Hailemariam A. Tekile
April 04, 2024

Solving Multiobjective Optimization Problem with Python

Multi-Objective Optimization and Decision-Making with pymoo: Balancing Objectives, Finding Solutions

https://pymoo.org

Exercise 1: Find solutions in pymoo using Compromise Programming and Pseudo-weights, visualize the results.

Min f1(x) = x1^2+x2^2+x3^2

Max f2(x) = -(x1-1)^2 - (x2-1)^2 - (x3-1)^2

Subject to:

```
g1 = x1 + x2 + x3 - 1 <= 0
g2 = -3x1 + x2 + x3 - 4 >=0
-10 <= x1 <= 10
-10 <= x2 <= 10
-10 <= x3 <= 10
```

Source: https://www.udemy.com/course/multi-objective-optimization-with-python-bootcamp-a-z/?couponCode=KEEPLEARNING

Steps:

1. Install pymoo and import all the required libraries accordingly.
2. Develop a class and define a problem.
3. Initialize NSGA-II algorithm using below parameters: pop_size = 50, n_offsprings = 10, cross_over = SBX(prob=0.9, eta=20), mutation = PM(eta=25).
4. Use n_eval = 100 termination criteria.
5. Check out your objectives vector and visualize it.
6. Normalize the objective vector using ideal point and nadir point.
7. Use Compromise Programming and Pseudo-weights methods to find the Optimum Point. Note! Imagine that the first objective is less important thatìn the other for us.
8. Visualise the results of each method and compare.

```
!pip install pymoo #install pymoo on colab
```

```
Collecting pymoo
  Downloading pymoo-0.6.1.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.1/4.1 MB 19.7 MB/s eta 0:00:0
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: scipy>=1.1 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: matplotlib>=3 in /usr/local/lib/python3.10/dis
Requirement already satisfied: autograd>=1.4 in /usr/local/lib/python3.10/dis
Collecting cma==3.2.2 (from pymoo)
  Downloading cma-3.2.2-py2.py3-none-any.whl (249 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 249.1/249.1 kB 20.4 MB/s eta 0:
Collecting alive-progress (from pymoo)
  Downloading alive_progress-3.1.5-py3-none-any.whl (75 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 76.0/76.0 kB 9.2 MB/s eta 0:00:
Collecting dill (from pymoo)
  Downloading dill-0.3.8-py3-none-any.whl (116 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 14.2 MB/s eta 0:
Collecting Deprecated (from pymoo)
  Downloading Deprecated-1.2.14-py2.py3-none-any.whl (9.6 kB)
Requirement already satisfied: future>=0.15.2 in /usr/local/lib/python3.10/di
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3
Collecting about-time==4.2.1 (from alive-progress->pymoo)
  Downloading about_time-4.2.1-py3-none-any.whl (13 kB)
Collecting grapheme==0.6.0 (from alive-progress->pymoo)
  Downloading grapheme-0.6.0.tar.gz (207 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.3/207.3 kB 24.8 MB/s eta 0:
  Preparing metadata (setup.py) ... done
Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.10/di
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-pac
Building wheels for collected packages: grapheme
  Building wheel for grapheme (setup.py) ... done
  Created wheel for grapheme: filename=grapheme-0.6.0-py3-none-any.whl size=2
  Stored in directory: /root/.cache/pip/wheels/01/e1/49/37e6bde9886439057450c
Successfully built grapheme
Installing collected packages: grapheme, dill, Deprecated, cma, about-time, a
Successfully installed Deprecated-1.2.14 about-time-4.2.1 alive-progress-3.1.
```

Class Development

```
import numpy as np
from pymoo.core.problem import ElementwiseProblem

class MyProblem(ElementwiseProblem):
    def __init__(self):
      super().__init__(n_var = 3,
                       n_obj = 2,
                       n_ieq_constr = 2,
                       xl = np.array([-10,-10,-10]),
                       xu = np.array([10,10,10]))
    def _evaluate(self,x,out,*args,**kwargs):
      f1 = (x[0]**2+x[1]**2+x[2]**2)
      f2 = (x[0]-1)**2 + (x[1]-1)**2 + (x[2]-1)**2 #changed sign to min

      g1 = x[0] + x[1] + x[2] - 1
      g2 = -3*x[0] + x[1] + x[2] - 4 #changed sign to <= due to pymoo nature

      out["F"] = [f1,f2]
      out["G"] = [g1,g2]

problem = MyProblem()
```

Initializing the algorithm

```
from pymoo.algorithms.moo.nsga2 import NSGA2. #Nondominated Searching GA
from pymoo.operators.sampling.rnd import FloatRandomSampling
from pymoo.operators.crossover.sbx import SBX #sbs - simulated binary crossover
from pymoo.operators.mutation.pm import PM  #pm - polynomial mutation

algorithm = NSGA2(
    pop_size = 50,
    n_offsprings = 10,
    sampling = FloatRandomSampling(),
    crossover = SBX(prob=0.9,eta=20), #prob - probability of crossover performed,
    mutation = PM(eta = 25), #controls the mutation rate
    eliminate_duplicates = True #duplicated solution is eliminated as we need a u
)
```

Termination criteria

```
from pymoo.termination import get_termination
termination = get_termination("n_gen",100) #number of max generation
```

Optimization process

```
from pymoo.optimize import minimize

res = minimize(problem,
               algorithm,
               termination,
               seed = 7, #to ensure the reproductivity of the result
               save_history = True,
               verbose = True) #progress information
```

```
        41 |      450 |       9 |  0.000000E+00 |  0.000000E+00 |  0.0106135853
        42 |      460 |      10 |  0.000000E+00 |  0.000000E+00 |  0.0031413755
        43 |      470 |      11 |  0.000000E+00 |  0.000000E+00 |  0.0073681145
        44 |      480 |      12 |  0.000000E+00 |  0.000000E+00 |  0.0005501714
        45 |      490 |       9 |  0.000000E+00 |  0.000000E+00 |  0.0583601551
        46 |      500 |       9 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00
        47 |      510 |      10 |  0.000000E+00 |  0.000000E+00 |  0.0852358178
        48 |      520 |      10 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00
```

```
    49 |      530 |      11 |  0.000000E+00 |  0.000000E+00 |  0.0034953149
    50 |      540 |      12 |  0.000000E+00 |  0.000000E+00 |  0.0095402962
    51 |      550 |      11 |  0.000000E+00 |  0.000000E+00 |  0.0108366689
    52 |      560 |      12 |  0.000000E+00 |  0.000000E+00 |  0.1790616998
    53 |      570 |      13 |  0.000000E+00 |  0.000000E+00 |  0.0000469777
    54 |      580 |      13 |  0.000000E+00 |  0.000000E+00 |  0.0000469777
    55 |      590 |      13 |  0.000000E+00 |  0.000000E+00 |  0.0000469777
    56 |      600 |      13 |  0.000000E+00 |  0.000000E+00 |  0.0000469777
    57 |      610 |      13 |  0.000000E+00 |  0.000000E+00 |  0.0000469777
    58 |      620 |      13 |  0.000000E+00 |  0.000000E+00 |  0.0000469777
    59 |      630 |      14 |  0.000000E+00 |  0.000000E+00 |  0.0022760854
    60 |      640 |      13 |  0.000000E+00 |  0.000000E+00 |  0.0074982977
    61 |      650 |      14 |  0.000000E+00 |  0.000000E+00 |  0.0172668472
    62 |      660 |      14 |  0.000000E+00 |  0.000000E+00 |  0.0038975676
    63 |      670 |      15 |  0.000000E+00 |  0.000000E+00 |  0.0171126562
    64 |      680 |      15 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00
    65 |      690 |      15 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00
    66 |      700 |      16 |  0.000000E+00 |  0.000000E+00 |  0.0005470753
    67 |      710 |      17 |  0.000000E+00 |  0.000000E+00 |  0.0012427643
    68 |      720 |      17 |  0.000000E+00 |  0.000000E+00 |  0.0017126401
    69 |      730 |      17 |  0.000000E+00 |  0.000000E+00 |  0.0017126401
    70 |      740 |      19 |  0.000000E+00 |  0.000000E+00 |  0.0028246345
    71 |      750 |      20 |  0.000000E+00 |  0.000000E+00 |  0.0005448801
    72 |      760 |      21 |  0.000000E+00 |  0.000000E+00 |  0.0013459296
    73 |      770 |      24 |  0.000000E+00 |  0.000000E+00 |  0.0057052447
    74 |      780 |      26 |  0.000000E+00 |  0.000000E+00 |  0.0024406486
    75 |      790 |      26 |  0.000000E+00 |  0.000000E+00 |  0.0018992394
    76 |      800 |      28 |  0.000000E+00 |  0.000000E+00 |  0.0017930954
    77 |      810 |      29 |  0.000000E+00 |  0.000000E+00 |  0.0102145992
    78 |      820 |      29 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00
    79 |      830 |      29 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00
    80 |      840 |      29 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00
    81 |      850 |      29 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00
    82 |      860 |      30 |  0.000000E+00 |  0.000000E+00 |  0.0025351970
    83 |      870 |      31 |  0.000000E+00 |  0.000000E+00 |  0.0002107872
    84 |      880 |      33 |  0.000000E+00 |  0.000000E+00 |  0.0026295236
    85 |      890 |      35 |  0.000000E+00 |  0.000000E+00 |  0.0003580824
    86 |      900 |      35 |  0.000000E+00 |  0.000000E+00 |  0.0015729377
    87 |      910 |      36 |  0.000000E+00 |  0.000000E+00 |  0.0016607241
    88 |      920 |      37 |  0.000000E+00 |  0.000000E+00 |  0.0023363304
    89 |      930 |      39 |  0.000000E+00 |  0.000000E+00 |  0.0440810259
    90 |      940 |      39 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00
    91 |      950 |      39 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00
    92 |      960 |      34 |  0.000000E+00 |  0.000000E+00 |  0.0091217739
    93 |      970 |      34 |  0.000000E+00 |  0.000000E+00 |  0.000000E+00
    94 |      980 |      34 |  0.000000E+00 |  0.000000E+00 |  0.0011715814
    95 |      990 |      34 |  0.000000E+00 |  0.000000E+00 |  0.0011715814
    96 |     1000 |      36 |  0.000000E+00 |  0.000000E+00 |  0.0017897348
    97 |     1010 |      36 |  0.000000E+00 |  0.000000E+00 |  0.0017897348
    98 |     1020 |      36 |  0.000000E+00 |  0.000000E+00 |  0.0017897348
```

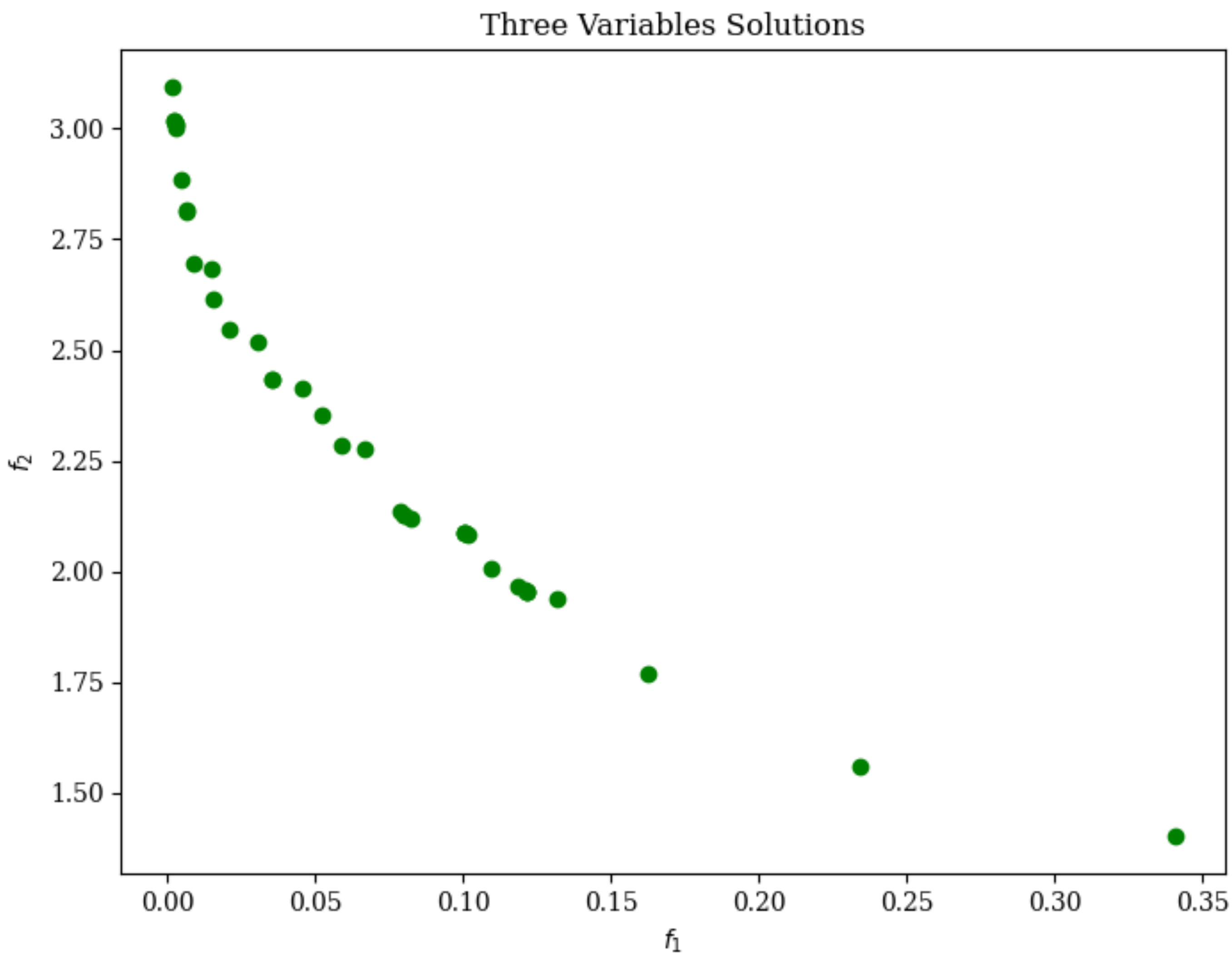
Results and Visualization

```
X = res.X
F = res.F

print(F) #print the two obj values f1 and f2
```

```
[[3.51516867e-02 2.43437673e+00]
 [2.07594373e-02 2.54764779e+00]
 [8.23006535e-02 2.11889252e+00]
 [3.06145410e-02 2.51682219e+00]
 [1.00683007e-01 2.08676401e+00]
 [8.90044126e-03 2.69551304e+00]
 [1.21610814e-01 1.95374097e+00]
 [1.00578037e-01 2.08759879e+00]
 [5.86838536e-02 2.28399100e+00]
 [7.99449369e-02 2.12817832e+00]
 [1.18426072e-01 1.96854103e+00]
 [1.00598368e-01 2.08745824e+00]
 [1.52238753e-02 2.68376208e+00]
 [1.21774081e-01 1.95258499e+00]
 [1.09545740e-01 2.00540250e+00]
 [2.29839994e-03 3.01689203e+00]
 [1.21066449e-01 1.95957984e+00]
 [5.21198953e-02 2.35204324e+00]
 [1.21612196e-01 1.95373115e+00]
 [8.01068083e-02 2.12703215e+00]
 [3.51506692e-02 2.43438461e+00]
 [2.34074871e-01 1.56002463e+00]
 [6.37545740e-03 2.81677905e+00]
 [7.86649677e-02 2.13734174e+00]
 [2.23136405e-03 3.01881850e+00]
 [2.91884497e-03 3.00294974e+00]
 [1.62568275e-01 1.76745803e+00]
 [2.57906613e-03 3.00939169e+00]
 [1.31899743e-01 1.93722098e+00]
 [6.55071114e-03 2.81183303e+00]
 [1.42626272e-03 3.09267842e+00]
 [1.52748799e-02 2.61558587e+00]
 [3.41171509e-01 1.40170473e+00]
 [4.45029679e-03 2.88345190e+00]
 [2.61057710e-03 3.00804030e+00]
 [6.66939895e-02 2.27859136e+00]
 [1.01513979e-01 2.08451111e+00]
 [4.58035755e-02 2.41218507e+00]]
```

```
#visualization
from pymoo.visualization.scatter import Scatter

plot = Scatter(title = "Three Variables Solutions")
plot.add(F, color = "green")
plot.show()
```

```
<pymoo.visualization.scatter.Scatter at 0x7f555f893610>
```



Normailization

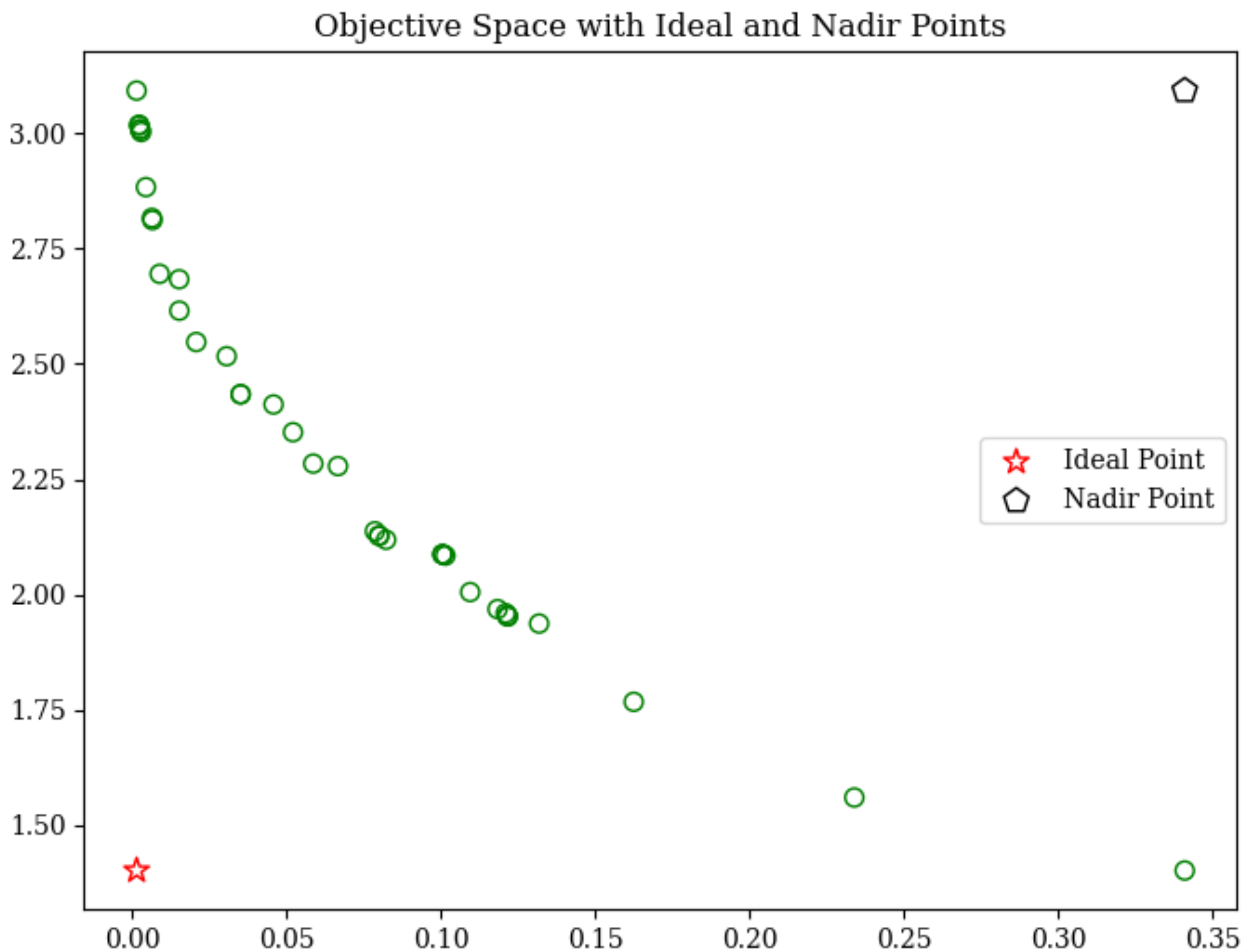
```
ideal_point = F.min(axis=0) #axis=0 means in each row
nadir_point = F.max(axis=0) #nadir point is the worst objective values of the sol

ideal_point
```

```
array([0.00142626, 1.40170473])
```

```
nadir_point
```

```
array([0.34117151, 3.09267842])
```

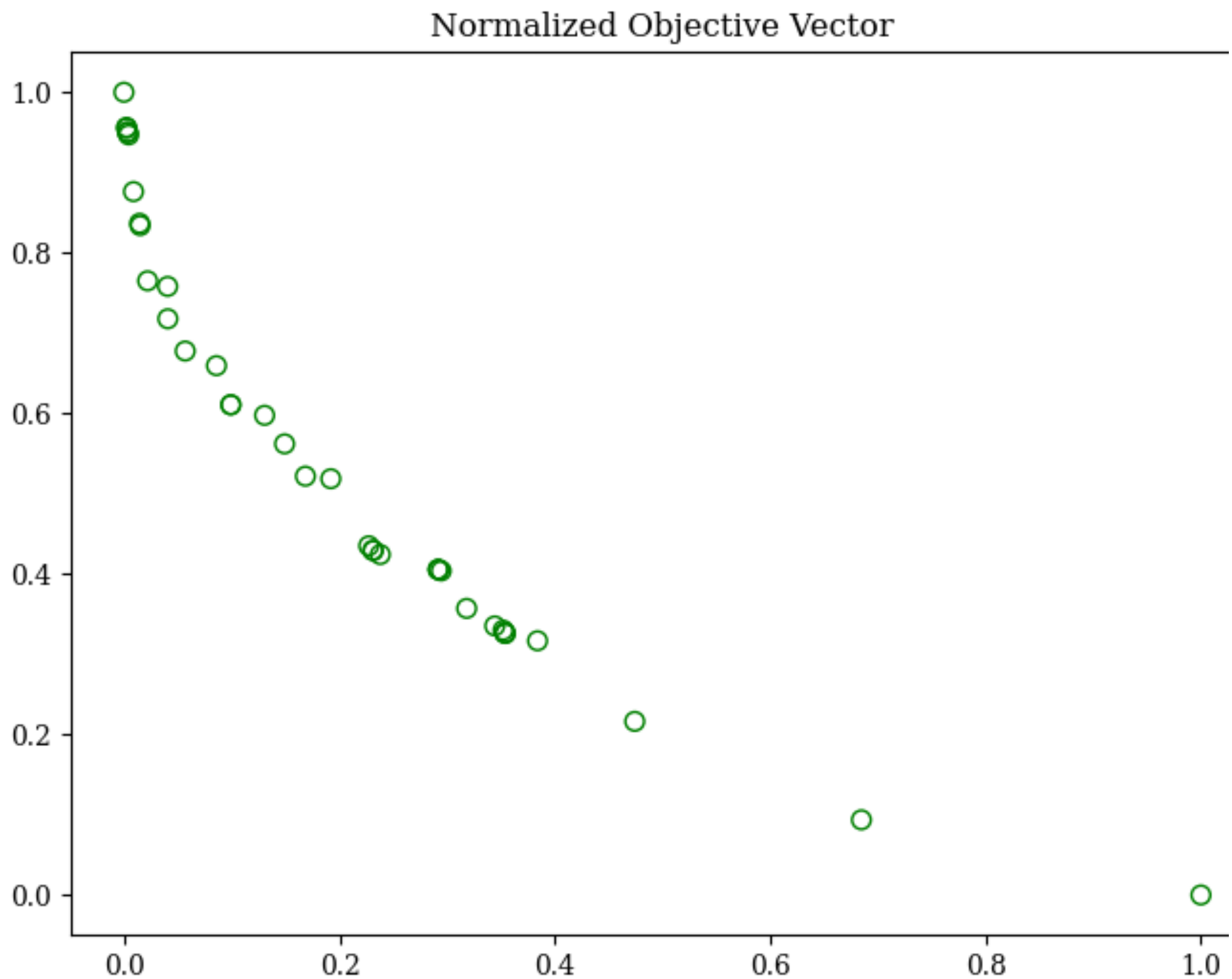
```
import matplotlib.pyplot as plt

plt.figure(figsize=(8,6))
plt.scatter(F[:,0],F[:,1],s=50,facecolor='none',edgecolors = 'green') #scatter pl
plt.scatter(ideal_point[0],ideal_point[1],facecolor='none',edgecolors = 'red',mar
plt.scatter(nadir_point[0],nadir_point[1],facecolor='none',edgecolors = 'black',m
plt.title('Objective Space with Ideal and Nadir Points')
plt.legend()
plt.show()
```



```
#normalized vector of objectives F
nF = (F-ideal_point) / (nadir_point - ideal_point) #normalized F

fl = nF.min(axis=0) #Lower f on each row
fu = nF.max(axis=0) #Upper f on each row

print(f"Scale f1: [{fl[0]},{fu[0]}]")
print(f"Scale f2: [{fl[1]},{fu[1]}]")
```

```
Scale f1: [0.0,1.0]
Scale f2: [0.0,1.0]
```

```
plt.figure(figsize=(8,6))
plt.scatter(nF[:,0],nF[:,1],s=50,facecolor='none',edgecolors = 'green')
plt.title('Normalized Objective Vector')
plt.show()
```

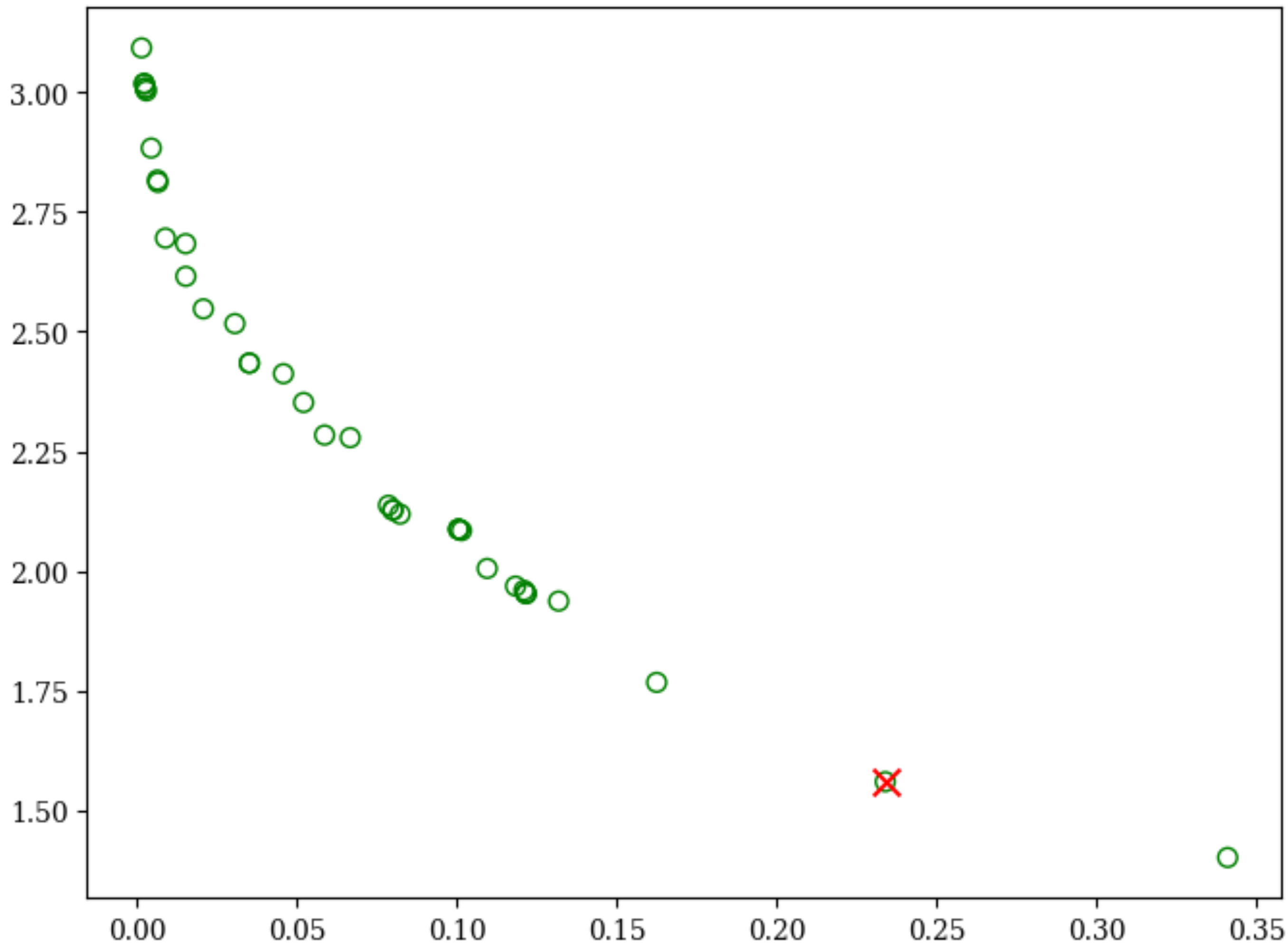


Decision making using Compromise Programming

```
from pymoo.decomposition.asf import ASF #ASF - Augmented Scalarization Function
decomp = ASF()

Weights = np.array([0.2,0.8]) #objective f1 is 20% important and f2 is 80% import

#optimum index, where the optimal value is located
opt_index = decomp.do(nF,1/Weights).argmin() #returns the index of the minimum va

print(f"Best ASF: \n Opt_index = {opt_index} \n F = {F[opt_index]}")
```

```
Best ASF:
 Opt_index = 21
 F = [0.23407487 1.56002463]
```

```
#display the optimal solution in a red color
plt.figure(figsize=(8,6))
plt.scatter(F[:,0],F[:,1],s=50,facecolor='none',edgecolors = 'green')
plt.scatter(F[opt_index,0],F[opt_index,1],marker = 'x',color='red',s=100)
plt.show()
```



```
# Values of THE decision variables x1, x2, x3
print(X)
```

```
[[ 0.14181568  0.11441062  0.04416117]
 [ 0.07202145  0.11417775  0.05035662]
 [ 0.21715542  0.12357996  0.14096869]
 [ 0.01005064  0.11856646  0.12827907]
 [ 0.26868596  0.11193526  0.12633828]
 [ 0.07202145  0.03431563  0.05035662]
 [ 0.21715542  0.12342837  0.24335114]
 [ 0.26868596  0.11146538  0.12633828]
 [ 0.07202145  0.11417775  0.20114722]
 [ 0.21226892  0.12342357  0.14019082]
 [ 0.21715542  0.11862835  0.23915876]
 [ 0.26868596  0.11146538  0.12641872]
 [ 0.01840404  0.11856646  0.0287604 ]
 [ 0.21715542  0.12408799  0.24335114]
 [ 0.23674665  0.11417775  0.20114722]
 [-0.03245749  0.03412542 -0.00896474]
 [ 0.22535888  0.11798132  0.2374031 ]
 [ 0.17728696  0.03267414  0.14007723]
 [ 0.21715542  0.12343396  0.24335114]
 [ 0.21226892  0.12407759  0.14019082]
 [ 0.14181568  0.11440618  0.04416117]
 [ 0.27437776  0.29722479  0.26542257]
 [ 0.07202145  0.03294038 -0.01016363]
 [ 0.21226892  0.1186256   0.1397671 ]
 [-0.03245749  0.03312867 -0.00896474]
 [-0.03293146  0.04188075 -0.00896474]
 [ 0.21715542  0.23704856  0.24335114]
 [-0.03245749  0.03801593 -0.00896474]
 [ 0.26868596  0.11093634  0.21771708]
 [ 0.07202145  0.03550102 -0.01016363]
 [-0.03724326 -0.00276549 -0.00561733]
 [ 0.07202145  0.03294038  0.09488267]
 [ 0.28760208  0.2274765   0.45465481]
 [ 0.06657114 -0.00276549 -0.00330645]
 [-0.03298751  0.03820345 -0.00793081]
 [ 0.07202145  0.23049616  0.0915337 ]
 [ 0.2702279   0.11193526  0.12633828]
 [ 0.1889862   0.03294038  0.09488267]]
```

```
# Optimal solution based on the optimum index 21
X_Optimum = X[21,:]

print(X_Optimum)
```

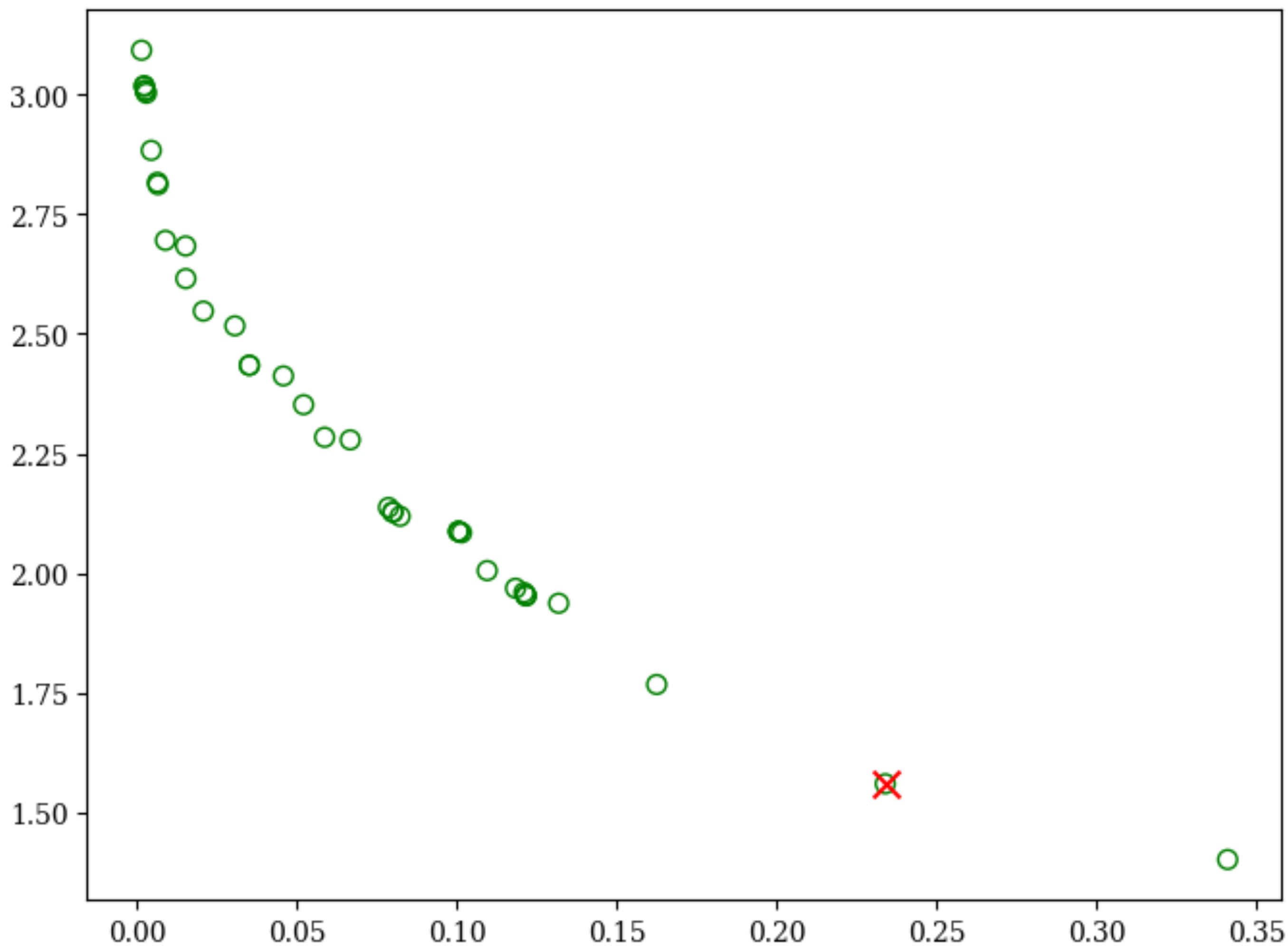
```
[0.27437776 0.29722479 0.26542257]
```

Decision Making Using Pseudo-Weights

```
from pymoo.mcdm.pseudo_weights import PseudoWeights

Weights = np.array([0.2,0.8])

Opt_Index2 = PseudoWeights(Weights).do(nF)

print(f"Best Pseudo Weights: \n Opt_Index2 = {Opt_Index2} \n F = {F[Opt_Index2]}"
```

```
Best Pseudo Weights:
 Opt_Index2 = 21
 F = [0.23407487 1.56002463]
```

```
plt.figure(figsize=(8,6))
plt.scatter(F[:,0],F[:,1],s=50,facecolor='none',edgecolors = 'green')
plt.scatter(F[Opt_Index2,0],F[Opt_Index2,1],marker = 'x',color='red',s=100)
plt.show()
```



Comparison

No solution change in both methods regarding the above example.