09 - Springs

April 3, 2023

0.0.1 An engineering problem from "Engineering Optimization" by Xin-She Yang

Goal: Design an optimal spring (cheapest / least material needed) that does the job. Parameters we can change: d, the diameter of the coil; L, the length of the spring; w, the thickness of the wire.

Task: Minimize

$$(2+L)dw^2$$

subject to the constraints

$$g_1(L,d,w) = 1 - \frac{d^3 L}{7178 w^4} \leq 0$$

$$g_2(L,d,w) = \frac{4d^2 - wd}{12566 dw^3 - w^4} + \frac{1}{5108 w^2} - 1 \leq 0$$

$$g_3(L,d,w) = 1 - \frac{140.45 w}{d^2 L} \leq 0$$

$$g_4(L,d,w) = \frac{w+d}{1.5} - 1 \leq 0$$

with boundary conditions

$$0.05 \leq w \leq 2.0 \qquad 0.25 \leq d \leq 1.3 \qquad 2.0 \leq L \leq 15.0$$

[1]:
```
import math
import random
```

[3]:
```
def random_in_range(lower, upper):
    return random.random() * (upper - lower) + lower
random_in_range(2, 15)
```

[3]:
```
12.214568759007836
```

[4]:
```
def g1(L,d,w):
    return 1 - d**3 * L / (7178 * w**4)

def g2(L,d,w):
    return (4*d**2 - w*d)/(12566 * d * w**3 - w**4) + 1/(5108*w**2) - 1

def g3(L,d,w):
    return 1 - 140.45 * w / (d**2*L)
```

```
def g4(L,d,w):
    return (w+d)/1.5 - 1

def satisfies_constraints(L,d,w):
    return g1(L,d,w) <= 0 and g2(L,d,w) <= 0 and g3(L,d,w) <= 0 and g4(L,d,w)<= 0
```

[5]:
```
def score(L,d,w):  # w-squared is "w**2" not "w^2"
    return (2+L)*d*w**2
```

[14]:
```
def tweak(L,d,w):
    delta_w = 0.01
    delta_d = 0.01
    delta_L = 0.1

    new_w = w + random_in_range(-1, 1) * delta_w
    while new_w < 0.05 or new_w > 2:
        new_w = w + random_in_range(-1, 1) * delta_w

    new_d = d + random_in_range(-1, 1) * delta_d
    while new_d < 0.25 or new_d > 1.3:
        new_d = d + random_in_range(-1, 1) * delta_d

    new_L = L + random_in_range(-1, 1) * delta_L
    while new_L < 2 or new_L > 15:
        new_L = L + random_in_range(-1, 1) * delta_L

    return (new_L, new_d, new_w)

tweak(15, 1.3, 2.0)
```

[14]:
```
(14.902588028615785, 1.2942705555373368, 1.9975626174453116)
```

[15]:
```
def random_solution():
    return (
        random_in_range(2, 15),
        random_in_range(0.25, 1.3),
        random_in_range(0.05, 2),
    )
```

[16]:
```
def hill_climbing():

    start = random_solution()
    while not satisfies_constraints(*start):
#         print(g1(*start), g2(*start), g3(*start), g4(*start))
        start = random_solution()
```

```
    sol = start
    value = score(*sol)

    bad = 0
    while True:
#         print(sol)
        new_sol = tweak(*sol)
        while not satisfies_constraints(*new_sol):
            new_sol = tweak(*sol)
        new_value = score(*new_sol)
        if new_value < value:
            bad = 0
            sol = new_sol
            value = new_value
        else:
            bad += 1
            if bad > 1000:
                print(value)
                return sol
```

[17]:
```
sol = hill_climbing()
#sol = [2.02310938, 0.25113418, 0.052182225]
print(sol)
print(g1(*sol), g2(*sol), g3(*sol), g4(*sol))
print(score(*sol))
```

```
0.008884266253488936
(12.177199077248762, 0.2502461418463225, 0.05004167852402444)
-3.2395391411204155 -0.3179167694642967 -8.216614547819589 -0.7998081197531021
0.008884266253488936
```

[ ]:

[ ]:

[ ]:
```
# smarter: sample 1000 tweaks to find a good initial_temp that gives a desired␣
 ↪p_0
# or: heat the system slowly until the % of worsening solutions is what you want
initial_temp = 0.005
alpha = 0.999
final_temp = initial_temp / 10000
trials_per_temp = 1000

start = random_solution()
while not satisfies_constraints(*start):
    start = random_solution()
sol = start
```

```
value = score(*sol)
```

[ ]:
```
temp = initial_temp
generation = 0
best_sol = None
best_score = None

while temp >= final_temp:
    generation += 1
    accepted_worse = 0
    total_worse = 0
    for i in range(trials_per_temp):
        new_sol = tweak(*sol)
        while not satisfies_constraints(*new_sol):
            new_sol = tweak(*sol)

        new_value = score(*new_sol)

        delta = new_value - value
        delta *= -1
        if delta >= 0:
            sol = new_sol
            value = new_value
            if best_score is None or value < best_score:
                best_sol = sol
                best_score = value
        else:
            total_worse += 1
            p = math.exp(delta/temp)
            r = random.random()
            if r <= p:
                accepted_worse += 1
                sol = new_sol
                value = new_value

    print(
        f"Gen #{generation}: temp = {temp:.6f}, "
        f"best score = {best_score:.8f}, "
        f"cur score = {value:.8f}, "
        f"worse accepted = {round(accepted_worse/total_worse*100,2):.2f}%"
    )
    temp = temp * alpha
```

[ ]:
```
best_sol
```

[ ]:
```
sol = best_sol
print(g1(*sol), g2(*sol), g3(*sol), g4(*sol))
```

[ ]:
```
score(*sol)
```

[ ]:
```

```