"P11" Frentes de Pareto

NESTOR

Mayo 2022

1. Objetivo

El objetivo se trata de en optimización multicriterio, a un mismo conjunto de variables ocupa asignarse valores de tal forma que se optimizen dos o más funciones objetivo, que pueden contradecir una a otra — una mejora en una puede corresponder en una empeora en otra. Además hay que respetar potenciales restricciones, si es que haya.

Para estudiar este problema, vamos a primero implementar un generador de polinomios aleatorios. Estos polinomios los utilizaremos como funciones objetivo. Vamos a permitir solamente una variable por término y un término por grado por variable [1].

2. Desarrollo

De acuerdo con el desarrollo en el código de la codificación implementado por E. Schaeffer y todas las instrucciones se encuentran en el repositorio de N. Rodríguez en GitHub.

Para comenzar se implementa una solución aleatoria para polinomios que se generan al azar, con esto se crea la solución de polinomios y evaluación. A continuación se ejecuta el siguiente código:

```
import numpy as np
from scipy import stats
import pandas as pd
from itertools import compress
from random import randint, random
import matplotlib.pyplot as plt

def poli(maxdeg, varcount, termcount):
    f = []
    for t in range(termcount):
        var = randint(0, varcount - 1)
        deg = randint(1, maxdeg)
        f.append({'var': var, 'coef': random(), 'deg': deg})
    return pd.DataFrame(f)

def evaluate(pol, var):
    return sum([t.coef * var[pol.at[i, 'var']]**t.deg for i, t in pol.iterrows()])


def domin_by(target, challenger):
    if np.any(challenger < target):
        return False
    return np.any(challenger > target)
```

```
vc = 4
md = 3
tc = 5
#k = 2

replicas= 30
CV=[]
for k in range(2,6):
    rep=[]
    for r in range(replicas):
        obj = [poli(md, vc, tc) for i in range(k)]
        minim = np.random.rand(k) > 0.8
        n = 100
        sol = np.random.rand(n, vc)
        val = np.zeros((n, k))
        for i in range(n):
            for j in range(k):
                val[i, j] = evaluate(obj[j], sol[i])
        sign = [1 + -2 * m for m in minim]
        dom = []
        for i in range(n):
            d = [domin_by(sign * val[i], sign * val[j]) for j in range(n)]
            dom.append(sum(d))
        frente = val[[d == 0 for d in dom], :]
        rep.append((len(frente)*100)/n)
    CV.append(rep)
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize=(4, 10))
plt.ylabel('Porcentaje (%) soluci n de pareto')
parts = ax.violinplot(CV, showmeans=False, showmedians=False, showextrema=False)
for p in parts['bodies']:
    p.set_facecolor('grey')
    p.set_edgecolor('green')
    p.set_alpha(1)
c='blue'
plt.boxplot([CV[0], CV[1], CV[2], CV[3]],
            capprops=dict(color=c),
            whiskerprops=dict(color=c),
            flierprops=dict(color=c, markeredgecolor=c),
            medianprops=dict(color='lime'),widths=(0.10, 0.10, 0.10, 0.10))
plt.subplots_adjust(bottom = 0.3, wspace = 0.05)
plt.xticks([1,2,3,4], ['2','3','4','5'])
plt.xlabel('Funci n')
plt.savefig('p11p_violin.png', bbox_inches = 'tight')
plt.close()

result = stats.kruskal(CV[0], CV[1], CV[2], CV[3])
print(result)
```

3. Resultados

El rango de las soluciones de porcentaje no dominada se empieza a distribuir....

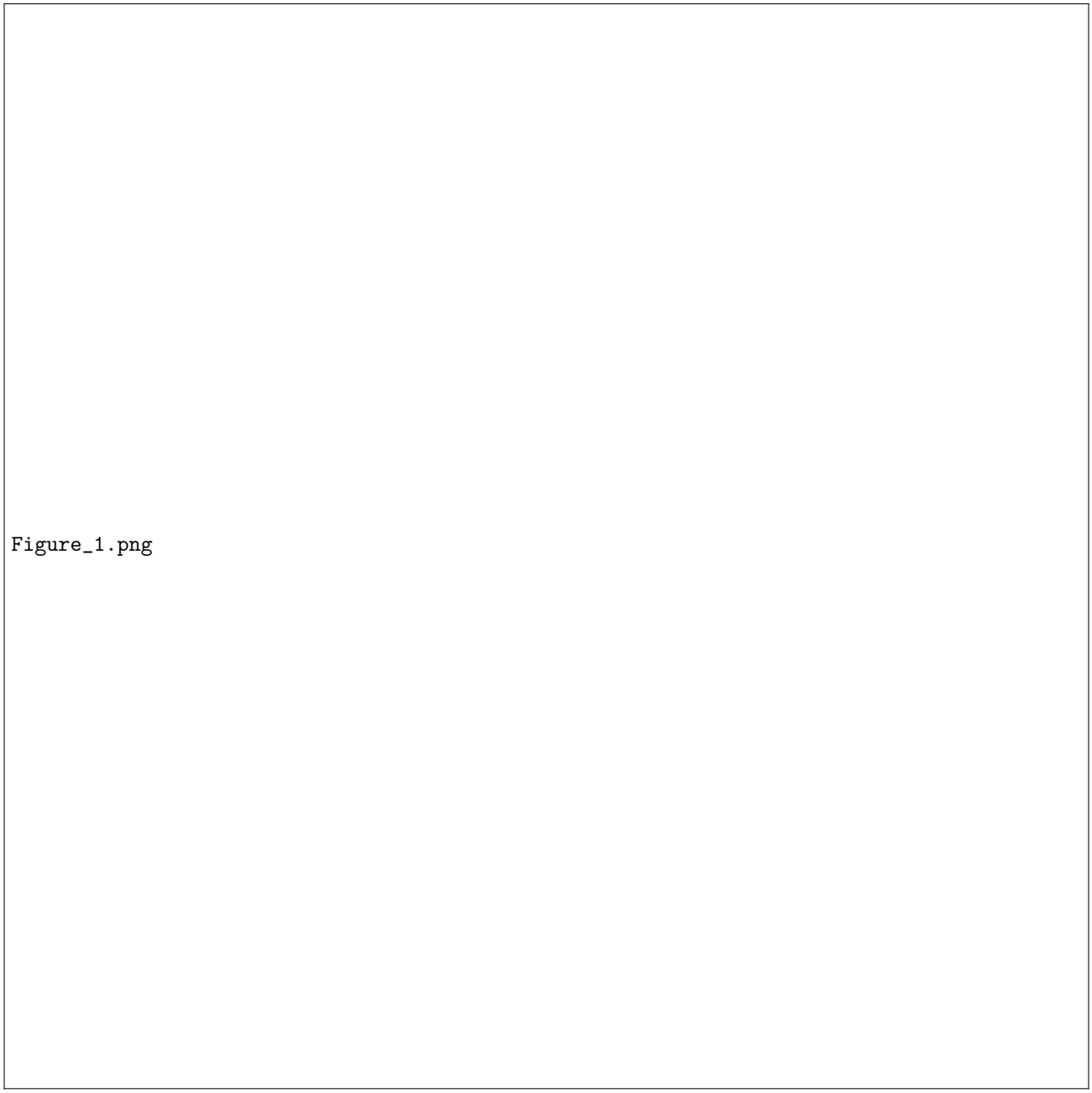

Figura 1: Diagrama violín de distribucion de porcentaje

4. Conclusiones

Rara vez se logran a coincidir y a veces son bastante opuestos, se ocupa una definición para qué en sí es una buena solución [1].

Referencias

[1] E. Schaeffer. Genetic algorithm. *Repositorio, GitHub*, 2022. URL `https://github.com/satuelisa/Simulation/blob/master/ParetoFronts/poligen.py`.