In [1]:
```
!pip install deap
```

```
Collecting deap
  Downloading deap-1.4.2-cp310-cp310-win_amd64.whl (109 kB)
Requirement already satisfied: numpy in c:\users\smain\appdata\local\programs\pyt
hon\python310\lib\site-packages (from deap) (1.26.4)
Installing collected packages: deap
Successfully installed deap-1.4.2
```

```
WARNING: You are using pip version 21.2.3; however, version 25.0.1 is available.
You should consider upgrading via the 'C:\Users\smain\AppData\Local\Programs\Pyth
on\Python310\python.exe -m pip install --upgrade pip' command.
```

In [2]:
```
import random

from deap import base, creator, tools

# Evaluation function
def eval_func(individual):
    target_sum = 45
    return len(individual) - abs(sum(individual) - target_sum),

# Create the toolbox with the right parameters
def create_toolbox(num_bits):
    creator.create("FitnessMax", base.Fitness, weights=(1.0,))
    creator.create("Individual", list, fitness=creator.FitnessMax)

    # Initialize the toolbox
    toolbox = base.Toolbox()

    # Generate attributes
    toolbox.register("attr_bool", random.randint, 0, 1)

    # Initialize structures
    toolbox.register("individual", tools.initRepeat, creator.Individual,
        toolbox.attr_bool, num_bits)

    # Define the population to be a list of individuals
    toolbox.register("population", tools.initRepeat, list, toolbox.individual)

    # Register the evaluation operator
    toolbox.register("evaluate", eval_func)

    # Register the crossover operator
    toolbox.register("mate", tools.cxTwoPoint)

    # Register a mutation operator
    toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)

    # Operator for selecting individuals for breeding
    toolbox.register("select", tools.selTournament, tournsize=3)

    return toolbox

if __name__ == "__main__":
    # Define the number of bits
    num_bits = 75

    # Create a toolbox using the above parameter
```

```
    toolbox = create_toolbox(num_bits)

    # Seed the random number generator
    random.seed(7)

    # Create an initial population of 500 individuals
    population = toolbox.population(n=500)

    # Define probabilities of crossing and mutating
    probab_crossing, probab_mutating  = 0.5, 0.2

    # Define the number of generations
    num_generations = 60

    print('\nStarting the evolution process')

    # Evaluate the entire population
    fitnesses = list(map(toolbox.evaluate, population))
    for ind, fit in zip(population, fitnesses):
        ind.fitness.values = fit

    print('\nEvaluated', len(population), 'individuals')

    # Iterate through generations
    for g in range(num_generations):
        print("\n===== Generation", g)

        # Select the next generation individuals
        offspring = toolbox.select(population, len(population))

        # Clone the selected individuals
        offspring = list(map(toolbox.clone, offspring))

        # Apply crossover and mutation on the offspring
        for child1, child2 in zip(offspring[::2], offspring[1::2]):
            # Cross two individuals
            if random.random() < probab_crossing:
                toolbox.mate(child1, child2)

                # "Forget" the fitness values of the children
                del child1.fitness.values
                del child2.fitness.values

        # Apply mutation
        for mutant in offspring:
            # Mutate an individual
            if random.random() < probab_mutating:
                toolbox.mutate(mutant)
                del mutant.fitness.values

        # Evaluate the individuals with an invalid fitness
        invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
        fitnesses = map(toolbox.evaluate, invalid_ind)
        for ind, fit in zip(invalid_ind, fitnesses):
            ind.fitness.values = fit

        print('Evaluated', len(invalid_ind), 'individuals')

        # The population is entirely replaced by the offspring
        population[:] = offspring
```

```
    # Gather all the fitnesses in one list and print the stats
    fits = [ind.fitness.values[0] for ind in population]

    length = len(population)
    mean = sum(fits) / length
    sum2 = sum(x*x for x in fits)
    std = abs(sum2 / length - mean**2)**0.5

    print('Min =', min(fits), ', Max =', max(fits))
    print('Average =', round(mean, 2), ', Standard deviation =',
            round(std, 2))

print("\n==== End of evolution")

best_ind = tools.selBest(population, 1)[0]
print('\nBest individual:\n', best_ind)
print('\nNumber of ones:', sum(best_ind))
```

```
Starting the evolution process

Evaluated 500 individuals

===== Generation 0
Evaluated 297 individuals
Min = 58.0 , Max = 75.0
Average = 70.43 , Standard deviation = 2.91

===== Generation 1
Evaluated 303 individuals
Min = 63.0 , Max = 75.0
Average = 72.44 , Standard deviation = 2.16

===== Generation 2
Evaluated 310 individuals
Min = 65.0 , Max = 75.0
Average = 73.31 , Standard deviation = 1.6

===== Generation 3
Evaluated 273 individuals
Min = 67.0 , Max = 75.0
Average = 73.76 , Standard deviation = 1.41

===== Generation 4
Evaluated 309 individuals
Min = 68.0 , Max = 75.0
Average = 73.87 , Standard deviation = 1.35

===== Generation 5
Evaluated 312 individuals
Min = 68.0 , Max = 75.0
Average = 73.83 , Standard deviation = 1.36

===== Generation 6
Evaluated 308 individuals
Min = 67.0 , Max = 75.0
Average = 73.76 , Standard deviation = 1.5

===== Generation 7
Evaluated 314 individuals
Min = 67.0 , Max = 75.0
Average = 73.85 , Standard deviation = 1.39

===== Generation 8
Evaluated 309 individuals
Min = 66.0 , Max = 75.0
Average = 73.84 , Standard deviation = 1.48

===== Generation 9
Evaluated 288 individuals
Min = 68.0 , Max = 75.0
Average = 73.85 , Standard deviation = 1.47

===== Generation 10
Evaluated 306 individuals
Min = 68.0 , Max = 75.0
Average = 73.83 , Standard deviation = 1.42

===== Generation 11
```

```
Evaluated 301 individuals
Min = 66.0 , Max = 75.0
Average = 73.83 , Standard deviation = 1.45

===== Generation 12
Evaluated 312 individuals
Min = 68.0 , Max = 75.0
Average = 73.87 , Standard deviation = 1.35

===== Generation 13
Evaluated 310 individuals
Min = 66.0 , Max = 75.0
Average = 73.7 , Standard deviation = 1.58

===== Generation 14
Evaluated 289 individuals
Min = 68.0 , Max = 75.0
Average = 73.77 , Standard deviation = 1.57

===== Generation 15
Evaluated 312 individuals
Min = 68.0 , Max = 75.0
Average = 73.92 , Standard deviation = 1.37

===== Generation 16
Evaluated 281 individuals
Min = 66.0 , Max = 75.0
Average = 73.92 , Standard deviation = 1.5

===== Generation 17
Evaluated 304 individuals
Min = 67.0 , Max = 75.0
Average = 74.03 , Standard deviation = 1.34

===== Generation 18
Evaluated 314 individuals
Min = 67.0 , Max = 75.0
Average = 73.96 , Standard deviation = 1.38

===== Generation 19
Evaluated 317 individuals
Min = 66.0 , Max = 75.0
Average = 73.71 , Standard deviation = 1.55

===== Generation 20
Evaluated 297 individuals
Min = 68.0 , Max = 75.0
Average = 73.77 , Standard deviation = 1.57

===== Generation 21
Evaluated 290 individuals
Min = 68.0 , Max = 75.0
Average = 73.99 , Standard deviation = 1.31

===== Generation 22
Evaluated 299 individuals
Min = 66.0 , Max = 75.0
Average = 73.89 , Standard deviation = 1.49

===== Generation 23
```

```
Evaluated 292 individuals
Min = 68.0 , Max = 75.0
Average = 73.92 , Standard deviation = 1.34

===== Generation 24
Evaluated 312 individuals
Min = 68.0 , Max = 75.0
Average = 73.83 , Standard deviation = 1.31

===== Generation 25
Evaluated 294 individuals
Min = 68.0 , Max = 75.0
Average = 73.89 , Standard deviation = 1.31

===== Generation 26
Evaluated 307 individuals
Min = 68.0 , Max = 75.0
Average = 73.93 , Standard deviation = 1.31

===== Generation 27
Evaluated 303 individuals
Min = 67.0 , Max = 75.0
Average = 73.86 , Standard deviation = 1.43

===== Generation 28
Evaluated 293 individuals
Min = 69.0 , Max = 75.0
Average = 74.02 , Standard deviation = 1.27

===== Generation 29
Evaluated 315 individuals
Min = 67.0 , Max = 75.0
Average = 73.9 , Standard deviation = 1.37

===== Generation 30
Evaluated 280 individuals
Min = 69.0 , Max = 75.0
Average = 74.07 , Standard deviation = 1.19

===== Generation 31
Evaluated 277 individuals
Min = 67.0 , Max = 75.0
Average = 74.07 , Standard deviation = 1.3

===== Generation 32
Evaluated 289 individuals
Min = 69.0 , Max = 75.0
Average = 74.0 , Standard deviation = 1.34

===== Generation 33
Evaluated 297 individuals
Min = 68.0 , Max = 75.0
Average = 73.9 , Standard deviation = 1.37

===== Generation 34
Evaluated 288 individuals
Min = 69.0 , Max = 75.0
Average = 74.02 , Standard deviation = 1.3

===== Generation 35
```

```
Evaluated 310 individuals
Min = 68.0 , Max = 75.0
Average = 74.04 , Standard deviation = 1.28

===== Generation 36
Evaluated 298 individuals
Min = 67.0 , Max = 75.0
Average = 74.04 , Standard deviation = 1.34

===== Generation 37
Evaluated 283 individuals
Min = 67.0 , Max = 75.0
Average = 74.13 , Standard deviation = 1.25

===== Generation 38
Evaluated 291 individuals
Min = 69.0 , Max = 75.0
Average = 74.12 , Standard deviation = 1.1

===== Generation 39
Evaluated 306 individuals
Min = 69.0 , Max = 75.0
Average = 74.01 , Standard deviation = 1.24

===== Generation 40
Evaluated 294 individuals
Min = 67.0 , Max = 75.0
Average = 73.97 , Standard deviation = 1.31

===== Generation 41
Evaluated 296 individuals
Min = 68.0 , Max = 75.0
Average = 74.08 , Standard deviation = 1.22

===== Generation 42
Evaluated 305 individuals
Min = 69.0 , Max = 75.0
Average = 74.09 , Standard deviation = 1.12

===== Generation 43
Evaluated 300 individuals
Min = 69.0 , Max = 75.0
Average = 73.99 , Standard deviation = 1.22

===== Generation 44
Evaluated 257 individuals
Min = 68.0 , Max = 75.0
Average = 74.15 , Standard deviation = 1.18

===== Generation 45
Evaluated 311 individuals
Min = 68.0 , Max = 75.0
Average = 74.01 , Standard deviation = 1.26

===== Generation 46
Evaluated 322 individuals
Min = 67.0 , Max = 75.0
Average = 74.08 , Standard deviation = 1.15

===== Generation 47
```

```
Evaluated 293 individuals
Min = 68.0 , Max = 75.0
Average = 74.11 , Standard deviation = 1.19

===== Generation 48
Evaluated 333 individuals
Min = 68.0 , Max = 75.0
Average = 73.97 , Standard deviation = 1.23

===== Generation 49
Evaluated 288 individuals
Min = 70.0 , Max = 75.0
Average = 74.03 , Standard deviation = 1.16

===== Generation 50
Evaluated 286 individuals
Min = 66.0 , Max = 75.0
Average = 74.04 , Standard deviation = 1.31

===== Generation 51
Evaluated 309 individuals
Min = 70.0 , Max = 75.0
Average = 74.08 , Standard deviation = 1.21

===== Generation 52
Evaluated 305 individuals
Min = 68.0 , Max = 75.0
Average = 74.1 , Standard deviation = 1.23

===== Generation 53
Evaluated 305 individuals
Min = 67.0 , Max = 75.0
Average = 74.03 , Standard deviation = 1.33

===== Generation 54
Evaluated 290 individuals
Min = 70.0 , Max = 75.0
Average = 74.15 , Standard deviation = 1.11

===== Generation 55
Evaluated 302 individuals
Min = 69.0 , Max = 75.0
Average = 74.07 , Standard deviation = 1.15

===== Generation 56
Evaluated 306 individuals
Min = 68.0 , Max = 75.0
Average = 73.96 , Standard deviation = 1.32

===== Generation 57
Evaluated 306 individuals
Min = 68.0 , Max = 75.0
Average = 74.02 , Standard deviation = 1.27

===== Generation 58
Evaluated 276 individuals
Min = 69.0 , Max = 75.0
Average = 74.15 , Standard deviation = 1.18

===== Generation 59
```

```
Evaluated 288 individuals
Min = 69.0 , Max = 75.0
Average = 74.12 , Standard deviation = 1.24

==== End of evolution

Best individual:
 [1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1,
0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0,
0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1]

Number of ones: 45
```

Le bloc de code précédentimplémente un algorithme génétique utilisant la bibliothèque DEAP pour optimiser une population d'individus représentés sous forme binaire.

L'objectif est de trouver une combinaison de bits dont la somme se rapproche de 45. Pour cela, une fonction d'évaluation mesure la différence entre la somme des bits de l'individu et la valeur cible.

L'algorithme commence par initialiser une population de 500 individus de 75 bits chacun. À chaque génération, il effectue une sélection des meilleurs individus en utilisant un tournoi. Ensuite, il applique des croisements et mutations avec des probabilités définies (50% pour le croisement et 20% pour la mutation).

Après chaque génération, les individus sont réévalués, et les statistiques sur la population sont affichées : valeur minimale, maximale, moyenne et écart-type des scores.

Après 60 générations, l'évolution s'arrête, et l'algorithme affiche le meilleur individu trouvé ainsi que le nombre de bits actifs (1) dans celui-ci.

Ce code est un exemple d'optimisation par algorithme génétique, illustrant comment l'évolution naturelle peut être simulée pour résoudre un problème numérique.

In [3]:

```python
import numpy as np
import matplotlib.pyplot as plt
from deap import algorithms, base, benchmarks, \
        cma, creator, tools

# Function to create a toolbox
def create_toolbox(strategy):
    creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
    creator.create("Individual", list, fitness=creator.FitnessMin)

    toolbox = base.Toolbox()
    toolbox.register("evaluate", benchmarks.rastrigin)

    # Seeed the random number generator
    np.random.seed(7)

    toolbox.register("generate", strategy.generate, creator.Individual)
    toolbox.register("update", strategy.update)

    return toolbox
```

```python
if __name__ == "__main__":
    # Problem size
    num_individuals = 10
    num_generations = 125

    # Create a strategy using CMA-ES algorithm
    strategy = cma.Strategy(centroid=[5.0]*num_individuals, sigma=5.0,
            lambda_=20*num_individuals)

    # Create toolbox based on the above strategy
    toolbox = create_toolbox(strategy)

    # Create hall of fame object
    hall_of_fame = tools.HallOfFame(1)

    # Register the relevant stats
    stats = tools.Statistics(lambda x: x.fitness.values)
    stats.register("avg", np.mean)
    stats.register("std", np.std)
    stats.register("min", np.min)
    stats.register("max", np.max)

    logbook = tools.Logbook()
    logbook.header = "gen", "evals", "std", "min", "avg", "max"

    # Objects that will compile the data
    sigma = np.ndarray((num_generations, 1))
    axis_ratio = np.ndarray((num_generations, 1))
    diagD = np.ndarray((num_generations, num_individuals))
    fbest = np.ndarray((num_generations,1))
    best = np.ndarray((num_generations, num_individuals))
    std = np.ndarray((num_generations, num_individuals))

    for gen in range(num_generations):
        # Generate a new population
        population = toolbox.generate()

        # Evaluate the individuals
        fitnesses = toolbox.map(toolbox.evaluate, population)
        for ind, fit in zip(population, fitnesses):
            ind.fitness.values = fit

        # Update the strategy with the evaluated individuals
        toolbox.update(population)

        # Update the hall of fame and the statistics with the
        # currently evaluated population
        hall_of_fame.update(population)
        record = stats.compile(population)
        logbook.record(evals=len(population), gen=gen, **record)

        print(logbook.stream)

        # Save more data along the evolution
        sigma[gen] = strategy.sigma
        axis_ratio[gen] = max(strategy.diagD)**2/min(strategy.diagD)**2
        diagD[gen, :num_individuals] = strategy.diagD**2
        fbest[gen] = hall_of_fame[0].fitness.values
        best[gen, :num_individuals] = hall_of_fame[0]
        std[gen, :num_individuals] = np.std(population, axis=0)
```

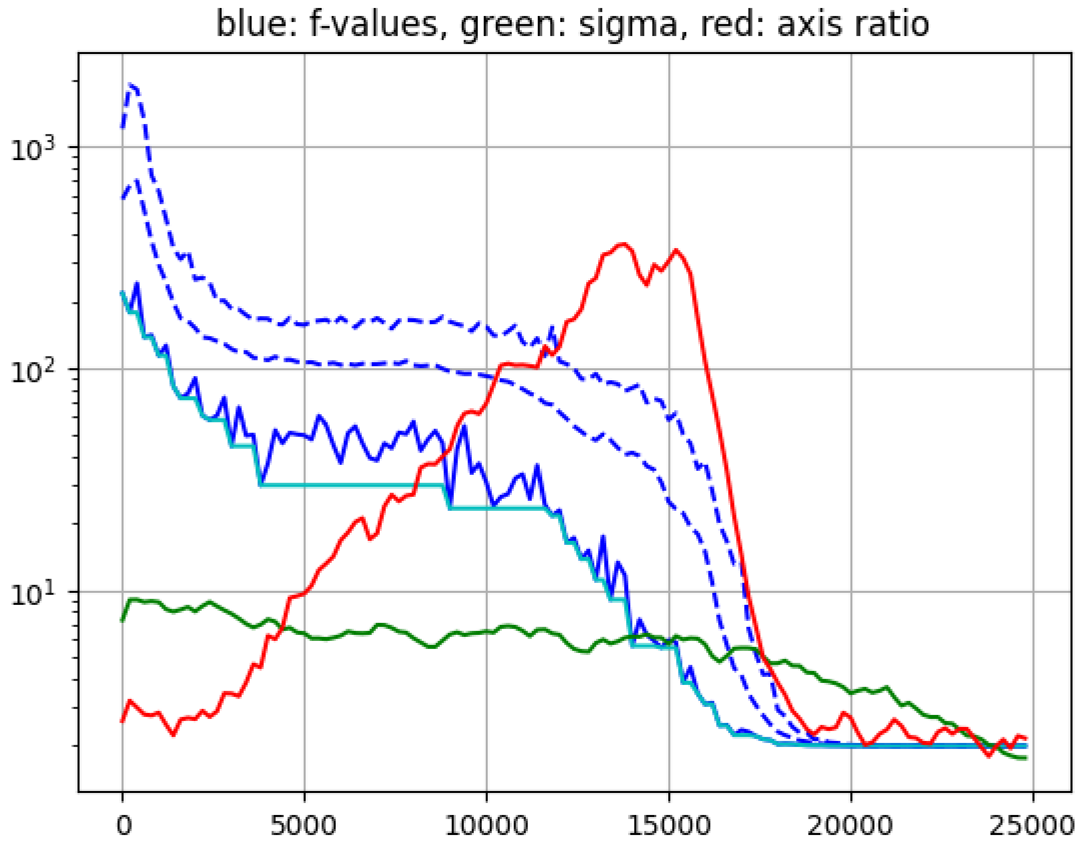
```python
    # The x-axis will be the number of evaluations
    x = list(range(0, strategy.lambda_ * num_generations, strategy.lambda_))
    avg, max_, min_ = logbook.select("avg", "max", "min")
    plt.figure()
    plt.semilogy(x, avg, "--b")
    plt.semilogy(x, max_, "--b")
    plt.semilogy(x, min_, "-b")
    plt.semilogy(x, fbest, "-c")
    plt.semilogy(x, sigma, "-g")
    plt.semilogy(x, axis_ratio, "-r")
    plt.grid(True)
    plt.title("blue: f-values, green: sigma, red: axis ratio")

    plt.figure()
    plt.plot(x, best)
    plt.grid(True)
    plt.title("Object Variables")

    plt.figure()
    plt.semilogy(x, diagD)
    plt.grid(True)
    plt.title("Scaling (All Main Axes)")

    plt.figure()
    plt.semilogy(x, std)
    plt.grid(True)
    plt.title("Standard Deviations in All Coordinates")

    plt.show()
```

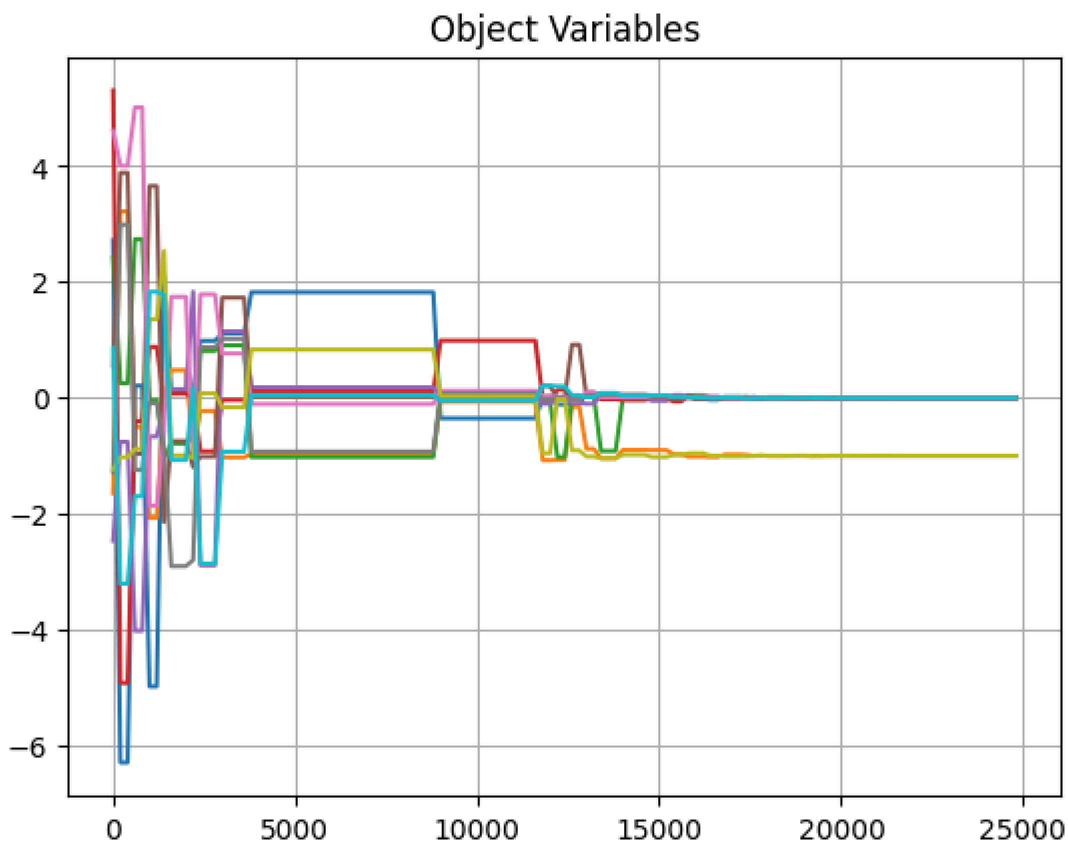
```
c:\Users\smain\AppData\Local\Programs\Python\Python310\lib\site-packages\deap\cre
ator.py:185: RuntimeWarning: A class named 'Individual' has already been created
and it will be overwritten. Consider deleting previous creation of that class or
rename it.
  warnings.warn("A class named '{0}' has already been created and it "
```

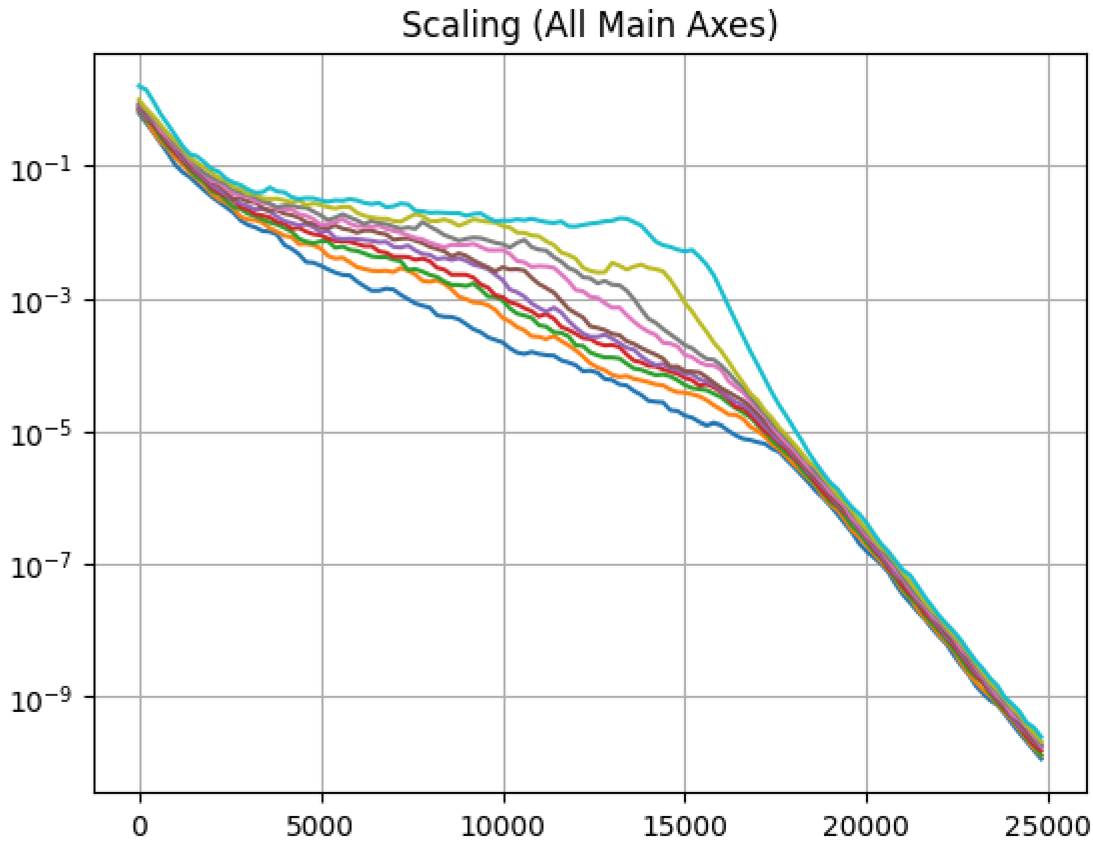
gen	evals	std	min	avg	max
0	200	188.36	217.082	576.281	1199.71
1	200	267.771	177.951	664.621	1886.26
2	200	278.323	241.077	697.146	1801.67
3	200	218.546	137.702	513.883	1307.41
4	200	131.992	140.886	374.041	739.372
5	200	84.5236	113.862	291.389	625.977
6	200	66.4526	126.375	243.34	469.926
7	200	49.7153	82.8412	197.511	348.043
8	200	38.3504	73.0274	167.743	309.409
9	200	37.4221	76.7296	162.881	337.876
10	200	31.584	90.0571	151.457	250.642
11	200	28.7502	60.9416	137.673	255.886
12	200	30.5608	58.3516	136.254	242.471
13	200	28.2293	61.0749	132.133	200.212
14	200	24.8966	73.6083	128.44	201.353
15	200	24.0441	44.6969	121.76	184.873
16	200	23.8312	66.5502	117.786	183.772
17	200	22.3703	49.7996	118.032	173.369
18	200	21.2466	50.2553	111.336	164.842
19	200	23.693	29.7588	110.346	167.159
20	200	23.1049	37.4303	108.531	166.706
21	200	23.6721	52.7345	112.505	158.215
22	200	21.6273	46.0387	108.519	156.803
23	200	24.2514	51.2927	109.15	168.51
24	200	21.8127	50.5265	106.74	157.684
25	200	20.9207	49.9726	105.805	156.414
26	200	22.3691	47.8372	106.215	161.921
27	200	20.6131	60.8783	103.797	163.346
28	200	22.352	55.5257	104.4	165.112
29	200	21.8205	45.1657	105.466	157.527
30	200	24.2144	37.5293	103.281	169.153
31	200	23.3052	51.1174	104.381	160.517
32	200	19.8094	54.9117	102.966	152.273
33	200	22.7628	45.9637	104.637	163.683
34	200	23.836	39.5106	103.974	161.04
35	200	23.1809	38.5163	104.198	167.82
36	200	23.2859	45.9853	105.027	159.627
37	200	21.0683	43.7065	105.435	150.396
38	200	21.438	51.4768	104.409	163.654
39	200	21.7997	50.6494	107.628	164.917
40	200	22.9667	57.3794	102.934	164.499
41	200	21.5094	42.6787	101.998	162.001
42	200	20.8918	47.9922	102.634	160.58
43	200	21.5459	52.3624	103.082	161.47
44	200	20.964	46.175	99.2767	170.321
45	200	21.501	23.4369	96.4574	161.724
46	200	22.3494	43.4501	95.5377	157.647
47	200	20.1508	55.0854	94.2775	154.576
48	200	20.4639	33.8392	94.3716	146.512
49	200	21.3419	37.3176	93.4708	159.105
50	200	20.3809	30.7809	92.0164	153.89
51	200	21.2228	23.9863	90.3982	140.441
52	200	20.2319	26.3634	88.0891	139.205
53	200	20.1861	27.1996	86.8638	146.324
54	200	20.7251	32.0014	83.8805	155.465
55	200	20.1298	33.3938	80.4648	131.114
56	200	19.6675	25.8042	76.8244	123.683
57	200	20.0037	36.6472	74.2462	136.131
58	200	18.753	24.3073	69.4367	113.338

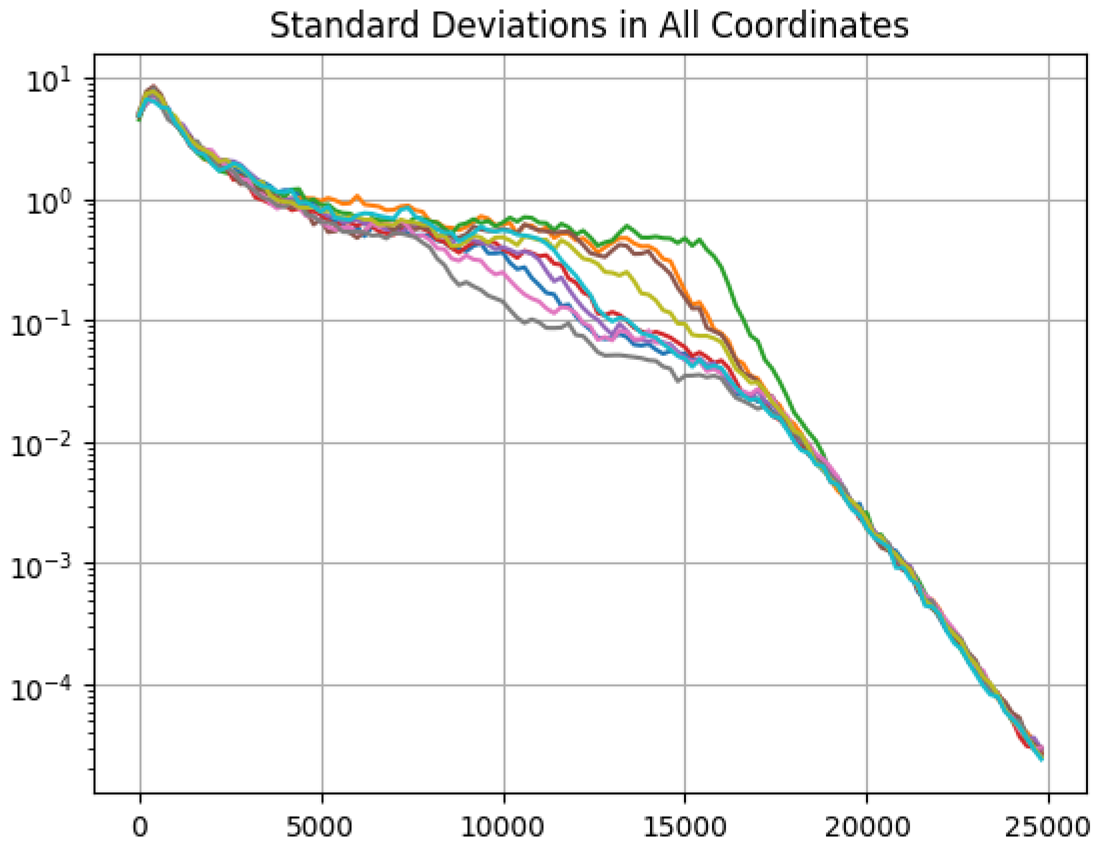
59	200	20.7605	21.6832	68.2677	152.739	
60	200	17.209	22.9793	63.2001	107.212	
61	200	15.909	16.3833	59.0137	104.109	
62	200	15.3991	17.2701	54.9715	99.7616	
63	200	13.5254	13.837	51.9443	89.3434	
64	200	14.1851	15.1459	49.015	88.9504	
65	200	15.1709	11.1446	47.5316	94.154	
66	200	13.3983	17.5089	50.6176	84.4196	
67	200	14.0185	9.08003	47.6193	86.302	
68	200	15.4981	13.3833	43.9521	84.7167	
69	200	14.3087	11.6821	40.8743	78.6644	
70	200	16.3664	5.61926	41.701	81.3813	
71	200	14.5372	7.38144	40.318	84.1218	
72	200	14.3096	6.29823	36.2936	69.7972	
73	200	13.6413	5.9195	34.9814	72.8342	
74	200	12.8216	5.51425	30.6979	70.9974	
75	200	10.857	5.96679	25.0592	58.3896	
76	200	9.43346	5.84021	23.2591	62.6284	
77	200	9.4706	3.83506	22.2552	50.5876	
78	200	8.8639	4.51084	19.3052	45.9704	
79	200	7.87684	3.42895	17.9275	35.3506	
80	200	7.64876	3.06865	14.9991	38.0132	
81	200	6.04097	3.1157	10.823	28.0705	
82	200	3.68263	2.45807	7.3001	19.8534	
83	200	2.59992	2.47261	5.6836	16.6489	
84	200	1.70882	2.22424	4.46063	13.1404	
85	200	1.55203	2.34965	4.14534	13.4844	
86	200	0.83579	2.30114	3.41059	6.73155	
87	200	0.570705		2.20132	3.03196	5.16571
88	200	0.380897		2.13355	2.72783	4.17252
89	200	0.271888		2.11179	2.48252	4.20811
90	200	0.156339		2.02693	2.30529	2.89079
91	200	0.116792		2.03792	2.22379	2.70929
92	200	0.0848562		2.0306	2.16286	2.42626
93	200	0.0503107		2.01237	2.10547	2.28392
94	200	0.0433387		2.00331	2.08067	2.27561
95	200	0.0261688		2.00064	2.04815	2.12766
96	200	0.0190777		2.00045	2.03426	2.13991
97	200	0.0125604		1.9953	2.01817	2.06475
98	200	0.00917564		1.99359	2.00882	2.05243
99	200	0.0064618		1.99324	2.00423	2.0337
100	200	0.00439964		1.99167	1.99953	2.01239
101	200	0.00286551		1.991	1.99608	2.00613
102	200	0.00221921		1.99069	1.99493	2.00493
103	200	0.0018175		1.99069	1.9937	2.0018
104	200	0.00112417		1.99014	1.99237	1.9958
105	200	0.00103508		1.99026	1.99187	1.99564
106	200	0.000636181		1.99025	1.99141	1.9937
107	200	0.000425067		1.99012	1.99079	1.99242
108	200	0.000240936		1.99002	1.99044	1.99123
109	200	0.00020636		1.99001	1.99035	1.99142
110	200	0.000132357		1.98996	1.99022	1.9907
111	200	9.14258e-05		1.98995	1.99011	1.99047
112	200	5.72182e-05		1.98995	1.99005	1.99022
113	200	4.59596e-05		1.98994	1.99001	1.99016
114	200	2.6461e-05		1.98993	1.98997	1.9901
115	200	2.01445e-05		1.98993	1.98996	1.99007
116	200	1.16758e-05		1.98993	1.98995	1.98999
117	200	8.52671e-06		1.98992	1.98994	1.98996
118	200	5.78396e-06		1.98992	1.98993	1.98995

```
119     200     4.02671e-06     1.98992 1.98993 1.98994
120     200     2.44438e-06     1.98992 1.98992 1.98993
121     200     1.74412e-06     1.98992 1.98992 1.98993
122     200     1.17254e-06     1.98992 1.98992 1.98993
123     200     9.4393e-07      1.98992 1.98992 1.98992
124     200     7.50613e-07     1.98992 1.98992 1.98992
```









Le bloc de code précédent implémente un algorithme d'optimisation évolutif basé sur CMA-ES (Covariance Matrix Adaptation Evolution Strategy) pour minimiser la fonction de Rastrigin, une fonction de test utilisée en optimisation.

Il commence par définir une stratégie CMA-ES et une boîte à outils (toolbox) qui gère la génération et l'évaluation des individus. L'algorithme évolue sur 125 générations, chaque génération contenant 10 individus, et utilise un processus d'évaluation, sélection et mise à jour pour améliorer progressivement la solution.

Au fil des générations, différentes métriques sont enregistrées : meilleure solution, moyenne, écart-type, ainsi que des paramètres liés à CMA-ES, comme la matrice de covariance et les axes principaux.

Enfin, des graphiques sont générés pour visualiser l'évolution des variables clés, notamment :

L'évolution des valeurs de la fonction objective (fonction Rastrigin). La meilleure solution trouvée. Les axes de la matrice de covariance. Les écarts-types des individus au fil du temps. Ce programme est un exemple avancé d'optimisation évolutionnaire, démontrant comment une stratégie adaptative comme CMA-ES ajuste dynamiquement ses paramètres pour converger vers une solution optimale.

In [4]:

```
import operator
import math
import random

import numpy as np
from deap import algorithms, base, creator, tools, gp

# Define new functions
def division_operator(numerator, denominator):
    if denominator == 0:
        return 1

    return numerator / denominator

# Define the evaluation function
def eval_func(individual, points):
    # Transform the tree expression in a callable function
    func = toolbox.compile(expr=individual)

    # Evaluate the mean squared error
    mse = ((func(x) - (2 * x**3 - 3 * x**2 + 4 * x - 1))**2 for x in points)

    return math.fsum(mse) / len(points),

# Function to create the toolbox
def create_toolbox():
    pset = gp.PrimitiveSet("MAIN", 1)
    pset.addPrimitive(operator.add, 2)
    pset.addPrimitive(operator.sub, 2)
    pset.addPrimitive(operator.mul, 2)
    pset.addPrimitive(division_operator, 2)
    pset.addPrimitive(operator.neg, 1)
    pset.addPrimitive(math.cos, 1)
    pset.addPrimitive(math.sin, 1)

    pset.addEphemeralConstant("rand101", lambda: random.randint(-1,1))

    pset.renameArguments(ARG0='x')
```

```
    creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
    creator.create("Individual", gp.PrimitiveTree, fitness=creator.FitnessMin)

    toolbox = base.Toolbox()

    toolbox.register("expr", gp.genHalfAndHalf, pset=pset, min_=1, max_=2)
    toolbox.register("individual", tools.initIterate, creator.Individual, toolbc
    toolbox.register("population", tools.initRepeat, list, toolbox.individual)
    toolbox.register("compile", gp.compile, pset=pset)
    toolbox.register("evaluate", eval_func, points=[x/10. for x in range(-10,10)
    toolbox.register("select", tools.selTournament, tournsize=3)
    toolbox.register("mate", gp.cxOnePoint)
    toolbox.register("expr_mut", gp.genFull, min_=0, max_=2)
    toolbox.register("mutate", gp.mutUniform, expr=toolbox.expr_mut, pset=pset)

    toolbox.decorate("mate", gp.staticLimit(key=operator.attrgetter("height"), m
    toolbox.decorate("mutate", gp.staticLimit(key=operator.attrgetter("height"),

    return toolbox

if __name__ == "__main__":
    random.seed(7)

    toolbox = create_toolbox()

    population = toolbox.population(n=450)
    hall_of_fame = tools.HallOfFame(1)

    stats_fit = tools.Statistics(lambda x: x.fitness.values)
    stats_size = tools.Statistics(len)

    mstats = tools.MultiStatistics(fitness=stats_fit, size=stats_size)
    mstats.register("avg", np.mean)
    mstats.register("std", np.std)
    mstats.register("min", np.min)
    mstats.register("max", np.max)

    # Define parameters
    probab_crossover = 0.4
    probab_mutate = 0.2
    num_generations = 60

    population, log = algorithms.eaSimple(population, toolbox,
            probab_crossover, probab_mutate, num_generations,
            stats=mstats, halloffame=hall_of_fame, verbose=True)
```

```
c:\Users\smain\AppData\Local\Programs\Python\Python310\lib\site-packages\deap\gp.
py:257: RuntimeWarning: Ephemeral rand101 function cannot be pickled because its
generating function is a lambda function. Use functools.partial instead.
  warnings.warn("Ephemeral {name} function cannot be "
c:\Users\smain\AppData\Local\Programs\Python\Python310\lib\site-packages\deap\cre
ator.py:185: RuntimeWarning: A class named 'FitnessMin' has already been created
and it will be overwritten. Consider deleting previous creation of that class or
rename it.
  warnings.warn("A class named '{0}' has already been created and it "
```

```
                                    fitness
size
                ----------------------------------------------- -----------------
------------------------------
gen     nevals  avg     gen     max     min     nevals  std     avg     gen     m
ax      min     nevals  std
0       450     18.6918 0       47.1923 7.39087 450     6.27543 3.73556 0       7
2       450     1.62449
1       251     15.4572 1       41.3823 4.46965 251     4.54993 3.80222 1       1
2       1       251     1.81316
2       236     13.2545 2       37.7223 4.46965 236     4.06145 3.96889 2       1
2       1       236     1.98861
3       251     12.2299 3       60.828  4.46965 251     4.70055 4.19556 3       1
2       1       251     1.9971
4       235     11.001  4       47.1923 4.46965 235     4.48841 4.84222 4       1
3       1       235     2.17245
5       229     9.44483 5       31.478  4.46965 229     3.8796  5.56    5       1
9       1       229     2.43168
6       225     8.35975 6       22.0546 3.02133 225     3.40547 6.38889 6       1
5       1       225     2.40875
7       237     7.99309 7       31.1356 1.81133 237     4.08463 7.14667 7       1
6       1       237     2.57782
8       224     7.42611 8       359.418 1.17558 224     17.0167 8.33333 8       1
9       1       224     3.11127
9       237     5.70308 9       24.1921 1.17558 237     3.71991 9.64444 9       2
3       1       237     3.31365
10      254     5.27991 10      30.4315 1.13301 254     4.13556 10.5089 10      2
5       1       254     3.51898
11      223     4.26809 11      18.7774 0.841562        223     3.16748 11.42   1
1       25      1       223     3.69613
12      249     4.0672  12      20.6729 0.686362        249     3.50578 12.34   1
2       27      1       249     4.34843
13      242     5.78507 13      1060    0.841562        242     49.873  13.4089 1
3       27      1       242     4.24912
14      221     3.28494 14      26.6665 0.841562        221     3.31483 14.1556 1
4       37      1       221     4.62772
15      243     2.98754 15      42.7003 0.756579        243     3.43806 14.8689 1
5       29      1       243     4.28363
16      258     3.1341  16      70.8279 0.61024         258     4.75699 15.26   1
6       30      1       258     4.46631
17      223     5.34041 17      1059.66 0.61024         223     49.8772 15.8733 1
7       44      1       223     5.03847
18      229     4.53387 18      776.269 0.508052        229     36.5925 16.8667 1
8       39      1       229     4.7949
19      204     2.65489 19      41.3823 0.508052        204     3.76843 16.84   1
9       31      1       204     4.3814
20      222     2.63441 20      28.3123 0.113042        222     3.65431 17.7267 2
0       46      1       222     4.91062
21      223     2.2272  21      26.038  0.113042        223     3.10943 18.9289 2
1       41      1       223     4.90254
22      227     2.21592 22      58.3649 0.113042        227     3.93487 19.6156 2
2       41      1       227     4.7927
23      244     2.2172  23      21.6536 0.113042        244     3.31138 19.7844 2
3       38      1       244     4.52845
24      237     2.19195 24      22.0546 0.113042        237     3.49711 20.3467 2
4       37      1       237     4.4741
25      223     1.95396 25      29.6646 0.113042        223     3.52879 20.4978 2
5       36      1       223     4.42279
26      246     1.94503 26      86.1329 0.113042        246     4.96868 21.3867 2
6       39      1       246     4.79322
```

```
27      236     1.80352 27      22.9113 0.113042        236     3.0955  21.78   2
7       41      1       236     4.5683
28      245     1.59749 28      20.6729 0.0601253       245     2.90415 21.9333 2
8       36      1       245     4.18834
29      205     1.45095 29      37.0156 0.0509541       205     3.35453 22.6844 2
9       41      1       205     4.09585
30      239     1.79087 30      49.9667 0.0509541       239     4.05304 22.5711 3
0       40      1       239     4.70135
31      250     1.88413 31      28.4646 0.0509541       250     3.90833 23.0111 3
1       41      1       250     5.34352
32      248     1.37476 32      16.4446 0.0509541       248     2.50597 23.7689 3
2       44      1       248     5.23067
33      239     1.446   33      22.0546 0.0509541       239     3.16672 24.2289 3
3       44      1       239     5.44129
34      244     1.50939 34      23.658  0.0474957       244     3.21624 24.7556 3
4       47      1       244     5.41215
35      209     0.956873        35      16.548  0.0474957       209     2.06747 2
5.66    35      44      1       209     4.5154
36      209     1.10464         36      22.0546 0.0474957       209     2.71898 2
6.4867  36      46      1       209     5.23289
37      258     1.61958         37      86.0936 0.0382386       258     6.1839  2
7.2111  37      45      3       258     4.75557
38      257     2.03651         38      70.4768 0.0342642       257     5.15243 2
6.5311  38      49      1       257     6.22327
39      235     1.95531         39      185.328 0.0472693       235     9.32516 2
6.9711  39      48      1       235     6.00345
40      234     1.51403         40      28.5529 0.0472693       234     3.24513 2
6.6867  40      52      1       234     5.39811
41      230     1.4753          41      70.4768 0.0472693       230     5.4607  2
7.1     41      46      3       230     4.7433
42      233     12.3648         42      4880.09 0.0396503       233     229.754 2
6.88    42      53      1       233     5.18192
43      251     1.807           43      86.0936 0.0396503       251     5.85281 2
6.4889  43      50      1       251     5.43741
44      236     9.30096         44      3481.25 0.0277886       236     163.888 2
6.9622  44      55      1       236     6.27169
45      231     1.73196         45      86.7372 0.0342642       231     6.8119  2
7.4711  45      51      2       231     5.27807
46      227     1.86086         46      185.328 0.0342642       227     10.1143 2
8.0644  46      56      1       227     6.10812
47      216     12.5214         47      4923.66 0.0342642       216     231.837 2
9.1022  47      54      1       216     6.45898
48      232     14.3469         48      5830.89 0.0322462       232     274.536 2
9.8244  48      58      3       232     6.24093
49      242     2.56984         49      272.833 0.0322462       242     18.2752 2
9.9267  49      51      1       242     6.31446
50      227     2.80136         50      356.613 0.0322462       227     21.0416 2
9.7978  50      56      4       227     6.50275
51      243     1.75099         51      86.0936 0.0322462       243     5.70833 2
9.8089  51      56      1       243     6.62379
52      253     10.9184         52      3435.84 0.0227048       253     163.602 2
9.9911  52      55      1       253     6.66833
53      243     1.80265         53      48.0418 0.0227048       243     4.73856 2
9.88    53      55      1       243     7.33084
54      234     1.74487         54      86.0936 0.0227048       234     6.0249  3
0.6067  54      55      1       234     6.85782
55      220     1.58888         55      31.094  0.0132398       220     3.82809 3
0.5644  55      54      1       220     6.96669
56      234     1.46711         56      103.287 0.00766444      234     6.81157 3
0.6689  56      55      3       234     6.6806
```

```
57      250     17.0896         57      6544.17 0.00424267      250     308.689 3
1.1267  57      60      4       250     7.25837
58      231     1.66757         58      141.584 0.00144401      231     7.35306 3
2       58      52      1       231     7.23295
59      229     2.22325         59      265.224 0.00144401      229     13.388  3
3.5489  59      64      1       229     8.38351
60      248     2.60303         60      521.804 0.00144401      248     24.7018 3
5.2533  60      58      1       248     7.61506
```

Le bloc de code précédent utilise la programmation génétique (GP) avec DEAP pour approximer une fonction mathématique. Il génère une population d'expressions mathématiques sous forme d'arbres, cherchant à se rapprocher de f(x) = $2x^3 - 3x^2 + 4x - 1$.

L'algorithme suit ces étapes :

Création d'individus avec des opérations de base (+, -, *, /, sin, cos). Évaluation de la précision des expressions via l'erreur quadratique moyenne (MSE). Évolution sur 60 générations en appliquant crossover (40%) et mutation (20%). Sélection des meilleurs individus via un tournoi et stockage du meilleur modèle. À la fin, il affiche les statistiques d'évolution et la meilleure équation trouvée.

In [9]:

```python
import copy
import random
from functools import partial
import os
import numpy as np
from deap import algorithms, base, creator, tools, gp

class RobotController(object):
    def __init__(self, max_moves):
        self.max_moves = max_moves
        self.moves = 0
        self.consumed = 0
        self.routine = None
        self.direction = ["north", "east", "south", "west"]
        self.direction_row = [1, 0, -1, 0]
        self.direction_col = [0, 1, 0, -1]

    def _reset(self):
        self.row = self.row_start
        self.col = self.col_start
        self.direction = 1
        self.moves = 0
        self.consumed = 0
        self.matrix_exc = copy.deepcopy(self.matrix)

    def _conditional(self, condition, out1, out2):
        out1() if condition() else out2()

    def turn_left(self):
        if self.moves < self.max_moves:
            self.moves += 1
            self.direction = (self.direction - 1) % 4

    def turn_right(self):
```

```
        if self.moves < self.max_moves:
            self.moves += 1
            self.direction = (self.direction + 1) % 4

    def move_forward(self):
        if self.moves < self.max_moves:
            self.moves += 1
            self.row = (self.row + self.direction_row[self.direction]) % self.ma
            self.col = (self.col + self.direction_col[self.direction]) % self.ma

            if self.matrix_exc[self.row][self.col] == "target":
                self.consumed += 1

            self.matrix_exc[self.row][self.col] = "passed"

    def sense_target(self):
        ahead_row = (self.row + self.direction_row[self.direction]) % self.matri
        ahead_col = (self.col + self.direction_col[self.direction]) % self.matri
        return self.matrix_exc[ahead_row][ahead_col] == "target"

    def if_target_ahead(self, out1, out2):
        return partial(self._conditional, self.sense_target, out1, out2)

    def run(self, routine):
        self._reset()
        while self.moves < self.max_moves:
            routine()

    def traverse_map(self, matrix):
        self.matrix = list()
        for i, line in enumerate(matrix):
            self.matrix.append(list())

            for j, col in enumerate(line.strip()):  # Supprimer espaces inutiles
                if col == "#":
                    self.matrix[-1].append("target")
                elif col == ".":
                    self.matrix[-1].append("empty")
                elif col == "S":
                    self.matrix[-1].append("empty")
                    self.row_start = self.row = i
                    self.col_start = self.col = j
                    self.direction = 1

        self.matrix_row = len(self.matrix)
        self.matrix_col = len(self.matrix[0])
        self.matrix_exc = copy.deepcopy(self.matrix)

class Prog(object):
    def _progn(self, *args):
        for arg in args:
            arg()

    def prog2(self, out1, out2):
        return partial(self._progn, out1, out2)

    def prog3(self, out1, out2, out3):
        return partial(self._progn, out1, out2, out3)

def eval_func(individual):
```

```
    global robot, pset
    routine = gp.compile(individual, pset)
    robot.run(routine)
    return robot.consumed,

def create_toolbox():
    global robot, pset
    pset = gp.PrimitiveSet("MAIN", 0)
    pset.addPrimitive(robot.if_target_ahead, 2)
    pset.addPrimitive(Prog().prog2, 2)
    pset.addPrimitive(Prog().prog3, 3)
    pset.addTerminal(robot.move_forward)
    pset.addTerminal(robot.turn_left)
    pset.addTerminal(robot.turn_right)

    if not hasattr(creator, "FitnessMax"):
        creator.create("FitnessMax", base.Fitness, weights=(1.0,))
    if not hasattr(creator, "Individual"):
        creator.create("Individual", gp.PrimitiveTree, fitness=creator.FitnessMa

    toolbox = base.Toolbox()
    toolbox.register("expr_init", gp.genFull, pset=pset, min_=1, max_=2)
    toolbox.register("individual", tools.initIterate, creator.Individual, toolbc
    toolbox.register("population", tools.initRepeat, list, toolbox.individual)
    toolbox.register("evaluate", eval_func)
    toolbox.register("select", tools.selTournament, tournsize=7)
    toolbox.register("mate", gp.cxOnePoint)
    toolbox.register("expr_mut", gp.genFull, min_=0, max_=2)
    toolbox.register("mutate", gp.mutUniform, expr=toolbox.expr_mut, pset=pset)

    return toolbox

if __name__ == "__main__":
    global robot
    random.seed(7)
    max_moves = 750
    robot = RobotController(max_moves)
    toolbox = create_toolbox()

    # 📌 Spécifier le chemin absolu
    file_path = r"C:\Users\smain\Downloads\target_map.txt"

    # 📌 Vérification si le fichier existe
    if not os.path.exists(file_path):
        print(f"❌ ERREUR : Le fichier {file_path} est introuvable ! Vérifiez sc
        exit(1)

    # 📌 Lecture du fichier et chargement de la carte
    with open(file_path, 'r') as f:
        map_data = f.readlines()
    robot.traverse_map(map_data)
    print("✅ Carte chargée avec succès")

    # 📌 Initialisation de la population et des statistiques
    population = toolbox.population(n=400)
    hall_of_fame = tools.HallOfFame(1)
    stats = tools.Statistics(lambda x: x.fitness.values)
    stats.register("avg", np.mean)
    stats.register("std", np.std)
    stats.register("min", np.min)
```

```python
    stats.register("max", np.max)

    # 📌 Exécution de l'algorithme évolutif
    print("✅ Début de l'évolution génétique")
    algorithms.eaSimple(population, toolbox, 0.4, 0.3, 50, stats, halloffame=hal
```

```
✅ Carte chargée avec succès
✅ Début de l'évolution génétique
gen	nevals	avg    	std    	min	max
0  	400   	1.4875 	4.37491	0  	62 
1  	231   	4.285  	7.56993	0  	73 
2  	235   	10.8925	14.8493	0  	73 
3  	231   	21.72  	22.1239	0  	73 
4  	238   	29.9775	27.7861	0  	76 
5  	224   	37.6275	31.8698	0  	76 
6  	231   	42.845 	33.0541	0  	80 
7  	223   	43.55  	33.9369	0  	83 
8  	234   	44.0675	34.5201	0  	83 
9  	231   	49.2975	34.3065	0  	83 
10 	249   	47.075 	36.4106	0  	93 
11 	222   	52.7925	36.2826	0  	97 
12 	248   	51.0725	37.2598	0  	97 
13 	234   	54.01  	37.4614	0  	97 
14 	229   	59.615 	37.7894	0  	97 
15 	228   	63.3   	39.8205	0  	97 
16 	220   	64.605 	40.3962	0  	97 
17 	236   	62.545 	40.5607	0  	97 
18 	233   	67.99  	38.9033	0  	97 
19 	236   	66.4025	39.6574	0  	97 
20 	221   	69.785 	38.7117	0  	97 
21 	244   	65.705 	39.0957	0  	97 
22 	230   	70.32  	37.1206	0  	97 
23 	241   	67.3825	39.4028	0  	97 
24 	227   	69.265 	38.8828	0  	97 
25 	230   	68.9875	38.2422	0  	97 
26 	214   	71.505 	36.964 	0  	97 
27 	246   	72.72  	37.1637	0  	97 
28 	238   	73.5975	36.5385	0  	97 
29 	239   	76.405 	35.5696	0  	97 
30 	246   	78.6025	33.4281	0  	97 
31 	240   	74.83  	36.5157	0  	97 
32 	216   	80.2625	32.6659	0  	97 
33 	220   	80.6425	33.0933	0  	97 
34 	247   	78.245 	34.6022	0  	97 
35 	241   	81.22  	32.1885	0  	97 
36 	234   	83.6375	29.0002	0  	97 
37 	228   	82.485 	31.7354	0  	97 
38 	219   	83.4625	30.0592	0  	97 
39 	212   	88.64  	24.2702	0  	97 
40 	231   	86.7275	27.0879	0  	97 
41 	229   	89.1825	23.8773	0  	97 
42 	216   	87.96  	25.1649	0  	97 
43 	218   	86.85  	27.1116	0  	97 
44 	236   	88.78  	23.7278	0  	97 
45 	225   	89.115 	23.4212	0  	97 
46 	232   	88.5425	24.187 	0  	97 
47 	245   	87.7775	25.3909	0  	97 
48 	231   	87.78  	26.3786	0  	97 
49 	238   	88.8525	24.5115	0  	97 
50 	233   	87.82  	25.4164	1  	97 
```

Le bloc de code précédent implémente un robot explorateur basé sur la programmation génétique (GP) avec DEAP. Il vise à optimiser le déplacement du robot pour consommer un maximum de cibles dans un environnement donné.

🔹 RobotController : Définit les mouvements du robot (tourner, avancer) et détecte les cibles. 🔹 Carte d'exploration : Chargée depuis un fichier texte (target_map.txt), elle contient des cibles #, des cases vides . et un point de départ S. 🔹 Programmation génétique :

Individus = arbres représentant une séquence d'actions du robot. Évaluation = nombre de cibles consommées. Sélection, mutation et croisement optimisent l'exploration. 🔹 Exécution : Chargement de la carte. Initialisation d'une population de 400 stratégies. Exécution sur 50 générations pour trouver la meilleure solution.