=L2.start, toy=L1.end, label='$V_{t}$', lbfst=0.3)
d.labelI(Vt, arrowlen=1.5, arrowfst=0.5)
d.draw()
d.save('1_resistor_no_loops.png')
#d.save('1_resistor_no_loops.pdf')
```



In [16]:

```
# Finding total current It

Vt = 5.2

R1 = 0.0132
R2 = 0.021
R3 = 0.00360
R4 = 0.0152
R5 = 0.0119
R6 = 0.0022
R7 = 0.00740

R45 = R4 + R5
R67 = R6 + R7

R2345 = ((1/R2)+(1/R3)+(1/R45))**(-1)
Rt = R1 + R2345 + R67

It = Vt/Rt
print(f'It = {round(It,2)} A')
```

It = 203.44 A

In [17]:

```
# Voltage drop V6 and V7

I6 = It
I7 = It
V6 = I6 * R6
V7 = I7 * R7
print(f'V6 = {round(V6,5)} V, V7 = {round(V7,5)} V')
```

V6 = 0.44757 V, V7 = 1.50547 V

In [18]:

```
I2345 = It
V2345 = I2345 * R2345
print(f'V2345 = {round(V2345,5)} V')
```

V2345 = 0.56153 V

In [19]:

```
# I3 and I6
V3 = V2345
I3 = V3 / R3
I6 = It
print(f'I3 = {round(I3,2)} A, I6 = {round(I6,2)} A')
```

I3 = 155.98 A, I6 = 203.44 A

In [20]:

```
# Power in resistor R7
I7 = It
P7 = R7 * I7**2
print(f'P7 = {round(P7,2)} W')
```

P7 = 306.27 W

In [21]:

```
# Current in R45
V45 = V2345
I45 = V45/R45
print(f'I45 = {round(I45,3)} A')
```

I45 = 20.721 A

In [22]:

```
# Power in R4
I4 = I45
P4 = R4 * I4**2
print(f'P4 = {round(P4,4)} W')
```

P4 = 6.5261 W

In [23]:

```
# Printing the results

print(f'V6 = {round(V6,3)} V')
print(f'V7 = {round(V7,2)} V')
print(f'I3 = {round(I3,0)} A')
print(f'I6 = {round(I6,0)} A')
print(f'P4 = {round(P4,2)} W')
print(f'P7 = {round(P7,0)} W')
```

```
V6 = 0.448 V
V7 = 1.51 V
I3 = 156.0 A
I6 = 203.0 A
P4 = 6.53 W
P7 = 306.0 W
```

Conclusion

SchemDraw is a great package for making circuit diagrams in Python. Python is also useful for doing calculations that involve lots of different values. Although none of the calculations in this problem were particularly difficult, keeping track of all the values as variables in Python can cut down on errors when there are multiple calculations and many parameters to keep track of.

In []:

3 - Linear Regression and Machine Learning

Multiple Linear Regression

In the following example, we will use multiple linear regression to predict the Output of a fictitious System.

By using 2 independent variables/inputs, namely Input_1 and Input_2 and one System Output.

It is assumed that inputs to the System are read every one hour and the output is recorded.

We will use Tkinter which is a Graphical User Interface (GUI). This will facilitate the inputs of the data by the users.

So it makes sense to create for them a simple interface, via widgets, where they can manage the data in a simplified

manner, and this is why we will use Tkinter objects for this purpose to enter the values and to display the predicted results.

Also displayed the Ordinary Least-Squares (OLS) statistics and our equation constant and coefficients.

OLS is defined as follows:

Ordinary least-squares (OLS) regression is a generalized linear modelling technique that may be used to model a single response variable which has been recorded on at least an interval scale. The technique may be applied to single or multiple explanatory variables and also categorical explanatory variables that have been appropriately coded. Please refer to the many available books for statistics for further explanation of OLS.

Real data can be used by reading "csv" or "Excel" files. The data here can be easily modified to suit your application,

for example reading the power output every hour from a power plant for a long period of time, say 24 Hrs.

In the example below, we have two inputs (Input_1 and Input_2) and one output. The time is 24Hrs period.

In [2]:

```

from pandas import DataFrame
from sklearn import linear_model
import tkinter as tk
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

System = {'Hrs': [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24],
          'Input_1':[2.75,2.5,2.5,2.5,2.5,2.5,2.5,2.25,2.25,2.25,2,2,2,1.75,1.75,1.75,
                    1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.75,1.75],
          'Input_2':[5.3,5.3,5.3,5.3,5.4,5.6,5.5,5.5,5.5,5.6,5.7,5.9,6,5.9,
                    5.8,6.1,6.2,6.1,6.1,6.1,5.9,6.2,6.2,6.1],
          'Output': [1464,1394,1357,1293,1256,1254,1234,1195,1159,1167,1130,1075,1047,
                    965,943,958,971,949,884,866,876,822,704,719]
          }

df = DataFrame(System,columns=['Hrs', 'Input_1', 'Input_2', 'Output'])

# Print our Data Table for clarity and check

print(df)

```

	Hrs	Input_1	Input_2	Output
0	1	2.75	5.3	1464
1	2	2.50	5.3	1394
2	3	2.50	5.3	1357
3	4	2.50	5.3	1293
4	5	2.50	5.4	1256
5	6	2.50	5.6	1254
6	7	2.50	5.5	1234
7	8	2.25	5.5	1195
8	9	2.25	5.5	1159
9	10	2.25	5.6	1167
10	11	2.00	5.7	1130
11	12	2.00	5.9	1075
12	13	2.00	6.0	1047
13	14	1.75	5.9	965
14	15	1.75	5.8	943
15	16	1.75	6.1	958
16	17	1.75	6.2	971
17	18	1.75	6.1	949
18	19	1.75	6.1	884
19	20	1.75	6.1	866
20	21	1.75	5.9	876
21	22	1.75	6.2	822
22	23	1.75	6.2	704
23	24	1.75	6.1	719

In [3]:

```
X = df[['Input_1', 'Input_2']].astype(float)

# here we have 2 input variables for multiple regression.

# If you want to use one variable for simple linear regression, then use:

# X = df['Input_1'], for example. Alternatively, you may add additional variables

# within the brackets.

# The output variable (what we are trying to predict).

Y = df['Output'].astype(float)

# Using sklearn

regr = linear_model.LinearRegression()
regr.fit(X, Y)

print('Intercept: \n', regr.intercept_)
print('Coefficients: \n', regr.coef_)
```

Intercept:

1798.4039776258546

Coefficients:

[345.54008701 -250.14657137]

Our Equation for our Linear Regression

From the Intercept and Coefficients, our equation becomes as follows:

The Equation is:

Output = Intercept + Input_1 * X1 + Input_2 * X2

Displaying Ordinary Least-Squares (OLS) Statistics

Here we need to import the stat models as shown below and printing statistics summary of the OLS.

In [4]:

```
import statsmodels.api as sm

X = sm.add_constant(X)
model = sm.OLS(Y, X).fit()
predictions = model.predict(X)

print_model = model.summary()

print(print_model)
```

C:\Users\ORANGE\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:2389: FutureWarning: Method .ptp is deprecated and will be removed in a future version. Use numpy.ptp instead.
 return ptp(axis=axis, out=out, **kwargs)

OLS Regression Results


```
=====
==
Dep. Variable:          Output    R-squared:            0.8
98
Model:                  OLS      Adj. R-squared:       0.8
88
Method:                 Least Squares  F-statistic:          92.
07
Date:                   Tue, 03 Sep 2019  Prob (F-statistic):    4.04e-
11
Time:                   14:05:01   Log-Likelihood:       -134.
61
No. Observations:      24      AIC:                  27
5.2
Df Residuals:          21      BIC:                  27
8.8
Df Model:               2
Covariance Type:       nonrobust
=====
==

```

	coef	std err	t	P> t	[0.025	0.97
5]						
--						
const	1798.4040	899.248	2.000	0.059	-71.685	3668.4
93						
Input_1	345.5401	111.367	3.103	0.005	113.940	577.1
40						
Input_2	-250.1466	117.950	-2.121	0.046	-495.437	-4.8
56						
=====						
==						
Omnibus:		2.691	Durbin-Watson:			0.5
30						
Prob(Omnibus):		0.260	Jarque-Bera (JB):			1.5
51						
Skew:		-0.612	Prob(JB):			0.4
61						
Kurtosis:		3.226	Cond. No.			39
4.						
=====						
==						

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.



From OLS table above we get the constant value as well as the intercept and the coefficients for our equation.

In []:

```

# Using Tkinter Object(s) or widgets for our Graphical User Interface (GUI)

root= tk.Tk()

canvas1 = tk.Canvas(root, width = 500, height = 300)
canvas1.pack()

Intercept_result = ('Intercept: ', regr.intercept_)
label_Intercept = tk.Label(root, text=Intercept_result, justify = 'center')
canvas1.create_window(260, 220, window=label_Intercept)

Coefficients_result = ('Coefficients: ', regr.coef_)
label_Coefficients = tk.Label(root, text=Coefficients_result, justify = 'center')
canvas1.create_window(260, 240, window=label_Coefficients)

# New_Input_1 Label and input box

label1 = tk.Label(root, text='Type Input_1: ')
canvas1.create_window(100, 100, window=label1)

entry1 = tk.Entry (root) # create 1st entry box
canvas1.create_window(270, 100, window=entry1)

# New_Input_2 Label and input box

label2 = tk.Label(root, text=' Type Input_2: ')
canvas1.create_window(120, 120, window=label2)

entry2 = tk.Entry (root) # create 2nd entry box
canvas1.create_window(270, 120, window=entry2)

def values():
    global New_Input_1 # our 1st input variable
    New_Input_1 = float(entry1.get())

    global New_Input_2 # our 2nd input variable
    New_Input_2 = float(entry2.get())

    Prediction_result = ('Predicted Output: ', regr.predict([[New_Input_1 ,New_Input_2]]))
    label_Prediction = tk.Label(root, text= Prediction_result, bg='orange')
    canvas1.create_window(260, 280, window=label_Prediction)

# Using button to call the 'values' command above

button1 = tk.Button (root, text='Predict Output',command=values, bg='orange')

canvas1.create_window(270, 150, window=button1)

# plot 1st scatter

figure3 = plt.Figure(figsize=(5,4), dpi=100)
ax3 = figure3.add_subplot(111)
ax3.scatter(df['Input_1'].astype(float),df['Output'].astype(float), color = 'r')

```

```
scatter3 = FigureCanvasTkAgg(figure3, root)
scatter3.get_tk_widget().pack(side=tk.RIGHT, fill=tk.BOTH)
ax3.legend()
ax3.set_xlabel('Input_1')
ax3.set_title('Input_1 Vs. Output')

# plot 2nd scatter

figure4 = plt.Figure(figsize=(5,4), dpi=100)
ax4 = figure4.add_subplot(111)
ax4.scatter(df['Input_2'].astype(float),df['Output'].astype(float), color = 'g')
scatter4 = FigureCanvasTkAgg(figure4, root)
scatter4.get_tk_widget().pack(side=tk.RIGHT, fill=tk.BOTH)
ax4.legend()
ax4.set_xlabel('Input_2')
ax4.set_title('Input_2 Vs. Output')

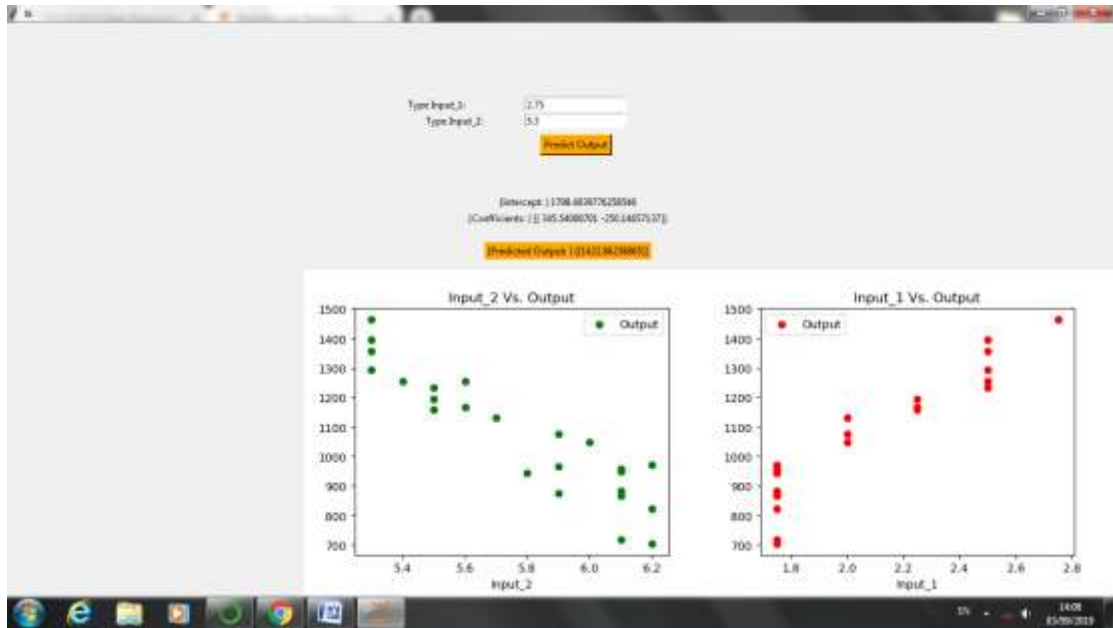
root.mainloop()
```

Run the program and the Tkinter window shown below is displayed.

To check, we have entered the values 2.75 for Input_1 and 5.3 for Input_2.

The output or the predicted output is shown to be 1422.862 which is very close to the output value in the above data.

In []:



Tkinter window for the program

Machine Learning

What is Machine Learning? A definition.

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide.

The primary aim is to allow the computers to learn automatically without human intervention or assistance and adjust actions accordingly.

Some machine learning methods:

Machine learning algorithms are often categorized as supervised or unsupervised.

Supervised machine learning algorithms can apply what has been learned in the past to new data using labeled examples to predict future events.

Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training.

The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.

In contrast, unsupervised machine learning algorithms are used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data.

The system doesn't figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.

Semi-supervised machine learning algorithms fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data. The systems that use this method are able to considerably improve learning accuracy.

Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in order to train it / learn from it. Otherwise, acquiring unlabeled data generally doesn't require additional resources.

Reinforcement machine learning algorithms is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behavior within a specific context in order to maximize its performance.

Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal. Machine learning enables analysis of massive quantities of data. While it generally delivers faster, more accurate results in order to identify profitable opportunities or dangerous risks, it may also

require additional time and resources to train it properly. Combining machine learning with AI and cognitive technologies can make it even more effective in processing large volumes of information.

Here, I we will use Decision Tree and Random Forest algorithm as an examples of machine learning.

To plot the decision tree, we need to download the package 'graphviz'.

Using Anaconda prompt with the internet connected,type:

```
conda install graphviz.
```

```
graphviz version is 2.38. or later.
```

Also we need to download 'pydotplus' using the same procedure above, type, conda install pydotplus.

```
Pydotplus version is 2.0.2 or later.
```

Adding Path To Graphviz

Inspite of the fact that graphviz will be successfully downloaded using Anaconda prompt, it is noticed that graphviz will not

work with Error message displayed " cannot find graphviz executable file".

To solve this problem, Path should be added manually as follows:

From the Control Panel go to System and Security then System then Advanced Setting then Environment Variables, and:

Click on Path and click on Edit.

Go to the end of the line and put ; (is a must), then write after ; the following:

```
C:\Users\ORANGE\Anaconda3\Library\bin\graphviz
```

NOTE: IN MY COMPUTER I HAVE THE main folder titled 'ORANGE' BUT YOURS IS DIFFERENT.

In [4]:

```
# Decision Tree example

# import required packages

from sklearn.tree import DecisionTreeClassifier
import pandas as pd
import numpy as np
from sklearn import tree
import collections

# Data here is given as an example, but nevertheless it reflects real situation.

# create a matrix including the data

# Data: All Data here are fictitious

# Pressure(psi) Height(in) Label
# 8 8 High
# 50 40 High
# 8 9 Low
# 15 12 High
# 9 9.8 Low

# Creat Data

data = [[8,8,'High'],[50,40,'High'],[8,9,'Low'],[15,12,'High'],[9,9.8,'Low']]

# generating a dataframe from the matrix

df = pd.DataFrame(data, columns = ['Pressure','Height','Label'])

# defining predictors, here we need to define what are the predictors i.e the inputs

X = df[['Pressure','Height']]

# definig the target variable and mapping it to 1 for High and 0 for Low

y = df['Label'].replace({'High':1, 'Low':0})

# instantiating the model

tree = DecisionTreeClassifier()

# fitting the model

model = tree.fit(X,y)
```

In [2]:

```

# Plotting the Tree

from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
import pydotplus
from IPython.display import Image

dot_data = StringIO()

export_graphviz(
    model,
    out_file = dot_data,
    filled=True, rounded=True, proportion=False,
    special_characters=True,
    feature_names=X.columns,
    class_names=["High", "Low"])

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

# now how we change the Tree Colors

# //////////////////////////////////Tree Color Changing //////////////////////////////////

#graph = pydotplus.graph_from_dot_data(dot_data)
nodes = graph.get_node_list()
edges = graph.get_edge_list()

colors = ('yellow', 'blue')
edges = collections.defaultdict(list)

for edge in graph.get_edge_list():
    edges[edge.get_source()].append(int(edge.get_destination()))

for edge in edges:
    edges[edge].sort()
    for i in range(2):
        dest = graph.get_node(str(edges[edge][i]))[0]
        dest.set_fillcolor(colors[i])

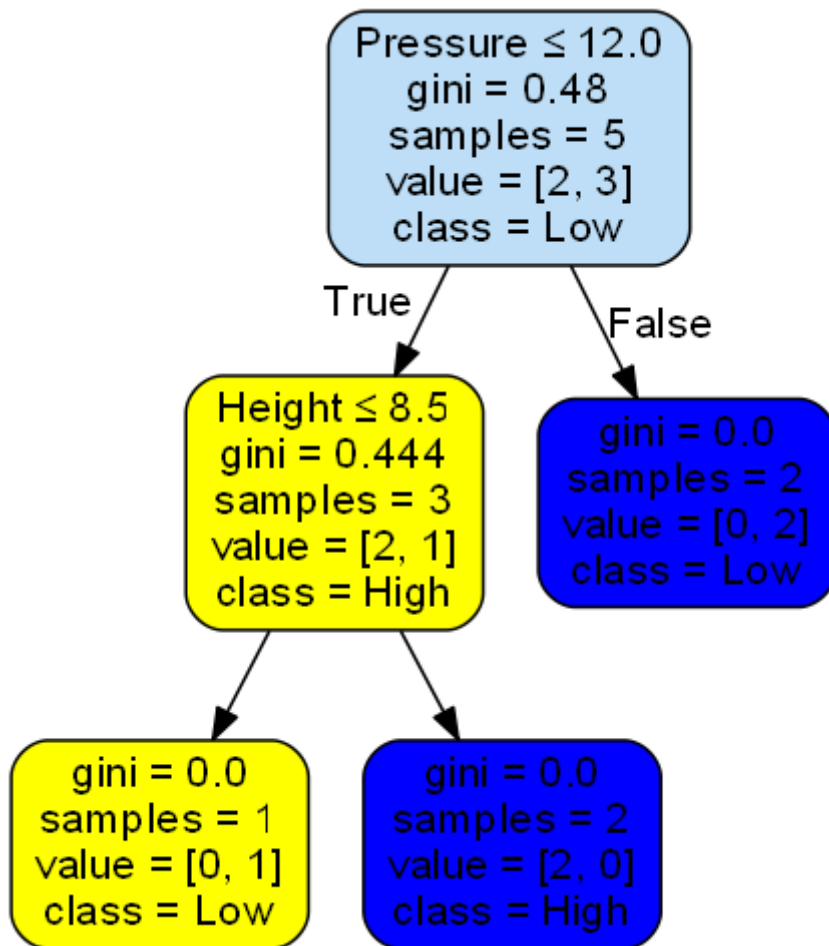
# //////////////////////////////////

# Here how we can modify the size of the tree displayed

graph.write_png('original_tree.png')
graph.set_size('5,5!')
graph.write_png('resized_tree.png')
# Show Tree
Image(graph.create_png())

```

Out[2]:



In [5]:

```

# Prediction
a=np.array
prediction = model.predict([[5,9]])
if prediction == 1:
    print("High")
else:
    print("Low")
  
```

Low

In []:

In []:

Random Forest

is flexible, easy to use machine learning algorithm that produces great results most of the time.

It is also one of the most used algorithms because it's simplicity and the fact that it can be used for both classification and regression tasks which form the majority of current machine learning systems.

We will discuss random forest in classification, since classification is sometimes considered the building block of machine learning.

In the example, we are going to use Iris dataset already available, within the many datasets in sklearn. The Iris flower dataset is rather small (consisting of only 150 evenly distributed samples), and is well behaved which makes it ideal for this study.

Iris dataset is used for flower classification.

Iris has four flower species identified as:

1- Setosa

2- Versicolor

3- Virginica

The flowers are identified by it's petal length and width as well as the sepal

length and width, all measured in cm.

For other applications, the dataset can be easily modified to suit your application

and can be loaded easily from Excel or text documents.

In [2]:

```
# First we need to Import scikit-Learn dataset Library

from sklearn import datasets

# Load dataset whuch is Iris in this case

iris = datasets.load_iris()
```

In [3]:

```
# print the Labels species, they are setosa, versicolor and virginica.
# The Labels are in the dataset

print(iris.target_names)

# print the names of the four features

print(iris.feature_names)
```

```
['setosa' 'versicolor' 'virginica']
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
(cm)']
```

In [4]:

```
# Print the iris data (only the first 5 records), just to make sure that
# We have loaded the dataset correctly
# These are petal and sepal Length and width all in cm
```

```
print(iris.data[0:5])
```

```
# The iris species are encoded as:
# (0:setosa, 1:versicolor, 2:virginica)
```

```
print(iris.target)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

In [5]:

```
# Create a DataFrame for the given iris dataset and print
# Some rows of the dataset.
```

```
import pandas as pd
```

```
data=pd.DataFrame({
    'sepal length':iris.data[:,0],
    'sepal width':iris.data[:,1],
    'petal length':iris.data[:,2],
    'petal width':iris.data[:,3],
    'species':iris.target
})
```

```
data.head()
```

Out[5]:

	sepal length	sepal width	petal length	petal width	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [6]:

```
# The data must be divided into training and test sets,  
# We need to import train_test_split function  
  
from sklearn.model_selection import train_test_split  
  
# These are the Features  
  
X=data[['sepal length', 'sepal width', 'petal length', 'petal width']]  
y=data['species'] # These are the Labels  
  
# Split dataset into training set and test set  
  
# Divide the dataset into 70% training and 30% testsets  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

In [7]:

```
#Import Random Forest Model  
  
from sklearn.ensemble import RandomForestClassifier  
  
#Create a Gaussian Classifier  
  
clf=RandomForestClassifier(n_estimators=100)  
  
#Train the model using the training sets y_pred=clf.predict(X_test)  
clf.fit(X_train,y_train)  
  
y_pred=clf.predict(X_test)
```

In [8]:

```
#Import scikit-Learn metrics module for accuracy calculation  
  
from sklearn import metrics  
  
# Check Model Accuracy, how often is the classifier correct?  
  
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Accuracy: 1.0

In [9]:

```
# Check prediction  
  
flower=[3,5,4,2] # petal, sepal length and width in cm  
  
class_code = clf.predict([[3,5,4,2]])  
  
print(class_code)
```

[2]

In [10]:

```
import numpy as np
from sklearn.preprocessing import LabelEncoder
labels = np.asarray(iris.feature_names)
le = LabelEncoder()
le.fit(labels)
```

```
# apply encoding to labels
```

```
labels = le.transform(labels)
print(labels)
```

```
[2 3 0 1]
```

In [11]:

```
# we have encoded the classes, now Decoding The Class
```

```
decoded_class = le.inverse_transform(class_code)
```

```
print (decoded_class)
```

```
['sepal length (cm)']
```

In [12]:

```
import pandas as pd
feature_imp=pd.Series(clf.feature_importances_,
                     index=iris.feature_names).sort_values(ascending=False)
feature_imp
```

Out[12]:

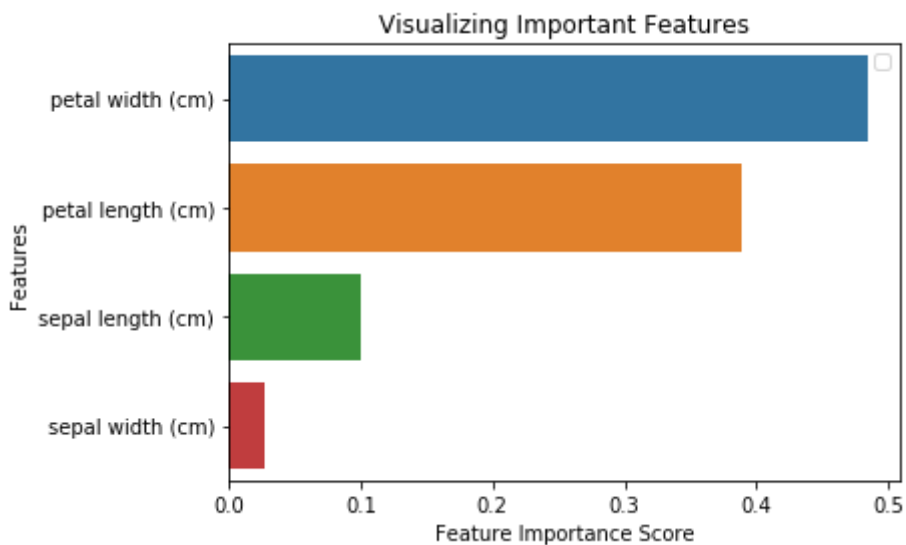
```
petal width (cm)      0.484043
petal length (cm)    0.388840
sepal length (cm)    0.100376
sepal width (cm)     0.026741
dtype: float64
```

In [13]:

```
# Make a bar plot to see the importance of the features

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# Creating a bar plot
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add Labels to your graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.legend()
plt.show()
```

No handles with labels found to put in legend.



In [14]:

```
# From the above bar chart, it can be seen that the Feature sepal width has the
# Lower importance among the others.
# So, we can remove and select the remaining 3-features.
```

In [15]:

```
# We will generate a model on the selected remaining features and train the model
# Import train_test_split function
from sklearn.model_selection import train_test_split
# Split dataset into features and Labels
# Removed feature is "sepal Length"
X=data[['petal length', 'petal width', 'sepal length']]
y=data['species']
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.70, random_state=5)
```

In [16]:

```
# N_estimators in Random Forest

# n_estimators represents the number of trees in the forest. Usually the higher the number
# trees the better to learn
# the data. However, adding a lot of trees can slow down the training process considerably.
# N_estimator is between 50 and 200.

# Create a Gaussian Classifier

clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)

clf.fit(X_train,y_train)

# prediction on test set

y_pred=clf.predict(X_test)

#Import scikit-learn metrics module for accuracy calculation

from sklearn import metrics

# Model Accuracy, how often is the classifier correct?

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Accuracy: 0.9523809523809523

In [17]:

```
# From the above accuracy, it is seen that after removing the Least important feature
# (sepal length), the accuracy of our model is increased.
# This is because we removed a noise data.
```

Using Decision Tree

In [21]:

```
# We will re-import the necessary packages (libraries) for clarity

import sklearn.datasets as datasets
import pandas as pd

# We have Loaded Iris DataSet already
iris=datasets.load_iris()

# Our DataFrame already given as data above

y=iris.target
```

In [23]:

```
from sklearn.tree import DecisionTreeClassifier
dtree=DecisionTreeClassifier()
dtree.fit(X_train,y_train)
```

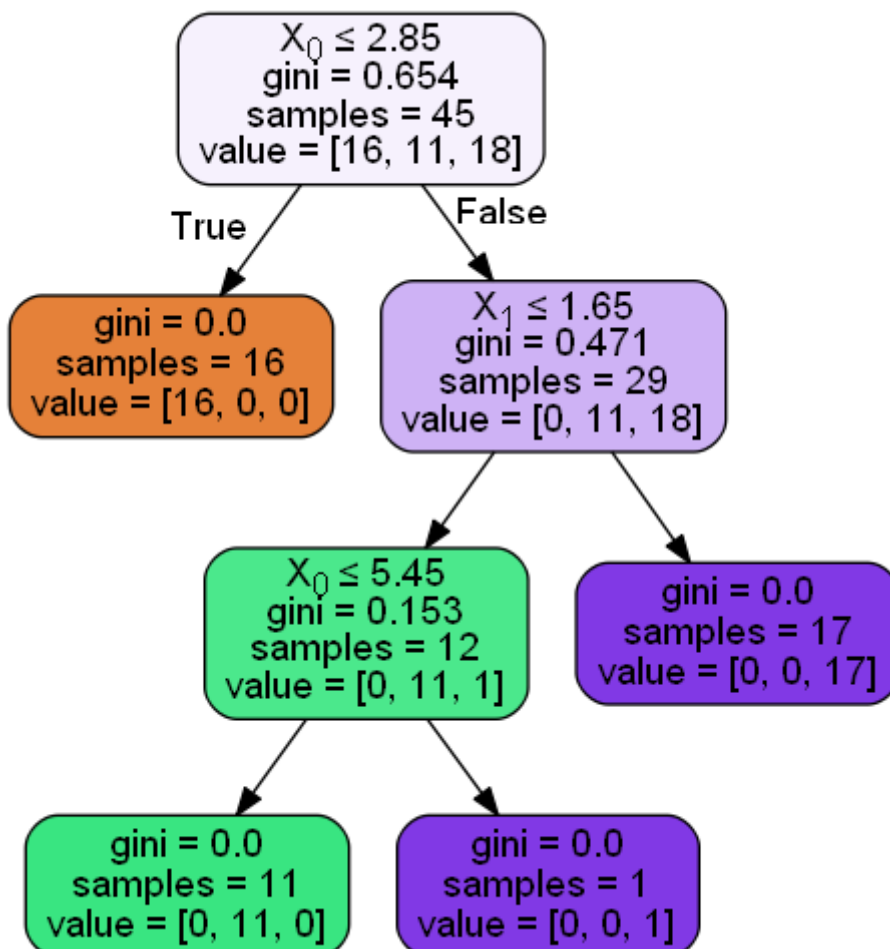
Out[23]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False, random_state=None,
splitter='best')
```

In [24]:

```
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(dtree, out_file=dot_data,
filled=True, rounded=True,
special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[24]:



4 - Symbolic Math (SymPy) and ODE

Algebra and Symbolic Math with SymPy

The mathematical problems and solutions in programs have all involved the manipulation of numbers. But there's another way math is taught, learned, and practiced, and that's in terms of symbols and the operations between them.

Just think of all the x's and y's in a typical algebra problem. We refer to this type of math as symbolic math. You know the factorize of $x^3 + 3x^2 + 3x + 1$ problems in math class. We'll use SymPy—a Python library that lets you write expressions containing symbols and perform operations on them.

The SymPy home page is:

<http://sympy.org>

and provides the full (and up-to-date) documentation for this library.

In [1]:

```
# To Start With, first Call init_printing.  
# This tells sympy to display expressions in a nicer format.  
  
import sympy  
sympy.init_printing()
```

In [4]:

```
# Find 2*x-x as below  
2 * x - x
```

Out[4]:

x

In [5]:

```
# Find the results of the following:  
  
y = Symbol('y')  
  
x + y + x + 10*y
```

Out[5]:

2x + 11y

In [6]:

```
# We can abbreviate the creation of multiple symbolic variables using the symbols function.  
# For example, to create the symbolic variables x, y and z, we can use:  
  
# Note:  
  
x, y, z = sympy.symbols('x,y,z')  
# And evaluate the following expression:  
x + 2*y + 3*z - x
```

Out[6]:

$2y + 3z$

In [7]:

```
# Once we have completed our term manipulation, we sometimes  
# like to insert numbers for variables.  
  
# This can be done using the subs method.  
# Find x+2*y for x = 10  
  
(x + 2*y).subs(x, 10)
```

Out[7]:

$2y + 10$

In [8]:

```
# Find:  
(x + 2*y).subs(x, 10).subs(y, 3)
```

Out[8]:

16

In [9]:

```
# We can also substitute a symbolic variable for another one such as in this  
# example where y is replaced  
# with x before we substitute x with the number 2.  
  
myterm = 3*x + y**2  
myterm
```

Out[9]:

$3x + y^2$

In [10]:

```
myterm.subs(x, y).subs(y, 2)
```

Out[10]:

10

In [11]:

```
# Numeric types:

# SymPy has the numeric types Rational and Real Number.
# The Rational class represents a rational number as a pair
# of two integers: the numerator and the denominator, so Rational(1,2) represents 1/2,
# Rational(5,2) represents 5/2 and so on.

from sympy import Rational # We have already imported SymPy.
a = Rational(1, 10)
# Find a
a
```

Out[11]:

$$\frac{1}{10}$$

In [12]:

```
b = Rational(45, 67)
b
```

Out[12]:

$$\frac{45}{67}$$

In [13]:

```
a-b
```

Out[13]:

$$-\frac{383}{670}$$

In [17]:

```
# Note that the Rational class works with rational expressions exactly.
# This is in contrast to Python's standard float data type which
# uses floating point representation to
# approximate (rational) numbers.

# We can convert the sympy.Rational type into a Python floating point variable
# using float or the evalf method of the Rational object.
# The evalf method can take an argument that specifies how many digits
# should be computed for the floating point approximation (not all of those may be used by
# floating point type of course).

c = Rational(2, 3)
c
```

Out[17]:

$$\frac{2}{3}$$

In [18]:

```
float(c)
```

Out[18]:

0.6666666666666666

In [19]:

```
c.evalf(20) # To 20 digits
```

Out[19]:

0.66666666666666666667

In [17]:

```
# Differentiation and Integration  
# SymPy is capable of carrying out differentiation and integration of many functions:
```

```
from sympy import Symbol, exp, sin, sqrt, diff  
x = Symbol('x')  
y = Symbol('y')  
diff(sin(x), x)
```

Out[17]:

cos(x)

In [18]:

```
diff(10 + 3*x + 4*y + 10*x**2 + x**9, x)
```

Out[18]:

$9x^8 + 20x + 3$

In [19]:

```
diff(10 + 3*x + 4*y + 10*x**2 + x**9, y)
```

Out[19]:

4

In [20]:

```
diff(10 + 3*x + 4*y + 10*x**2 + x**9, x).subs(x,1)
```

Out[20]:

32

In [21]:

```
diff(10 + 3*x + 4*y + 10*x**2 + x**9, x).subs(x,1.5)
```

Out[21]:

263.66015625

In [22]:

```
diff(diff(3*x**4*y**7, x, x), y, y)
```

Out[22]:

$$1512x^2y^5$$

In [23]:

```
r = sqrt(x**2 + y**2)
sigma = Symbol('σ')
def phi(x,y,sigma):
    return sqrt(x**2 + y**2 + sigma**2)
```

```
mydfdx= x / sqrt(r**2 + sigma**2)
print(diff(phi(x, y, sigma), x))
```

$$x/\sqrt{x^2 + y^2 + \sigma^2}$$

In [24]:

```
# Integration uses a similar syntax.
# For the indefinite case, specify the function and the variable with respect
# to which the integration is performed:
```

```
from sympy import integrate
integrate(x**2, x)
```

Out[24]:

$$\frac{x^3}{3}$$

In [25]:

```
integrate(x**2, (x, 0, 2))
```

Out[25]:

$$\frac{8}{3}$$

In [26]:

```
float(integrate(x**2, (x, 0, 2)))
```

Out[26]:

$$2.6666666666666665$$

Ordinary differential equations:

SymPy has inbuilt support for solving several kinds of ordinary differential equation via its `dsolve` command. We need to set up the ODE and pass it as the first argument, `eq`. The second argument is the function $f(x)$ to solve for. An optional third argument is `hint`, influences the method that `dsolve` uses. some methods are better-suited to certain classes of ODEs, or will express the solution more simply, than others.

To set up the ODE solver, we need a way to refer to the unknown function for which we are solving, as well as its derivatives. The `Function` and `Derivative` classes facilitate this.

In [27]:

```
from sympy import Symbol, dsolve, Function, Derivative, Eq
y = Function("y")
x = Symbol('x')
y_ = Derivative(y(x), x)
dsolve(y_ + 5*y(x), y(x))
```

Out[27]:

$$y(x) = C_1 e^{-5x}$$

In [29]:

```
# Note how dsolve has introduced a constant of integration, C1.
# It will introduce as many constants as are
# required, and they will all be named Cn, where n is an integer.
# Note also that the first argument to dsolve is taken to be equal to
# zero unless we use the Eq() function to specify otherwise:
```

```
dsolve(Eq(y_ + 5*y(x), 12), y(x))
```

Out[29]:

$$y(x) = \frac{C_1 e^{-5x}}{5} + \frac{12}{5}$$

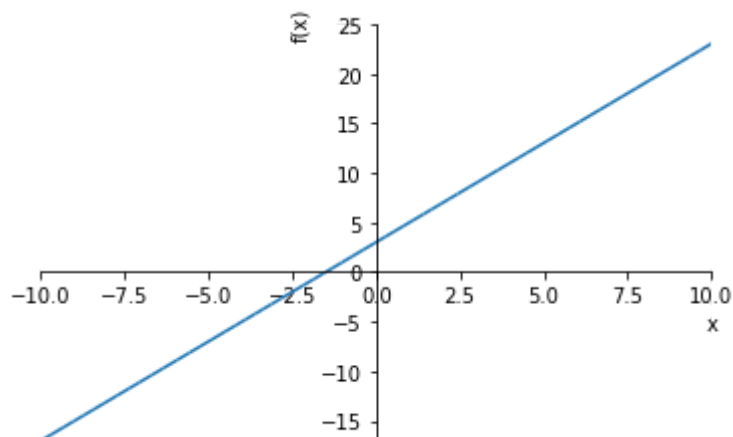
In []:

Plotting Using SymPy

In [4]:

```
# With SymPy, you can just tell SymPy the equation of the line you want to plot,
# and the graph will be created for you. Let's plot a line whose equation is given by
#  $y = 2x + 3$ :
```

```
from sympy.plotting import plot
from sympy import Symbol
x = Symbol('x')
plot(2*x+3)
# Note that we didn't have to call the show() function to show the graphs
because this is done automatically by SymPy.
```



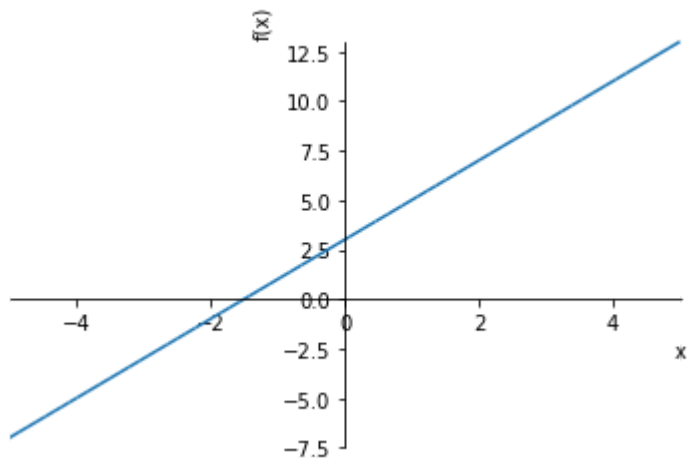
Out[4]:

```
<sympy.plotting.plot.Plot at 0x8c95cf0>
```

In [5]:

```
# Now, let's say that you wanted to limit the values of 'x' in the preceding graph to lie
# in the range -5 to 5.
# (instead of -10 to 10). You'd do that as follows:
```

```
plot((2*x + 3), (x, -5, 5))
```



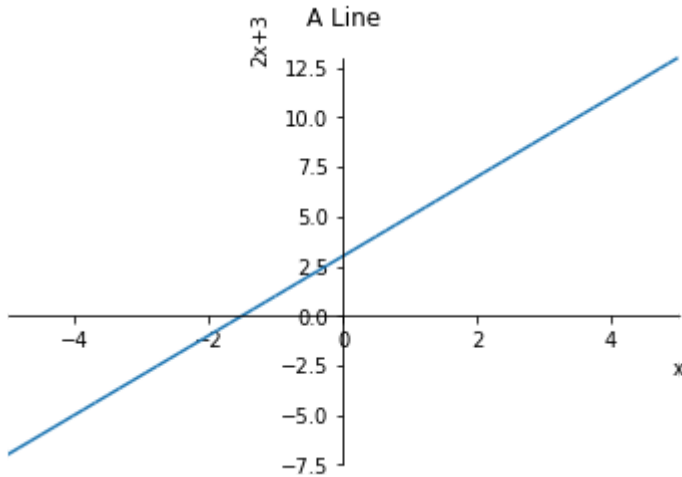
Out[5]:

```
<sympy.plotting.plot.Plot at 0x8c95190>
```

In [7]:

```
# You can use other keyword arguments in the plot() function, such as title to enter
# a title or xlabel and ylabel
# to label the x-axis and the y-axis, respectively.
# The following plot() function specifies the preceding three keyword arguments.

plot(2*x + 3, (x, -5, 5), title='A Line', xlabel='x', ylabel='2x+3')
```



Out[7]:

```
<sympy.plotting.plot.Plot at 0x8d67990>
```

In [8]:

```
# The show keyword argument mentioned above allows us to specify whether
# we want the graph to be displayed.
# Passing show=False will cause the graph to not be displayed when you
# call the plot() function:
p = plot(2*x + 3, (x, -5, 5), title='A Line', xlabel='x', ylabel='2x+3', show=False)
```

In []:

```
# You will see that no graph is shown. The label p refers to the plot that is
# created, so you can now call p.show() to display the graph. You can also save
# the graph as an image file using the save() method, as follows:
```

```
p.save('line.png')
```

```
# This will save the plot to a file line.png in the current directory.
```

Plotting Expressions Input by the User

The expression that you pass to the `plot()` function must be expressed in terms of x only. For example, earlier we plotted $y = 2x + 3$, which we entered to the plot function as simply $2x + 3$. If the expression were not originally in this form, we'd have to rewrite it. Of course, we could do this manually, outside the program. But what if you want to write a program that allows its users to graph any expression? If the user enters an expression in the form of $2x + 3y - 6$, say, we have to first convert it. The `solve()` function will help us here. Let's see an example below:

In [11]:

```
from sympy import solve, sympify
expr = input('Enter an expression: ')
# Say, we will enter the expression: 2*x + 3*y - 6
expr = sympify(expr)
y = Symbol('y')
solve(expr, y)
```

Enter an expression: 2*x+3*y-6

Out[11]:

```
[-2*x/3 + 2]
```

In the above example, we use the `sympify()` function to convert the input expression to a SymPy object and we create a Symbol object to represent 'y' so that we can tell SymPy which variable we want to solve the equation for. Then we solve the expression to find y in terms of x by specifying y as the second argument to the `solve()` function. Solution, is returned in terms of x, which is what we need for plotting.

Notice that this final expression is stored in a list, so before we can use it, we'll have to extract it from the list as below:

In [12]:

```
solutions = solve(expr, 'y')
expr_y = solutions[0]
expr_y
```

Out[12]:

```
-2*x/3 + 2
```

Plotting Multiple Functions

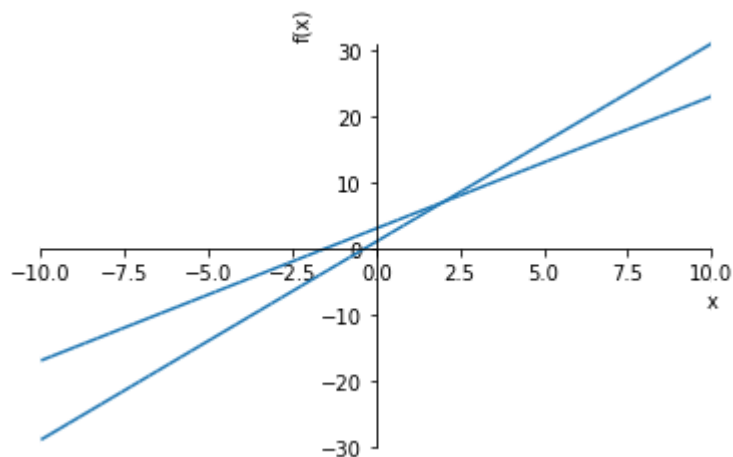
You can enter multiple expressions when calling the SymPy plot function to plot more than one expression on the same graph. For example, the following code plots two lines at once, below:

In [21]:

```

from sympy.plotting import plot
from sympy import Symbol
x = Symbol('x')
plot(2*x+3, 3*x+1)

```



Out[21]:

```
<sympy.plotting.plot.Plot at 0x8e8b910>
```

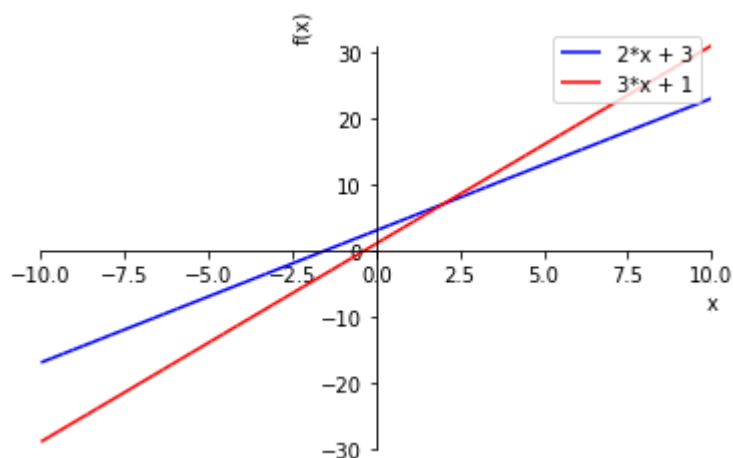
This example brings out another difference between plotting in matplotlib and in SymPy. Here, using SymPy, both lines are the same color, whereas matplotlib would have automatically made the lines different colors. To set different colors for each line with SymPy, we'll need to perform some extra steps, as shown in the following code, which also adds a legend to the graph:

In [22]:

```

from sympy.plotting import plot
from sympy import Symbol
x = Symbol('x')
p = plot(2*x+3, 3*x+1, legend=True, show=False)
p[0].line_color = 'b'
p[1].line_color = 'r'

```



Further Examples In SymPy

In [5]:

```
# Compute  $\int (e^x \sin(x) + e^x \cos(x)) dx$ 
integrate(exp(x)*sin(x) + exp(x)*cos(x), x)
```

Out[5]:

$$e^x \sin(x)$$

In [6]:

```
# Compute  $\int \sin(x^2) dx$  from  $(-\infty)$  to  $(+\infty)$ 
integrate(sin(x**2), (x, -oo, oo))
```

Out[6]:

$$\frac{\sqrt{2}\sqrt{\pi}}{2}$$

In [7]:

```
# Find  $\lim(\sin(x)/x)$  when  $x \rightarrow 0$ 
limit(sin(x)/x, x, 0)
```

Out[7]:

1

In [8]:

```
# Solve  $x^2 - 2 = 0$ 
solve(x**2 - 2, x)
```

Out[8]:

$$[-\sqrt{2}, \sqrt{2}]$$

In [10]:

```
# Solve the differential equation  $y'' - y = e^t$ 
y = Function('y')
dsolve(Eq(y(t).diff(t, t) - y(t), exp(t)), y(t))
```

Out[10]:

$$y(t) = C_2 e^{-t} + \left(C_1 + \frac{t}{2}\right) e^t$$

In [11]:

```
# Find the eigenvalues of  $\begin{bmatrix} 1 & 2 \\ 2 & 2 \end{bmatrix}$ 
Matrix([[1, 2], [2, 2]]).eigenvals()
```

Out[11]:

$$\left\{ \frac{3}{2} + \frac{\sqrt{17}}{2} : 1, \quad -\frac{\sqrt{17}}{2} + \frac{3}{2} : 1 \right\}$$

In [12]:

```
# Rewrite the Bessel function Jv(z) in terms of the spherical Bessel function jv(z)
besselj(nu, z).rewrite(jn)
```

Out[12]:

$$\frac{\sqrt{2}\sqrt{z}j_{\nu-\frac{1}{2}}(z)}{\sqrt{\pi}}$$

In [30]:

```
import sympy
sympy.factorial(40)
```

Out[30]:

815915283247897734345611269596115894272000000000

In [22]:

```
"%.25f" % 0.3 # create a string representation with 25 decimals
```

Out[22]:

'0.29999999999999999888977698'

In [23]:

```
sympy.Float(0.3, 25)
```

Out[23]:

0.29999999999999999888977698

In [24]:

```
sympy.Float('0.3', 25)
```

Out[24]:

0.3

Simplification

In [66]:

```
expr = 2 * (x**2 - x) - x * (x + 1)
```

In [67]:

```
expr
```

Out[67]:

$$2x^2 - x(x + 1) - 2x$$

In [68]:

```
sympy.simplify(expr)
```

Out[68]:

$$x(x - 3)$$

In [69]:

```
expr.simplify()
```

Out[69]:

$$x(x - 3)$$

In [70]:

```
expr
```

Out[70]:

$$2x^2 - x(x + 1) - 2x$$

In [71]:

```
expr = 2 * sympy.cos(x) * sympy.sin(x)
```

In [72]:

```
expr
```

Out[72]:

$$2 \sin(x) \cos(x)$$

In [73]:

```
sympy.trigsimp(expr)
```

Out[73]:

$$\sin(2x)$$

In [74]:

```
expr = sympy.exp(x) * sympy.exp(y)
```

In [75]:

```
expr
```

Out[75]:

$$e^x e^y$$

In [76]:

```
sympy.powsimp(expr)
```

Out[76]:

e^{x+y}

Expand

In [77]:

```
expr = (x + 1) * (x + 2)
```

In [78]:

```
sympy.expand(expr)
```

Out[78]:

$x^2 + 3x + 2$

In [79]:

```
sympy.sin(x + y).expand(trig=True)
```

Out[79]:

$\sin(x) \cos(y) + \sin(y) \cos(x)$

In [80]:

```
a, b = sympy.symbols("a, b", positive=True)
```

In [81]:

```
sympy.log(a * b).expand(log=True)
```

Out[81]:

$\log(a) + \log(b)$

In [82]:

```
sympy.exp(I*a + b).expand(complex=True)
```

Out[82]:

$i e^b \sin(a) + e^b \cos(a)$

In [83]:

```
sympy.expand((a * b)**x, power_exp=True)
```

Out[83]:

$a^x b^x$

In [84]:

```
sympy.exp(I*(a-b)*x).expand(power_exp=True)
```

Out[84]:

$$e^{iax} e^{-ibx}$$

Factor

In [85]:

```
sympy.factor(x**2 - 1)
```

Out[85]:

$$(x - 1)(x + 1)$$

In [86]:

```
sympy.factor(x * sympy.cos(y) + sympy.sin(z) * x)
```

Out[86]:

$$x(\sin(z) + \cos(y))$$

In [87]:

```
sympy.logcombine(sympy.log(a) - sympy.log(b))
```

Out[87]:

$$\log\left(\frac{a}{b}\right)$$

In [88]:

```
expr = x + y + x * y * z
```

In [89]:

```
expr.factor()
```

Out[89]:

$$xyz + x + y$$

In [90]:

```
expr.collect(x)
```

Out[90]:

$$x(yz + 1) + y$$

In [91]:

```
expr.collect(y)
```

Out[91]:

$$x + y(xz + 1)$$

In [92]:

```
expr = sympy.cos(x + y) + sympy.sin(x - y)
```

In [93]:

```
expr.expand(trig=True).collect([sympy.cos(x),sympy.sin(x)]).collect(sympy.cos(y)-sympy.sin(y))
```

Out[93]:

$$(\sin(x) + \cos(x))(-\sin(y) + \cos(y))$$

Together, apart, cancel

In [94]:

```
sympy.apart(1/(x**2 + 3*x + 2), x)
```

Out[94]:

$$-\frac{1}{x + 2} + \frac{1}{x + 1}$$

In [95]:

```
sympy.together(1 / (y * x + y) + 1 / (1+x))
```

Out[95]:

$$\frac{y + 1}{y(x + 1)}$$

In [96]:

```
sympy.cancel(y / (y * x + y))
```

Out[96]:

$$\frac{1}{x + 1}$$

Numerical evaluation

In [103]:

```
sympy.N(1 + pi)
```

Out[103]:

$$4.14159265358979$$

In [104]:

```
sympy.N(pi, 50)
```

Out[104]:

```
3.1415926535897932384626433832795028841971693993751
```

In [105]:

```
(x + 1/pi).evalf(7)
```

Out[105]:

```
x + 0.3183099
```

In [106]:


```
expr = sympy.sin(pi * x * sympy.exp(x))
```

In [107]:

```
[expr.subs(x, xx).evalf(3) for xx in range(0, 10)]
```

Out[107]:

```
[0, 0.774, 0.642, 0.722, 0.944, 0.205, 0.974, 0.977, -0.87, -0.69]
```



In [108]:

```
expr_func = sympy.lambdify(x, expr)
```

In [109]:

```
expr_func(1.0)
```

Out[109]:

```
0.773942685266709
```

In [110]:

```
expr_func = sympy.lambdify(x, expr, 'numpy')
```

In [111]:

```
import numpy as np
```

In [112]:

```
xvalues = np.arange(0, 10)
```

In [113]:

```
expr_func(xvalues)
```

Out[113]:

```
array([ 0.          ,  0.77394269,  0.64198244,  0.72163867,  0.94361635,
        0.20523391,  0.97398794,  0.97734066, -0.87034418, -0.69512687])
```

Calculus

In [114]:

```
f = sympy.Function('f')(x)
```

In [115]:

```
sympy.diff(f, x)
```

Out[115]:

$$\frac{d}{dx} f(x)$$

In [116]:

```
sympy.diff(f, x, x)
```

Out[116]:

$$\frac{d^2}{dx^2} f(x)$$

In [117]:

```
sympy.diff(f, x, 3)
```

Out[117]:

$$\frac{d^3}{dx^3} f(x)$$

In [118]:

```
g = sympy.Function('g')(x, y)
```

In [119]:

```
g.diff(x, y)
```

Out[119]:

$$\frac{\partial^2}{\partial x \partial y} g(x, y)$$

In [120]:

```
g.diff(x, 3, y, 2) # equivalent to s.diff(g, x, x, x, y, y)
```

Out[120]:

$$\frac{\partial^5}{\partial x^3 \partial y^2} g(x, y)$$

In [121]:

```
expr = x**4 + x**3 + x**2 + x + 1
```

In [122]:

```
expr.diff(x)
```

Out[122]:

$$4x^3 + 3x^2 + 2x + 1$$

In [123]:

```
expr.diff(x, x)
```

Out[123]:

$$2(6x^2 + 3x + 1)$$

In [124]:

```
expr = (x + 1)**3 * y ** 2 * (z - 1)
```

In [125]:

```
expr.diff(x, y, z)
```

Out[125]:

$$6y(x + 1)^2$$

In [126]:

```
expr = sympy.sin(x * y) * sympy.cos(x / 2)
```

In [127]:

```
expr.diff(x)
```

Out[127]:

$$y \cos\left(\frac{x}{2}\right) \cos(xy) - \frac{1}{2} \sin\left(\frac{x}{2}\right) \sin(xy)$$

In [128]:

```
expr = sympy.special.polynomials.hermite(x, 0)
```

In [129]:

```
expr.diff(x).doit()
```

Out[129]:

$$\frac{2^x \sqrt{\pi} \operatorname{polygamma}\left(0, -\frac{x}{2} + \frac{1}{2}\right)}{2\Gamma\left(-\frac{x}{2} + \frac{1}{2}\right)} + \frac{2^x \sqrt{\pi} \log(2)}{\Gamma\left(-\frac{x}{2} + \frac{1}{2}\right)}$$

In [130]:

```
d = sympy.Derivative(sympy.exp(sympy.cos(x)), x)
```


In [131]:

```
d
```

Out[131]:

$$\frac{d}{dx} e^{\cos(x)}$$

In [132]:

```
d.doit()
```

Out[132]:

$$-e^{\cos(x)} \sin(x)$$

Integrals

In [133]:

```
a, b = sympy.symbols("a, b")
x, y = sympy.symbols('x, y')
f = sympy.Function('f')(x)
```

In [134]:

```
sympy.integrate(f)
```

Out[134]:

$$\int f(x) dx$$

In [135]:

```
sympy.integrate(f, (x, a, b))
```

Out[135]:

$$\int_a^b f(x) dx$$

In [136]:

```
sympy.integrate(sympy.sin(x))
```

Out[136]:

$$-\cos(x)$$

In [137]:

```
sympy.integrate(sympy.sin(x), (x, a, b))
```

Out[137]:

$$\cos(a) - \cos(b)$$

In [138]:

```
sympy.integrate(sympy.exp(-x**2), (x, 0, oo))
```

Out[138]:

$$\frac{\sqrt{\pi}}{2}$$

In [139]:

```
a, b, c = sympy.symbols("a, b, c", positive=True)
```

In [140]:

```
sympy.integrate(a * sympy.exp(-((x-b)/c)**2), (x, -oo, oo))
```

Out[140]:

$$\sqrt{\pi}ac$$

In [141]:

```
sympy.integrate(sympy.sin(x * sympy.cos(x)))
```

Out[141]:

$$\int \sin(x \cos(x)) dx$$

In [142]:

```
expr = sympy.sin(x*sympy.exp(y))
```

In [143]:

```
sympy.integrate(expr, x)
```

Out[143]:

$$-e^{-y} \cos(xe^y)$$

In [144]:

```
expr = (x + y)**2
```

In [145]:

```
sympy.integrate(expr, x)
```

Out[145]:

$$\frac{x^3}{3} + x^2y + xy^2$$

In [146]:

```
sympy.integrate(expr, x, y)
```

Out[146]:

$$\frac{x^3 y}{3} + \frac{x^2 y^2}{2} + \frac{x y^3}{3}$$

In [147]:

```
sympy.integrate(expr, (x, 0, 1), (y, 0, 1))
```

Out[147]:

$$\frac{7}{6}$$

Series

In [148]:

```
x = sympy.Symbol("x")
```

In [149]:

```
f = sympy.Function("f")(x)
```

In [150]:

```
sympy.series(f, x)
```

Out[150]:

$$f(0) + x \left. \frac{d}{dx} f(x) \right|_{x=0} + \frac{x^2}{2} \left. \frac{d^2}{dx^2} f(x) \right|_{x=0} + \frac{x^3}{6} \left. \frac{d^3}{dx^3} f(x) \right|_{x=0} + \frac{x^4}{24} \left. \frac{d^4}{dx^4} f(x) \right|_{x=0} + \frac{x^5}{120}$$

In [151]:

```
x0 = sympy.Symbol("{x_0}")
```

In [152]:

```
f.series(x, x0, n=2)
```

Out[152]:

$$f(x_0) + (x - x_0) \left. \frac{d}{d\xi_1} f(\xi_1) \right|_{\xi_1=x_0} + \mathcal{O}((x - x_0)^2; x \rightarrow x_0)$$

In [153]:

```
f.series(x, x0, n=2).removeO()
```

Out[153]:

$$(x - x_0) \left. \frac{d}{d\xi_1} f(\xi_1) \right|_{\xi_1=x_0} + f(x_0)$$

In [154]:

```
sympy.cos(x).series()
```

Out[154]:

$$1 - \frac{x^2}{2} + \frac{x^4}{24} + \mathcal{O}(x^6)$$

In [155]:

```
sympy.sin(x).series()
```

Out[155]:

$$x - \frac{x^3}{6} + \frac{x^5}{120} + \mathcal{O}(x^6)$$

In [156]:

```
sympy.exp(x).series()
```

Out[156]:

$$1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \mathcal{O}(x^6)$$

In [157]:

```
(1/(1+x)).series()
```

Out[157]:

$$1 - x + x^2 - x^3 + x^4 - x^5 + \mathcal{O}(x^6)$$

In [158]:

```
expr = sympy.cos(x) / (1 + sympy.sin(x * y))
```

In [159]:

```
expr.series(x, n=4)
```

Out[159]:

$$1 - xy + x^2 \left(y^2 - \frac{1}{2} \right) + x^3 \left(-\frac{5y^3}{6} + \frac{y}{2} \right) + \mathcal{O}(x^4)$$

In [160]:

```
expr.series(y, n=4)
```

Out[160]:

$$\cos(x) - xy \cos(x) + x^2 y^2 \cos(x) - \frac{5x^3}{6} y^3 \cos(x) + \mathcal{O}(y^4)$$

In [161]:

```
expr.series(y).remove0().series(x).remove0().expand()
```

Out[161]:

$$-\frac{61x^5}{120}y^5 + \frac{5x^5}{12}y^3 - \frac{x^5y}{24} + \frac{2x^4}{3}y^4 - \frac{x^4y^2}{2} + \frac{x^4}{24} - \frac{5x^3}{6}y^3 + \frac{x^3y}{2} + x^2y^2 - \frac{x^2}{2} -$$

Limits

In [162]:

```
sympy.limit(sympy.sin(x) / x, x, 0)
```

Out[162]:

1

In [163]:

```
f = sympy.Function('f')
x, h = sympy.symbols("x, h")
```

In [164]:

```
diff_limit = (f(x + h) - f(x))/h
```

In [165]:

```
sympy.limit(diff_limit.subs(f, sympy.cos), h, 0)
```

Out[165]:

- sin(x)

In [166]:

```
sympy.limit(diff_limit.subs(f, sympy.sin), h, 0)
```

Out[166]:

cos(x)

In [167]:

```
expr = (x**2 - 3*x) / (2*x - 2)
```

In [168]:

```
p = sympy.limit(expr/x, x, oo)
```

In [169]:

```
q = sympy.limit(expr - p*x, x, oo)
```

In [170]:

```
p, q
```

Out[170]:

$$\left(\frac{1}{2}, -1\right)$$

Sums and products

In [171]:

```
n = sympy.symbols("n", integer=True)
```

In [172]:

```
x = sympy.Sum(1/(n**2), (n, 1, oo))
```

In [173]:

```
x
```

Out[173]:

$$\sum_{n=1}^{\infty} \frac{1}{n^2}$$

In [174]:

```
x.doit()
```

Out[174]:

$$\frac{\pi^2}{6}$$

In [175]:

```
x = sympy.Product(n, (n, 1, 7))
```

In [176]:

```
x
```

Out[176]:

$$\prod_{n=1}^7 n$$

In [177]:

```
x.doit()
```

Out[177]:

5040

In [178]:

```
x = sympy.Symbol("x")
```

In [179]:

```
sympy.Sum((x)**n/(sympy.factorial(n)), (n, 1, oo)).doit().simplify()
```

Out[179]:

 $e^x - 1$

Equations

In [180]:

```
x = sympy.symbols("x")
```

In [181]:

```
sympy.solve(x**2 + 2*x - 3)
```

Out[181]:

[-3, 1]

In [182]:

```
a, b, c = sympy.symbols("a, b, c")
```

In [183]:

```
sympy.solve(a * x**2 + b * x + c, x)
```

Out[183]:

$$\left[\frac{1}{2a} \left(-b + \sqrt{-4ac + b^2} \right), -\frac{1}{2a} \left(b + \sqrt{-4ac + b^2} \right) \right]$$

In [184]:

```
sympy.solve(sympy.sin(x) - sympy.cos(x), x)
```

Out[184]:

$$\left[-\frac{3\pi}{4}, \frac{\pi}{4} \right]$$

In [185]:

```
sympy.solve(sympy.exp(x) + 2 * x, x)
```

Out[185]:

$$\left[-\text{LambertW}\left(\frac{1}{2}\right) \right]$$

In [186]:

```
sympy.solve(x**5 - x**2 + 1, x)
```

Out[186]:

[RootOf($x^5 - x^2 + 1, 0$), RootOf($x^5 - x^2 + 1, 1$), RootOf($x^5 - x^2 + 1, 2$), Ro

In [187]:

```
1 #s.solve(s.tan(x) - x, x)
```

Out[187]:

1

In [188]:

```
eq1 = x + 2 * y - 1
eq2 = x - y + 1
```

In [189]:

```
sympy.solve([eq1, eq2], [x, y], dict=True)
```

Out[189]:

$$\left[\left\{ x: -\frac{1}{3}, y: \frac{2}{3} \right\} \right]$$

In [190]:

```
eq1 = x**2 - y
eq2 = y**2 - x
```

In [191]:

```
sols = sympy.solve([eq1, eq2], [x, y], dict=True)
```


In [192]:

sols

Out[192]:

$$\left[\{x: 0, y: 0\}, \{x: 1, y: 1\}, \left\{ x: -\frac{1}{2} + \frac{\sqrt{3}i}{2}, y: -\frac{1}{2} - \frac{\sqrt{3}i}{2} \right\}, \left\{ \right.$$

In [193]:

```
[eq1.subs(sol).simplify() == 0 and eq2.subs(sol).simplify() == 0 for sol in sols]
```

Out[193]:

[True, True, True, True]

Linear algebra

In [194]:

```
sympy.Matrix([1,2])
```

Out[194]:

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

In [195]:

```
sympy.Matrix([[1,2]])
```

Out[195]:

$$\begin{bmatrix} 1 & 2 \end{bmatrix}$$

In [196]:

```
sympy.Matrix([[1, 2], [3, 4]])
```

Out[196]:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

In [197]:

```
sympy.Matrix(3, 4, lambda m,n: 10 * m + n)
```

Out[197]:

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 10 & 11 & 12 & 13 \\ 20 & 21 & 22 & 23 \end{bmatrix}$$

In [198]:

```
a, b, c, d = sympy.symbols("a, b, c, d")
```

In [199]:

```
M = sympy.Matrix([[a, b], [c, d]])
```

In [200]:

```
M
```

Out[200]:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

In [201]:

```
M * M
```

Out[201]:

$$\begin{bmatrix} a^2 + bc & ab + bd \\ ac + cd & bc + d^2 \end{bmatrix}$$

In [202]:

```
x = sympy.Matrix(sympy.symbols("x_1, x_2"))
```

In [203]:

```
M * x
```

Out[203]:

$$\begin{bmatrix} ax_1 + bx_2 \\ cx_1 + dx_2 \end{bmatrix}$$

In [204]:

```
p, q = sympy.symbols("p, q")
```

In [205]:

```
M = sympy.Matrix([[1, p], [q, 1]])
```

In [206]:

```
M
```

Out[206]:

$$\begin{bmatrix} 1 & p \\ q & 1 \end{bmatrix}$$

In [207]:

```
b = sympy.Matrix(sympy.symbols("b_1, b_2"))
```

In [208]:

b

Out[208]:

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

In [209]:

```
x = M.solve(b)
x
```

Out[209]:

$$\begin{bmatrix} b_1 \left(\frac{pq}{-pq+1} + 1 \right) - \frac{b_2 p}{-pq+1} \\ -\frac{b_1 q}{-pq+1} + \frac{b_2}{-pq+1} \end{bmatrix}$$

In [210]:

```
x = M.LUsolve(b)
```

In [211]:

x

Out[211]:

$$\begin{bmatrix} b_1 - \frac{p(-b_1 q + b_2)}{-pq+1} \\ \frac{-b_1 q + b_2}{-pq+1} \end{bmatrix}$$

In [212]:

```
x = M.inv() * b
```

In [213]:

x

Out[213]:

$$\begin{bmatrix} b_1 \left(\frac{pq}{-pq+1} + 1 \right) - \frac{b_2 p}{-pq+1} \\ -\frac{b_1 q}{-pq+1} + \frac{b_2}{-pq+1} \end{bmatrix}$$

In []:

5 - Linear Optimization Using PuLP

Simple Example

Suppose that we have the equation given below and we want to maximize the variable z i.e the objective function.

Objective function

$$z(\max) = 5x_1 + 4x_2$$

Constraints:

- $C_1 = x_1 + x_2 \leq 5$
- $C_2 = 10 * x_1 + 6 * x_2 \leq 45$
- $x_1, x_2 \geq 0$

We can solve the problem graphically but when the number of variables increases the problem becomes very complicated. Here PuLP comes to simplify our optimization problem.

In [8]:

```
# Import PuLP package  
  
import pulp
```

In [9]:

```
z = pulp.LpProblem('Problem', pulp.LpMaximize) # Objective function to be maximized  
  
# Constraints  
  
x1 = pulp.LpVariable('x1', lowBound=0)  
x2 = pulp.LpVariable('x2', lowBound=0)
```

In [10]:

```
# Objective function, we use the += combined to add the objective and constraints  
# to our problem.  
  
z += 5*x1 + 4*x2  
  
z += x1 + x2 <= 5  
z += 10*x1 + 6*x2 <= 45
```

In [11]:

```
# To see our problem in details.  
z
```

Out[11]:

```
Problem:  
MAXIMIZE  
5*x1 + 4*x2 + 0  
SUBJECT TO  
_C1: x1 + x2 <= 5  
  
_C2: 10 x1 + 6 x2 <= 45  
  
VARIABLES  
x1 Continuous  
x2 Continuous
```

In [15]:

```
# Now solving the problem.  
  
z.solve()
```

Out[15]:

1

In [16]:

```
# Now we can view our objective value (z) and the values of our variables, x1 and x2.  
  
for variable in z.variables():  
    print(format(variable.name),variable.varValue)
```

```
x1 3.75  
x2 1.25
```

In [17]:

```
# Printing objective value  
  
print(pulp.value(z.objective))
```

23.75

In []:

Company X Problem

The **Company X** wants to know, How many products the company should make monthly. This company makes, **tables, sofas and chairs**.

- The Company needs to pay \$75000 monthly, this includes, 1540 hours of work (\$48.70 per hour).
- Prices of each product:
 - Tables: \$400 per unit.
 - Sofas: \$750 per unit.
 - Chairs: \$240 per unit.

Variables

- X_1 = Table
- X_2 = Sofa
- X_3 = Chair

Objective function

- $z(max) = (400 - 100)X_1 + (750 - 75 - 175)X_2 + (240 - 40)X_3 - 75000$

This equation is simplified to:

- $z(max) = 300X_1 + 500X_2 + 200X_3 - 75000$

Constraints:

- C_1 (Wood): $10X_1 + 7.5X_2 + 4X_3 \leq 4350$
- C_2 (Cloth): $10X_2 \leq 2500$
- C_3 (Saw): $0.5X_1 + 0.4X_2 + 0.5X_3 \leq 280$
- C_4 (Cut Cloth): $0.4X_2 \leq 140$
- C_5 (Sand): $0.5X_1 + 0.1X_2 + 0.5X_3 \leq 280$
- C_6 (Inky): $0.4X_1 + 0.2X_2 + 0.4X_3 \leq 140$
- C_7 (Assembly): $1X_1 + 1.5X_2 + 0.5X_3 \leq 700$

Demand

- C_8 (Tables): $X_1 \leq 300$
- C_9 (Sofas): $X_2 \leq 180$
- C_{10} (Chairs): $X_3 \leq 400$

$x_1, x_2, x_3 \geq 0$

In [1]:

```
import pulp
```

In [2]:

```
z = pulp.LpProblem("Company X", pulp.LpMaximize)
x1 = pulp.LpVariable("x1", lowBound=0)
x2 = pulp.LpVariable("x2", lowBound=0)
x3 = pulp.LpVariable("x3", lowBound=0)
```

In [3]:

```
# Objective function

z += 300*x1 + 500*x2 + 200*x3 - 75000
```

In [4]:

```
# Constraints

z += 10*x1 + 7.5*x2 + 4*x3 <= 4350
z += 10*x2 <= 2500
z += 0.5*x1 + 0.4*x2 + 0.5*x3 <= 280
z += 0.4*x2 <= 140
z += 0.5*x1 + 0.1*x2 + 0.5*x3 <= 280
z += 0.4*x1 + 0.2*x2 + 0.4*x3 <= 140
z += 1*x1 + 1.5*x2 + 0.5*x3 <= 700

# Demand

z += 1*x1 <= 300
z += 1*x2 <= 180
z += 1*x3 <= 400
```


In [5]:

```
# Problem formulation for clarity
```

```
z
```

Out[5]:

Company X:

MAXIMIZE

$300 \cdot x_1 + 500 \cdot x_2 + 200 \cdot x_3 + -75000$

SUBJECT TO

_C1: $10 \cdot x_1 + 7.5 \cdot x_2 + 4 \cdot x_3 \leq 4350$

_C2: $10 \cdot x_2 \leq 2500$

_C3: $0.5 \cdot x_1 + 0.4 \cdot x_2 + 0.5 \cdot x_3 \leq 280$

_C4: $0.4 \cdot x_2 \leq 140$

_C5: $0.5 \cdot x_1 + 0.1 \cdot x_2 + 0.5 \cdot x_3 \leq 280$

_C6: $0.4 \cdot x_1 + 0.2 \cdot x_2 + 0.4 \cdot x_3 \leq 140$

_C7: $x_1 + 1.5 \cdot x_2 + 0.5 \cdot x_3 \leq 700$

_C8: $x_1 \leq 300$

_C9: $x_2 \leq 180$

_C10: $x_3 \leq 400$

VARIABLES

x1 Continuous

x2 Continuous

x3 Continuous

In [7]:

```
# Status of the prblem
```

```
pulp.LpStatus[z.solve()]
```

Out[7]:

```
'Optimal'
```

In [8]:

```
# Solving for the objective function and the variables
```

```
pulp.value(x1), pulp.value(x2), pulp.value(x3), pulp.value(z.objective)
```

Out[8]:

```
(260.0, 180.0, 0.0, 93000.0)
```

In []:

Power Company

Operations research (often referred to as management science) is simply a scientific approach to decision making that seeks to best design and operate a system, usually under conditions requiring the allocation of scarce resources.

By a system, we mean an organization of interdependent components that work together to accomplish the goal of the system.

For example, Ford Motor Company is a system whose goal consists of maximizing the profit that can be earned by producing quality vehicles.

The term operations research was coined during World War II when British military leaders asked scientists and engineers to analyze several military problems such as the deployment of radar and the management of convoy, bombing, antisubmarine, and mining operations.

The scientific approach to decision making usually involves the use of one or more mathematical models. A mathematical model is a mathematical representation of an actual situation that may be used to make better decisions or simply to understand the actual situation better.

The following example should clarify many of the key terms used to describe mathematical models.

The example below is taken from the book Operations Research APPLICATIONS AND ALGORITHMS by:

Wayne L. Winston FOURTH EDITION INDIANA UNIVERSITY-2004.

The example was solved by using Excel and Lingo/Lindo softwares, here I will be using PuLP.

Powerco has three electric power plants that supply the needs of four cities. Each power plant can supply the following numbers of kilowatt-hours (kwh) of electricity:

Plant_1 = 35 million

Plant_2 = 50 million

Plant_3 = 40 million

The peak power demands in these cities, which occur at the same time (2 P.M.), are as follows (in kwh):

City1 = 45 million

City2 = 20 million

City3 = 30 million

City4 = 30 million

The costs of sending 1 million kwh of electricity from plant to city depend on the distance the electricity must travel.

Formulate an LP to minimize the cost of meeting each city's peakpower demand.

Solution:

To formulate Powerco's problem as an LP, we begin by defining a variable for each decision that Powerco must make.

Because Powerco must determine how much power is sent from each plant to each city, we define $i = 1, 2, 3$, for plants and $j = 1, 2, 3, 4$, for cities.

x_{ij} = number of (million) kwh produced at plant i and sent to city j .

Note: Each city will receive power from all plants. The table below shows the requirements.

TABLE 1:

Supply, Demand requirements, Shipping Costs for Powerco:

From	City 1	City 2	City 3	City 4	Supply(million kwh)
Plant 1	\$8	\$6	\$10	\$9	35
Plant 2	\$9	\$12	\$13	\$7	50
Plant 3	\$14	\$9	\$16	\$5	40
Demand (million KWh)	45	20	30	30	

In terms of these variables, the total cost of supplying the peak power demands to cities 1–4 may be written as:

$$8x_{11} + 6x_{12} + 10x_{13} + 9x_{14} \text{ (Cost of shipping power from plant 1)}$$

$$+ 9x_{21} + 12x_{22} + 13x_{23} + 7x_{24} \text{ (Cost of shipping power from plant 2)}$$

$$+ 14x_{31} + 9x_{32} + 16x_{33} + 5x_{34} \text{ (Cost of shipping power from plant 3)}$$

Powerco faces two types of constraints:

First:

The total power supplied by each plant cannot exceed the plant's capacity.

For example, the total amount of power sent from plant1 to the four cities cannot exceed 35 million kwh.

Each variable with first subscript 1 represents a shipment of power from plant 1, so we may express this restriction by the LP constraint:

$$x_{11} + x_{12} + x_{13} + x_{14} \leq 35$$

In a similar fashion, we can find constraints that reflect plant 2's and plant 3's capacities.

Because power is supplied by the power plants, each is a supply point. Analogously, a constraint that ensures that the total quantity shipped from a plant does not exceed plant capacity is a supply constraint.

The LP formulation of Powerco's problem contains the following three supply constraints:

$$x_{11} + x_{12} + x_{13} + x_{14} \leq 35 \text{ (Plant 1 supply constraint)}$$

$$x_{21} + x_{22} + x_{23} + x_{24} \leq 50 \text{ (Plant 2 supply constraint)}$$

$$x_{31} + x_{32} + x_{33} + x_{34} \leq 40 \text{ (Plant 3 supply constraint)}$$

Second:

We need constraints that ensure that each city will receive sufficient power to meet its peak demand. Each city demands power, so each is a demand point. For example, city 1 must receive at least 45 million kwh. Each variable with second subscript 1 represents a shipment of power to city 1, so we obtain the following constraint:

$$x_{11} + x_{21} + x_{31} \geq 45$$

Similarly, we obtain a constraint for each of cities 2, 3, and 4. A constraint that ensures that a location receives its demand is a demand constraint. Powerco must satisfy the following four demand constraints:

$$x_{11} + x_{21} + x_{31} \geq 45 \text{ (City 1 demand constraint)}$$

$$x_{12} + x_{22} + x_{32} \geq 20 \text{ (City 2 demand constraint)}$$

$$x_{13} + x_{23} + x_{33} \geq 30 \text{ (City 3 demand constraint)}$$

$$x_{14} + x_{24} + x_{34} \geq 30 \text{ (City 4 demand constraint)}$$

Because all the x_{ij} 's must be nonnegative, we add the sign restrictions $x_{ij} \geq 0$ ($i = 1, 2, 3$ for plants and $j = 1, 2, 3, 4$ for cities).

Combining the objective function, supply constraints, demand constraints, and sign restrictions yields the following LP formulation of Powerco's problem:

$$\min z = 8x_{11} + 6x_{12} + 10x_{13} + 9x_{14} + 9x_{21} + 12x_{22} + 13x_{23} + 7x_{24} + 14x_{31} + 9x_{32} + 16x_{33} + 5x_{34}$$

s.t:

$$x_{11} + x_{12} + x_{13} + x_{14} \leq 35 \text{ (Supply constraints)}$$

$$x_{21} + x_{22} + x_{23} + x_{24} \leq 50$$

$$x_{31} + x_{32} + x_{33} + x_{34} \leq 40$$

$$x_{11} + x_{21} + x_{31} + x_{34} \geq 45 \text{ (Demand constraints)}$$

$$x_{12} + x_{22} + x_{32} + x_{34} \geq 20$$

$$x_{13} + x_{23} + x_{33} + x_{34} \geq 30$$

$$x_{14} + x_{24} + x_{34} + x_{34} \geq 30$$

$$x_{ij} \geq 0 \text{ (} i = 1, 2, 3; j = 1, 2, 3, 4\text{)}$$

#

Power Company

In [2]:

```
import pulp
```

Then instantiate a problem class, we'll name it "My LP problem" and we're looking for an optimal minimum so we use LpMinimize.

In [3]:

```
my_lp_problem = pulp.LpProblem("My LP Problem", pulp.LpMinimize)
```

We then model our decision variables using the LpVariable class. In our example the variable x_{ij} had a lower bound of 0.

In [4]:

```
# xij values are continuous variables.

x11 = pulp.LpVariable('x11', lowBound=0, cat='Continuous')
x12 = pulp.LpVariable('x12', lowBound=0, cat='Continuous')
x13 = pulp.LpVariable('x13', lowBound=0, cat='Continuous')
x14 = pulp.LpVariable('x14', lowBound=0, cat='Continuous')
x21 = pulp.LpVariable('x21', lowBound=0, cat='Continuous')
x22 = pulp.LpVariable('x22', lowBound=0, cat='Continuous')
x23 = pulp.LpVariable('x23', lowBound=0, cat='Continuous')
x24 = pulp.LpVariable('x24', lowBound=0, cat='Continuous')
x31 = pulp.LpVariable('x31', lowBound=0, cat='Continuous')
x32 = pulp.LpVariable('x32', lowBound=0, cat='Continuous')
x33 = pulp.LpVariable('x33', lowBound=0, cat='Continuous')
x34 = pulp.LpVariable('x34', lowBound=0, cat='Continuous')
```

The objective function and constraints are added to our model using the += operator, as usual.

The objective function is added first, then the individual constraints.

In [5]:

```
# Objective function

my_lp_problem += 8*x11+6*x12+10*x13+9*x14+9*x21+12*x22+13*x23+7*x24+14*x31+9*x32+
16*x33+5*x34, "Z"

# Constraints

my_lp_problem += x11 + x12 + x13 + x14 <= 35
my_lp_problem += x21 + x22 + x23 + x24 <= 50
my_lp_problem += x31 + x32 + x33 + x34 <= 40
my_lp_problem += x11 + x21 + x31 >= 45
my_lp_problem += x12 + x22 + x32 >= 20
my_lp_problem += x13 + x23 + x33 >= 30
my_lp_problem += x14 + x24 + x34 >= 30
```

We have now constructed our problem and we can have a look at it.

In [6]:

```
my_lp_problem
```

Out[6]:

My LP Problem:

MINIMIZE

$$8x_{11} + 6x_{12} + 10x_{13} + 9x_{14} + 9x_{21} + 12x_{22} + 13x_{23} + 7x_{24} + 14x_{31} + 9x_{32} + 16x_{33} + 5x_{34} + 0$$

SUBJECT TO

C1: $x{11} + x_{12} + x_{13} + x_{14} \leq 35$

C2: $x{21} + x_{22} + x_{23} + x_{24} \leq 50$

C3: $x{31} + x_{32} + x_{33} + x_{34} \leq 40$

C4: $x{11} + x_{21} + x_{31} \geq 45$

C5: $x{12} + x_{22} + x_{32} \geq 20$

C6: $x{13} + x_{23} + x_{33} \geq 30$

C7: $x{14} + x_{24} + x_{34} \geq 30$

VARIABLES

x11 Continuous

x12 Continuous

x13 Continuous

x14 Continuous

x21 Continuous

x22 Continuous

x23 Continuous

x24 Continuous

x31 Continuous

x32 Continuous

x33 Continuous

x34 Continuous

In [7]:

```
# Solving the Poweco problem
```

```
my_lp_problem.solve()
pulp.LpStatus[my_lp_problem.status]
```

Out[7]:

'Optimal'

We have also checked the status of the solver, there are 5 status codes:

- **Not Solved:** Status prior to solving the problem.
- **Optimal:** An optimal solution has been found.
- **Infeasible:** There are no feasible solutions (e.g. if you set the constraints $x \leq 1$ and $x \geq 2$).
- **Unbounded:** The constraints are not bounded, maximising the solution will tend towards infinity (e.g. if the only constraint was $x \geq 3$).
- **Undefined:** The optimal solution may exist but may not have been found.

We can now view our variable values and the minimum value of Z i.e the total cost in \$.

We can use the `varValue` method to retrieve the values of our variables x_{ij} , and the `pulp.value` function to view the minimum value of the objective function.

In [8]:

```
for variable in my_lp_problem.variables():  
    print(format(variable.name),variable.varValue)
```

```
x11 0.0  
x12 10.0  
x13 25.0  
x14 0.0  
x21 45.0  
x22 0.0  
x23 5.0  
x24 0.0  
x31 0.0  
x32 10.0  
x33 0.0  
x34 30.0
```

In [9]:

```
# Printing the total objective cost  
print (pulp.value(my_lp_problem.objective))
```

```
1020.0
```

In []:

"The 'Formulation' of a problem is often more essential than its solution, which may be merely a matter of mathematical or experimental skills"

***Albert Einstein
1876-1955***

Introduction to Optimization

Numerical optimization is one of the most vital and common modern applications of mathematics. It is considered a very important subject and widely used in every engineering field, finance, industry manufacturing etc.

Optimization is the problem of finding numerically the minimums (or maximums or zeros) of a function. In this context, the function is called cost function, or objective function, or energy.

Optimization is classified into Linear Optimization and Non-Linear Optimization. Linear optimization covers 90% of the optimization problems.

1-Linear Programming (LP):

Linear Programming is a powerful modeling tool for optimization. It is a mathematical model whose requirements are linear relationships.

Mathematical optimization or just optimization has two parts:

- a- Objective function (or cost function) and,
- b- constraints which define the set, in which we are looking for the

Softwares to solve optimization problems are available such as Google R, Lindo/Lingo, Gams, Mathcad, Octave and many others.

In solving optimization problems, problem formulation is the most important in writing the objective function and constraints.

We will use Pulp for linear optimization problems and SciPy for both linear and non-linear optimization.

We need to download Pulp first as this library or module is not included in Anaconda.

To download Pulp, open Anaconda Prompt and type after the prompt sign >, with internet connected, the following:

```
>pip install pulp
```

The version is 1.6.10 or later.

What is Pulp:

Pulp is a linear programming framework in Python. The aim of pulp is to allow a practitioner or programmer to express Linear Programming (LP), and Integer Programming (IP) models in python in a way similar to the conventional mathematical notation.

Pulp can be interfaces with other solvers like, CPLEX, COIN, Gurobi, COBYLA etc.

We will use Jupyter in Anaconda to solve optimization problems using Pulp and SciPy.

Optimization of Boiler Turbo Generators

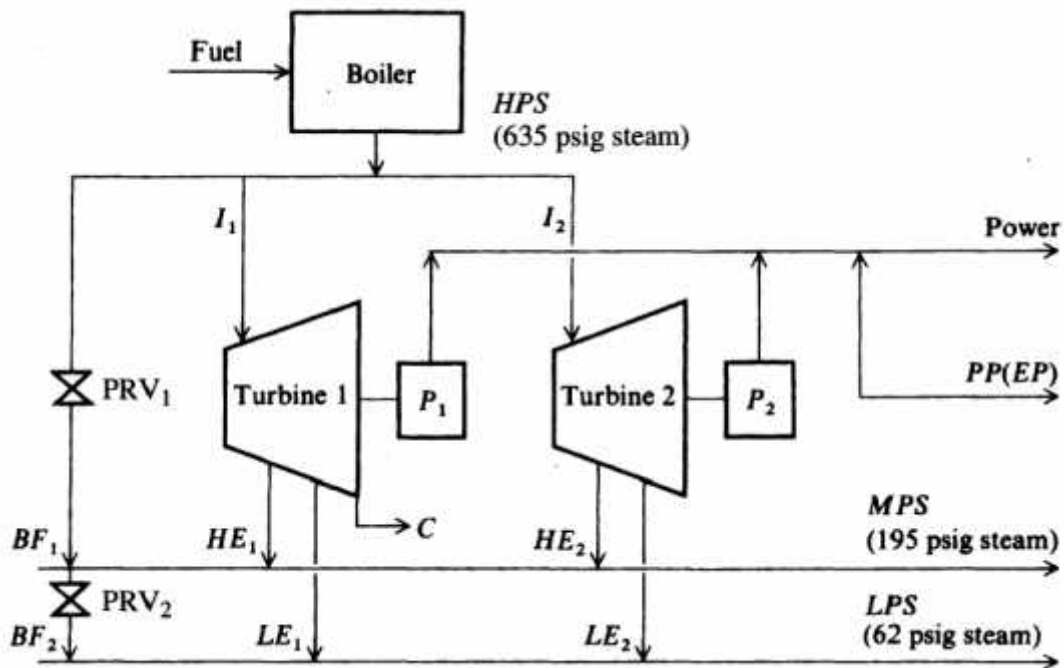
The Boiler/Turbo-Generators Example is taken from the book:
"Optimization Of Chemical Processes-2001".

By:

Thomas F. Edgar, David M. Himmelblau and Leon Lasdon.

The example is not solved but the optimization results are given. Here I will solve the example using Pulp.

The example from 11.4/Chapter-11/Page-435:



Boiler/ Turbo-Generator System Block Diagram

- Key:** I_i = inlet flow rate for turbine i [lb_m/h]
 HE_i = exit flow rate from turbine i to 195 psi header [lb_m/h]
 LE_i = exit flow rate from turbine i to 62 psi header [lb_m/h]
 C = condensate flow rate from turbine 1 [lb_m/h]
 P_i = power generated by turbine i [kW]
 BF_1 = bypass flow rate from 635 psi to 195 psi header [lb_m/h]
 BF_2 = bypass flow rate from 195 psi to 62 psi header [lb_m/h]
 HPS = flow rate through 635 psi header [lb_m/h]
 MPS = flow rate through 195 psi header [lb_m/h]
 LPS = flow rate through 62 psi header [lb_m/h]
 PP = purchased power [kW]
 EP = excess power [kW] (difference of purchased power from base power)
 PRV = pressure-reducing valve

Linear programming is often used in the design and operation of steam systems in the chemical industry. The figure above shows a steam and power system for a small power house fired by wood pulp.

To produce electric power, this system contains two turbo generators whose characteristics are listed in Table 11.4A. Turbine 1 is a double extraction turbine with two intermediate streams leaving at 195 and 62 psi. The final stage produces condensate that is used as boiler feed water. Turbine 2 is a single - extraction turbine with one intermediate stream at 195 psi and an exit stream leaving at 62 psi with no condensate being formed. The first turbine is more efficient due to the energy released from the condensation of steam, but it cannot produce as much power as the second turbine. Excess steam may bypass the turbines to the two levels of steam through pressure-reducing valves. Table 11.4B lists information about the different levels of steam, and Table 11.4C gives the demands on the system.

To meet the electric power demand, electric power may be purchased from another producer with a minimum base of 12,000 kW. If the electric power required to meet the system demand is less than this base, the power that is not used will be charged at a penalty cost. Table 11.4D gives the costs of fuel for the boiler and additional electric power to operate the utility system.

The system shown in the above figure may be modeled as a linear constraints and combined with a linear objective function. The objective is to minimize the operating cost of the system by choice of steam flow rates and power generated or purchased, subject to the demands and restrictions on the system.

TABLE 11.4A
Turbine data

Turbine 1		Turbine 2	
Maximum generative capacity	6,250 kW	Maximum generative capacity	9,000 kW
Minimum load	2,500 kW	Minimum load	3,000 kW
Maximum inlet flow	192,000 lb _m /h	Maximum inlet flow	244,000 lb _m /h
Maximum condensate flow	62,000 lb _m /h	Maximum 62 psi exhaust	142,000 lb _m /h
Maximum internal flow	132,000 lb _m /h	High-pressure extraction at	195 psig
High-pressure extraction at	195 psig	Low-pressure extraction at	62 psig
Low-pressure extraction at	62 psig		

TABLE 11.4B
Steam header data

Header	Pressure (psig)	Temperature (°F)	Enthalpy (Btu/lb_m)
High-pressure steam	635	720	1359.8
Medium-pressure steam	195	130 superheat	1267.8
Low-pressure steam	62	130 superheat	1251.4
Feedwater (condensate)			193.0

TABLE 11.4C
Demands on the system

Resource	Demand
Medium-pressure steam (195 psig)	271,536 lb _m /h
Low-pressure steam (62 psig)	100,623 lb _m /h
Electric power	24,550 kW

Optimal Results

TABLE 11.4D
Energy data

Fuel cost	\$1.68/10 ⁶ Btu
Boiler efficiency	0.75
Steam cost (635 psi)	\$2.24/10 ⁶ Btu = \$2.24 (1359.8 - 193)/10 ⁶ = \$0.002614/lb _m
Purchased electric power	\$0.0239/kWh average
Demand penalty	\$0.009825/kWh
Base-purchased power	12,000 kW

TABLE E11.4E
Optimal solution to steam system LP

Variable	Name	Value	Status
1	I_1	136,329	BASIC
2	I_2	244,000	BOUND
3	HE_1	128,158	BASIC
4	HE_2	143,377	BASIC
5	LE_1	0	ZERO
6	LE_2	100,623	BASIC
7	C	8,170	BASIC
8	BF_1	0	ZERO
9	BF_2	0	ZERO
10	HPS	380,329	BASIC
11	MPS	271,536	BASIC
12	LPS	100,623	BASIC
13	P_1	6,250	BOUND
14	P_2	7,061	BASIC
15	PP	11,239	BASIC
16	EP	761	BASIC

Value of objective function = 1268.75 \$/h

The Jupyter program shows the optimization procedure and as it seen that the results of using Pulp are very close to the values given in the above table E11.4E.

First Import Pulp

In [1]:

```
import pulp
```

Name The Program and Functionality (Minimize)

In [2]:

```
Model = pulp.LpProblem("My LP Problem", pulp.LpMinimize)
```

Name Variables and it's Categories using Cat

In [3]:

```

I1 = pulp.LpVariable('I1',    lowBound=0, cat='Continuous')
I2 = pulp.LpVariable('I2',    lowBound=0, cat='Continuous')
HE1 = pulp.LpVariable('HE1',  lowBound=0, cat='Continuous')
HE2 = pulp.LpVariable('HE2',  lowBound=0, cat='Continuous')
LE1 = pulp.LpVariable('LE1',  lowBound=0, cat='Continuous')
LE2 = pulp.LpVariable('LE2',  lowBound=0, cat='Continuous')
C   = pulp.LpVariable('C',    lowBound=0, cat='Continuous')
BF1 = pulp.LpVariable('BF1',  lowBound=0, cat='Continuous')
BF2 = pulp.LpVariable('BF2',  lowBound=0, cat='Continuous')
HPS = pulp.LpVariable('HPS',  lowBound=0, cat='Continuous')
MPS = pulp.LpVariable('MPS',  lowBound=0, cat='Continuous')
LPS = pulp.LpVariable('LPS',  lowBound=0, cat='Continuous')
P1  = pulp.LpVariable('P1',   lowBound=0, cat='Continuous')
P2  = pulp.LpVariable('P2',   lowBound=0, cat='Continuous')
PP  = pulp.LpVariable('PP',   lowBound=0, cat='Continuous')
EP  = pulp.LpVariable('Ep',   lowBound=0, cat='Continuous')

```

Objective Function to be Minimized

In [4]:

```
Model += 0.00261 * HPS + 0.0239 * PP + 0.00983 * EP, "f"
```

Constraints are as follows:

```

## Turbine-1
## Turbine-2
## Material Balances
## Power Purchased
## Demand
## Energy Balances

```


In [5]:

```

# Turbine1

Model += P1 <= 6250.0
Model += P1 >= 2500.0
Model += HE1 <= 192000.0
Model += C <= 62000.0
Model += I1 - HE1 <= 132000.0

# Turbine2

Model += P2 <= 9000.0
Model += P2 >= 3000.0
Model += I2 <= 244000.0
Model += LE2 <= 142000.0

# Material Balances

Model += HPS - I1 - I2 - BF1 == 0
Model += I1 + I2 + BF1 - C - MPS - LPS == 0
Model += I1 - HE1 - LE1 - C == 0
Model += I2 - HE2 - LE2 == 0
Model += HE1 + HE2 + BF1 - BF2 - MPS == 0
Model += LE1 + LE2 + BF2 - LPS == 0

# Power Purchased

Model += EP + PP >= 12000.0

# Demand

Model += MPS >= 271536.0
Model += LPS >= 100623.0
Model += P1 + P2 + PP >= 24550.0

# Energy Balances

Model += 1359.8 * I1 - 1267.8 * HE1 - 1251.4 * LE1 - 192 * C - 3413 * P1 == 0
Model += 1359.8 * I2 - 1267.8 * HE2 - 1251.4 * LE2 - 3413 * P2 == 0

```

Print Out The Model For Clarity

In [6]:

Model

Out[6]:

My LP Problem:

MINIMIZE

 $0.00983*Ep + 0.00261*HPS + 0.0239*PP + 0.0$

SUBJECT TO

 $_C1: P1 \leq 6250$ $_C2: P1 \geq 2500$ $_C3: HE1 \leq 192000$ $_C4: C \leq 62000$ $_C5: -HE1 + I1 \leq 132000$ $_C6: P2 \leq 9000$ $_C7: P2 \geq 3000$ $_C8: I2 \leq 244000$ $_C9: LE2 \leq 142000$ $_C10: -BF1 + HPS - I1 - I2 = 0$ $_C11: BF1 - C + I1 + I2 - LPS - MPS = 0$ $_C12: -C - HE1 + I1 - LE1 = 0$ $_C13: -HE2 + I2 - LE2 = 0$ $_C14: BF1 - BF2 + HE1 + HE2 - MPS = 0$ $_C15: BF2 + LE1 + LE2 - LPS = 0$ $_C16: Ep + PP \geq 12000$ $_C17: MPS \geq 271536$ $_C18: LPS \geq 100623$ $_C19: P1 + P2 + PP \geq 24550$ $_C20: -192 C - 1267.8 HE1 + 1359.8 I1 - 1251.4 LE1 - 3413 P1 = 0$ $_C21: -1267.8 HE2 + 1359.8 I2 - 1251.4 LE2 - 3413 P2 = 0$

VARIABLES

BF1 Continuous

BF2 Continuous

C Continuous

Ep Continuous

HE1 Continuous

HE2 Continuous

HPS Continuous

I1 Continuous

```
I2 Continuous  
LE1 Continuous  
LE2 Continuous  
LPS Continuous  
MPS Continuous  
P1 Continuous  
P2 Continuous  
PP Continuous
```

Solve The Model and Check Model Status

In [7]:

```
Model.solve()  
pulp.LpStatus[Model.status]
```

Out[7]:

```
'Optimal'
```

Print Model Results (Variables)

In [8]:

```
for variable in Model.variables():  
    print(format(variable.name),variable.varValue)
```

```
BF1 0.0  
BF2 0.0  
C 8169.7397  
Ep 760.71409  
HE1 128159.0  
HE2 143377.0  
HPS 380328.74  
I1 136328.74  
I2 244000.0  
LE1 0.0  
LE2 100623.0  
LPS 100623.0  
MPS 271536.0  
P1 6250.0  
P2 7060.7141  
PP 11239.286
```

Print Total Cost Of The Objective function

In [9]:

```
print (pulp.value(Model.objective))
```

```
1268.7547663046998
```

In []:

6 - Linear and Non-Linear Optimization using SciPy

SciPy (Scientific Python) optimize provides functions for minimizing or maximizing objective functions, possibly subject to constraints.

It includes solvers for nonlinear problems (with support for both local and global optimization algorithms), linear programming, constrained and nonlinear least-squares, root finding and curve fitting.

Here, simple linear and nonlinear examples will be given as an introduction to show how to use SciPy in solving an optimization problem. The method can be modified easily for other applications.

Example-1 / Linear Programming Problem:

Three products can be manufactured either manually, semi-automatic or automatic.

The manual production:

Require, 1 minute of qualified work, 40 minutes of non-qualified work and three minutes of assemblage.

The semi-automatic method:

Require, 4, 30 and 2 minutes.

Fully automatic method:

Require, 8, 20 and 4 minutes.

A startup time of 4500 minutes of qualified work, 36000 minutes of nonqualified work and 2700 minutes for assembly.

The production costs are \$70, \$80 and \$85 for the manual, semi-automatic and automatic methods. The number of units to be produced is 999.

It is required to minimize the cost function.

Solution:

The variables are the number of units x_1 , x_2 and x_3 to be built using each method (manual, semi-automatic and automatic). The cost to be minimized is the production cost and is given by:

$$\text{Production Cost} = 70*x_1 + 80*x_2 + 85*x_3.$$

The objective function is, therefore, given by:

$$\text{Minimize } f(x) = 70*x_1 + 80*x_2 + 85*x_3$$

Subject to:

$$x_1 + x_2 + x_3 = 999$$

$$x_1 + 4x_2 + 8x_3 \leq 4500$$

$$40*x_1 + 30*x_2 + 20*x_3 \leq 36000$$

$$3*x_1 + 2*x_2 + 4*x_3 \leq 2700$$

$$X \geq 0$$

SciPy solution is given in Jupyter Notebook program.

Example-2/Nonlinear Programming Problem:

Pulp is dedicated to solve linear optimization problems, SciPy can be used to solve both linear and nonlinear problems.

Suppose that we have the following problem.

Minimize: $x_1*x_4*(x_1+x_2+x_3) + x_3$

Subject to:

$$x_1*x_2*x_3*x_4 \geq 25$$

$$x_1^2 + x_2^2 + x_3^2 + x_4^2 = 40$$

$$1 \leq x_1, x_2, x_3, x_4 \leq 5$$

Initial guess is required, say $x_0 = (1,5,5,1)$

SciPy solution is given in Jupyter Notebook program.

##

Linear and Non-Linear Examples

In [1]:

```
# Example-1 / Linear Programming.
```

```
import numpy as np
from scipy.optimize import linprog
from numpy.linalg import solve
A_eq = np.array([[1,1,1]])
b_eq = np.array([999])
A_ub = np.array([
[1, 4, 8],
[40,30,20],
[3,2,4]])
b_ub = np.array([4500, 36000,2700])
c = np.array([70, 80, 85])
res = linprog(c, A_eq=A_eq, b_eq=b_eq, A_ub=A_ub, b_ub=b_ub,
bounds=(0, None))
print('Optimal value:', res.fun, '\nX:', res.x)
```

Optimal value: 73725.0

X: [636. 330. 33.]

In [5]:

```
# Example-2 / Non-Linear Programming.

import numpy as np
from scipy.optimize import minimize
def objective(x):
    return x[0]*x[3]*(x[0]+x[1]+x[2])+x[2]
def constraint1(x):
    return x[0]*x[1]*x[2]*x[3]-25.0
def constraint2(x):
    sum_eq = 40.0
    for i in range(4):
        sum_eq = sum_eq - x[i]**2
    return sum_eq
# initial guesses
n = 4
x0 = np.zeros(n)
x0[0] = 1.0
x0[1] = 5.0
x0[2] = 5.0
x0[3] = 1.0
# show initial objective
print('Initial Objective: ' + str(objective(x0)))
# optimize
b = (1.0,5.0)
bnds = (b, b, b, b)
con1 = {'type': 'ineq', 'fun': constraint1}
con2 = {'type': 'eq', 'fun': constraint2}
cons = ([con1,con2])
solution = minimize(objective,x0,method='SLSQP',\
                    bounds=bnds,constraints=cons)
x = solution.x
# show final objective
print('Final Objective: ' + str(objective(x)))
# print solution
print('Solution')
print('x1 = ' + str(x[0]))
print('x2 = ' + str(x[1]))
print('x3 = ' + str(x[2]))
print('x4 = ' + str(x[3]))
```

Initial Objective: 16.0

Final Objective: 49.49466354824061

Solution

x1 = 5.0

x2 = 2.449466356243835

x3 = 2.041221961170239

x4 = 1.0

Appendix

My work sheets with the PTC Community-USA/Mathcad **Division 2010-2017**

- Mathcad Scriptable Control to calculate input impedance (3/5 Stars)
- Optimization using Mathcad Prime 2.0 (5/5 Stars)
- Ask Mathcad Prime 2.0 (5/5 Stars)
- The Good Old Radio Design using Mathcad (5/5 Stars)
- Powerful Electromagnet Design using Mathcad (5/5 Stars)
- GSM Traffic Calculations with Mathcad Prime 2.0 (-)
- Penumatic Conveying with Mathcad Prime 2.0 (-)
- High Power Coaxial Cable with Mathcad Prime 2.0 (5/5 Stars)
- AC Circuit Performance with Mathcad 15 M10 (2/5 Stars)
- Fuel Day Tank for Engine Generator Set with Mathcad Prime 2.0 (-)
- Binary Genetic Algorithm With Mathcad Prime 2.0 (5/5 Stars)
- Factors A,B and C For Power Transformer/Mathcad Prime 4.0-Express (4/5 Stars)
- Solving Engineering PSO Optimization Problem with Mathcad Prim 2.0 (5/5 Stars)
- My First Oscillator 45-Years Ago with Mathcad Prime 2.0 (5/5 Stars)

References

- Electrical schematic drawing with Python - Collin's Site-2014.
- Solving circuit equations with SchemDraw.
- Stavelly_python_ebook-2014.
- Math with Python Documentation-2016.
- Pulp Documentation-2019.
- Scipy Lecture Notes-2019
- CVXOPT Documentaton-2018.
- COBYLA Documentaton-2018.
- Optimization with Pulp
- PyDotPlus Documentaton-Sep.2017.
- <https://graphviz.gitlab.io/documentation>.
- The Decision tree Algorithm- Full version-Statinfer.
- Numerical Methods in Engineering with Python-Jaan Kiusalaas 2005.
- Python tutorial using GUI with Tkinter.
- Rapid GUI programming with Python and Qt-Mark Summerfield-2008.
- Anaconda (Python distribution) Wikipedia.
- Companies using Python-Full Stack Python_files.

Python - A Brief History

Python, was originally conceptualized by Guido van Rossum in the late 1980s as a member of the National Research Institute of Mathematics and Computer Science. Initially, it was designed as a response to the ABC programming language that was also fore grounded in the Netherlands. Among the main features of Python compared to the ABC language was that Python had exception handling and was targeted for the Amoeba operating system (go Python!).

Python is not named after the snake. It's named after the British TV show Monty Python.

Of course, Python, like other languages, has gone through a number of versions. Python 0.9.0 was first released in 1991.

In 2000, Python 2.0 was released. This version of was more of an open-source project from members of the National Research Institute of Mathematics and Computer Science. This version of Python included list comprehensions, a full garbage collector, and it supported Unicode.

Python 3.0 was the next version and was released in December of 2008 (the latest version of Python is 3.6.4).

The above is obtained from: www.medium.com / by **John Wolfe-Medium**