Optimization Methods PS s7 Lab 1

Joris Plaščinskas

September 2024

1 One Dimension Optimization

The goal of this laboratory work is to get familiar with one dimensional optimization methods. I will try to minimize this 4th degree polynomial function: $(\frac{(x^2-a)^2}{b}) - 1$. In my case: $a = 2, b = 2$, so: $\frac{(x^2-2)^2}{2}) - 1$ because my student number is 2016020. I will use 3 different optimization methods and try to compare them in the end.

2 Interval Split Method

I chose to use 3-point interval split method. The algorithm relies on 5 points: left bound, right bound, left x, middle x, right x. Each iteration the interval is split into 4 equal parts: $I - x_1 - x_m - x_2 - r$ - here I/r are the left/right bounds. At the start of each iteration $f(x_1), f(x_m), f(x_2)$ are calculated. The algorithm then branches out:

- If $f(x_1) < f(x_m)$, then $r = x_m$.
- If $f(x_2) < f(x_m)$, then $I = x_m$.
- Else ($f(x_1) \geq f(x_m)$ and $f(x_2) \geq f(x_m)$), then $I = x_1$ and $r = x_2$.

The algorithm is slightly optimized by recycling the x_1 or x_2 and assigning them to x_m after each iteration. The algorithm implementation in code:

```
def intervalMethod(objectiveFunction, I:float, r:float, epsilon:float = 10**-4) -> float:
    iteration_count = 0
    # 1
    x_m = (I + r) / 2
    L = r - I
    f_x_m = objectiveFunction(x_m)
    while L >= epsilon:
        # 2
        x_1 = I + L/4
        x_2 = r - L/4
        f_x_1 = objectiveFunction(x_1)
```

```python
        f_x_2 = objectiveFunction(x_2)
        # 3
        if f_x_1 < f_x_m:
            r = x_m # 3.1
            x_m = x_1 # 3.2
            f_x_m = f_x_1
        # 4
        elif f_x_2 < f_x_m:
            I = x_m # 4.1
            x_m = x_2 # 4.2
            f_x_m = f_x_2
        # 5
        else:
            I = x_1
            r = x_2
        # 6
        L = r - I
        # Extra:
        iteration_count += 1
        print(f"Completed {iteration_count} iterations. I = {I:.5f}, x_1 = {x_1:.5f}, x_m =
    return x_m
```

Results

```
Completed 1 iterations. I = 0.00000, x_1 = 2.50000, x_m = 2.50000, x_2 = 7.50000, r = 5.00000 | f(x_m) = 8.03125
Completed 2 iterations. I = 0.00000, x_1 = 1.25000, x_m = 1.25000, x_2 = 3.75000, r = 2.50000 | f(x_m) = -0.90430
Completed 3 iterations. I = 0.62500, x_1 = 0.62500, x_m = 1.25000, x_2 = 1.87500, r = 1.87500 | f(x_m) = -0.90430
Completed 4 iterations. I = 0.93750, x_1 = 0.93750, x_m = 1.25000, x_2 = 1.56250, r = 1.56250 | f(x_m) = -0.90430
Completed 5 iterations. I = 1.25000, x_1 = 1.09375, x_m = 1.40625, x_2 = 1.40625, r = 1.56250 | f(x_m) = -0.99975
Completed 6 iterations. I = 1.32812, x_1 = 1.32812, x_m = 1.40625, x_2 = 1.48438, r = 1.48438 | f(x_m) = -0.99975
Completed 7 iterations. I = 1.36719, x_1 = 1.36719, x_m = 1.40625, x_2 = 1.44531, r = 1.44531 | f(x_m) = -0.99975
Completed 8 iterations. I = 1.38672, x_1 = 1.38672, x_m = 1.40625, x_2 = 1.42578, r = 1.42578 | f(x_m) = -0.99975
Completed 9 iterations. I = 1.40625, x_1 = 1.39648, x_m = 1.41602, x_2 = 1.41602, r = 1.42578 | f(x_m) = -0.99999
Completed 10 iterations. I = 1.41113, x_1 = 1.41113, x_m = 1.41602, x_2 = 1.42090, r = 1.42090 | f(x_m) = -0.99999
Completed 11 iterations. I = 1.41113, x_1 = 1.41357, x_m = 1.41357, x_2 = 1.41846, r = 1.41602 | f(x_m) = -1.00000
Completed 12 iterations. I = 1.41357, x_1 = 1.41235, x_m = 1.41479, x_2 = 1.41479, r = 1.41602 | f(x_m) = -1.00000
Completed 13 iterations. I = 1.41357, x_1 = 1.41418, x_m = 1.41418, x_2 = 1.41541, r = 1.41479 | f(x_m) = -1.00000
Completed 14 iterations. I = 1.41388, x_1 = 1.41388, x_m = 1.41418, x_2 = 1.41449, r = 1.41449 | f(x_m) = -1.00000
Completed 15 iterations. I = 1.41403, x_1 = 1.41403, x_m = 1.41418, x_2 = 1.41434, r = 1.41434 | f(x_m) = -1.00000
Completed 16 iterations. I = 1.41411, x_1 = 1.41411, x_m = 1.41418, x_2 = 1.41426, r = 1.41426 | f(x_m) = -1.00000
Completed 17 iterations. I = 1.41418, x_1 = 1.41415, x_m = 1.41422, x_2 = 1.41422, r = 1.41426 | f(x_m) = -1.00000
1.4142227172851562
```

Figure 1: Interval split results

3 Golden Ratio Search Method

Golden Ratio Search is in principal also an interval split method and is very similar to the previous method. The main difference is that the interval is only split at 2 points and the points are placed at specifically $L * 0.618...$ away from the boundaries, so that they can be recycled after each iteration. $\rho = \frac{\sqrt{5}-1}{2}$, this number is called the golden ratio and it can be found by solving a few equations. Essentially, to find the golden ratio number your self you should setup the equations so that the leftover number x is equals to one of the other x from the new interval. You can see the code that implements golden ratio search method below:

```
def goldenRatioSearchMethod(objectiveFunction, I:float, r:float, epsilon:float = 10**-4) ->
    GR = 1/((1 + math.sqrt(5))/2)
    iteration_count:int = 0
    # 1
    L = r - I
    x_1 = r - GR*L
    x_2 = I + GR*L
    f_x_1 = objectiveFunction(x_1)
    f_x_2 = objectiveFunction(x_2)
    # 4
    while L >= epsilon:
        # 2
        if f_x_1 > f_x_2:
            I = x_1
            L = r - I
            x_1 = x_2 # GR optimization
            f_x_1 = f_x_2
            x_2 = I + GR*L
            f_x_2 = objectiveFunction(x_2)
        # 3
        else:
            r = x_2
            L = r - I
            x_2 = x_1 # GR optimization
            f_x_2 = f_x_1
            x_1 = r - GR*L
            f_x_1 = objectiveFunction(x_1)
        # Extra:
        iteration_count += 1
        print(f"Completed {iteration_count} iterations. I = {I:.5f}, x_1 = {x_1:.5f}, x_2 =
    return I + L * 0.5
```

Results

```
Completed 1 iterations. I = 0.00000, x_1 = 2.36068, x_2 = 3.81966, r = 6.18034 | f(x_1) = 5.38248 f(x_2) = 78.25157
Completed 2 iterations. I = 0.00000, x_1 = 1.45898, x_2 = 2.36068, r = 3.81966 | f(x_1) = -0.99173 f(x_2) = 5.38248
Completed 3 iterations. I = 0.00000, x_1 = 0.90170, x_2 = 1.45898, r = 2.36068 | f(x_1) = -0.29559 f(x_2) = -0.99173
Completed 4 iterations. I = 0.90170, x_1 = 1.45898, x_2 = 1.80340, r = 2.36068 | f(x_1) = -0.99173 f(x_2) = -0.21594
Completed 5 iterations. I = 0.90170, x_1 = 1.24612, x_2 = 1.45898, r = 1.80340 | f(x_1) = -0.90001 f(x_2) = -0.99173
Completed 6 iterations. I = 1.24612, x_1 = 1.45898, x_2 = 1.59054, r = 1.80340 | f(x_1) = -0.99173 f(x_2) = -0.85965
Completed 7 iterations. I = 1.24612, x_1 = 1.37767, x_2 = 1.45898, r = 1.59054 | f(x_1) = -0.99480 f(x_2) = -0.99173
Completed 8 iterations. I = 1.24612, x_1 = 1.32742, x_2 = 1.37767, r = 1.45898 | f(x_1) = -0.97169 f(x_2) = -0.99480
Completed 9 iterations. I = 1.32742, x_1 = 1.37767, x_2 = 1.40873, r = 1.45898 | f(x_1) = -0.99480 f(x_2) = -0.99988
Completed 10 iterations. I = 1.37767, x_1 = 1.40873, x_2 = 1.42792, r = 1.45898 | f(x_1) = -0.99988 f(x_2) = -0.99924
Completed 11 iterations. I = 1.37767, x_1 = 1.39687, x_2 = 1.40873, r = 1.42792 | f(x_1) = -0.99881 f(x_2) = -0.99988
Completed 12 iterations. I = 1.39687, x_1 = 1.40873, x_2 = 1.41606, r = 1.42792 | f(x_1) = -0.99988 f(x_2) = -0.99999
Completed 13 iterations. I = 1.40873, x_1 = 1.41606, x_2 = 1.42059, r = 1.42792 | f(x_1) = -0.99999 f(x_2) = -0.99984
Completed 14 iterations. I = 1.40873, x_1 = 1.41326, x_2 = 1.41606, r = 1.42059 | f(x_1) = -1.00000 f(x_2) = -0.99999
Completed 15 iterations. I = 1.40873, x_1 = 1.41153, x_2 = 1.41326, r = 1.41606 | f(x_1) = -0.99997 f(x_2) = -1.00000
Completed 16 iterations. I = 1.41153, x_1 = 1.41326, x_2 = 1.41433, r = 1.41606 | f(x_1) = -1.00000 f(x_2) = -1.00000
Completed 17 iterations. I = 1.41326, x_1 = 1.41433, x_2 = 1.41499, r = 1.41606 | f(x_1) = -1.00000 f(x_2) = -1.00000
Completed 18 iterations. I = 1.41326, x_1 = 1.41392, x_2 = 1.41433, r = 1.41499 | f(x_1) = -1.00000 f(x_2) = -1.00000
Completed 19 iterations. I = 1.41392, x_1 = 1.41433, x_2 = 1.41458, r = 1.41499 | f(x_1) = -1.00000 f(x_2) = -1.00000
Completed 20 iterations. I = 1.41392, x_1 = 1.41417, x_2 = 1.41433, r = 1.41458 | f(x_1) = -1.00000 f(x_2) = -1.00000
Completed 21 iterations. I = 1.41392, x_1 = 1.41408, x_2 = 1.41417, r = 1.41433 | f(x_1) = -1.00000 f(x_2) = -1.00000
Completed 22 iterations. I = 1.41408, x_1 = 1.41417, x_2 = 1.41423, r = 1.41433 | f(x_1) = -1.00000 f(x_2) = -1.00000
Completed 23 iterations. I = 1.41417, x_1 = 1.41423, x_2 = 1.41427, r = 1.41433 | f(x_1) = -1.00000 f(x_2) = -1.00000
Completed 24 iterations. I = 1.41417, x_1 = 1.41421, x_2 = 1.41423, r = 1.41427 | f(x_1) = -1.00000 f(x_2) = -1.00000
1.4142231887499337
```

Figure 2: Golden ratio search results

4 Newton's Method

Newton's Method for finding 0's of a function has two steps: making a linear approximation of a function and then trying to reach 0 of that linear function: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ Using Newton's Method for optimization is very similar, but it relies on finding 0's of the derivative of the original function: $x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$. The first derivative of my function is: $f'(x) = 2x(x^2 - 2)$, second: $f''(x) = 6x^2 - 4$ Here is the implementation in code:

```
def newtonsMethod(objectiveFunction, x:float = 5., epsilon:float = 10**-4) -> float:
    first_derivative = jax.grad(objectiveFunction)
    second_derivative = jax.grad(first_derivative)
    iteration_count:int = 0
    step_size = epsilon + 0.1
    while step_size > epsilon:
        step_size = (first_derivative(x)/second_derivative(x))
        x = x - step_size
        iteration_count += 1
        print(f"Completed {iteration_count} iterations. Step size = {step_size:.5f}, x = {x:
    return x
```

Results

```
Completed 1 iterations. Step size = 1.57534, x = 3.42466, f(x) = 46.31971, f'(x) = 66.63205, f''(x) = 66.36967
Completed 2 iterations. Step size = 1.00395, x = 2.42070, f(x) = 6.44906, f'(x) = 18.68691, f''(x) = 31.15886
Completed 3 iterations. Step size = 0.59973, x = 1.82097, f(x) = -0.13414, f'(x) = 4.79261, f''(x) = 15.89568
Completed 4 iterations. Step size = 0.30150, x = 1.51947, f(x) = -0.95232, f'(x) = 0.93839, f''(x) = 9.85274
Completed 5 iterations. Step size = 0.09524, x = 1.42423, f(x) = -0.99960, f'(x) = 0.08097, f''(x) = 8.17056
Completed 6 iterations. Step size = 0.00991, x = 1.41432, f(x) = -1.00000, f'(x) = 0.00084, f''(x) = 8.00178
Completed 7 iterations. Step size = 0.00010, x = 1.41421, f(x) = -1.00000, f'(x) = 0.00000, f''(x) = 8.00000
Completed 8 iterations. Step size = 0.00000, x = 1.41421, f(x) = -1.00000, f'(x) = -0.00000, f''(x) = 8.00000
1.4142135
```

Figure 3: Newton's results

5 Comparison

All functions managed to reach the same minimum (when rounded to 4th decimal place). The key differences were: iteration count, computations count and code length. Newton's method was around 3 times shorter to code, compared to the other two methods. The count of computing the objective function is the same as iterations count in the Golden Ratio Search method and double the iterations count in Newton's and Interval method's (note: in Newton's method you are computing the first and second derivatives of the objective function, but not the objective function it's self).

	Interval	Golden 2	Newton's
Iterations	17	24	8
Computations	34	24	16

Table 1: Final results

6 Plots

Two plots below highlight the entire history of the three methods (red - interval, yellow - golden, blue - newton's). Each point represents where the function was calculated.

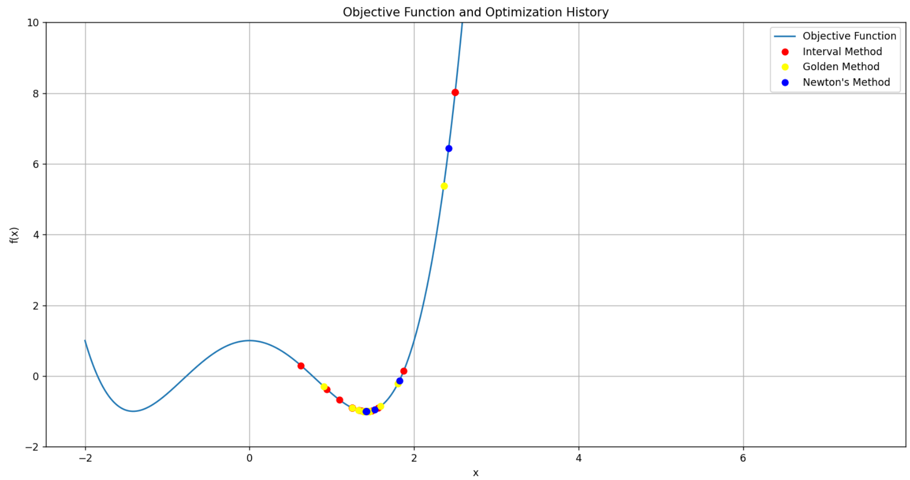

Figure 4: Objective function and optimization history plot

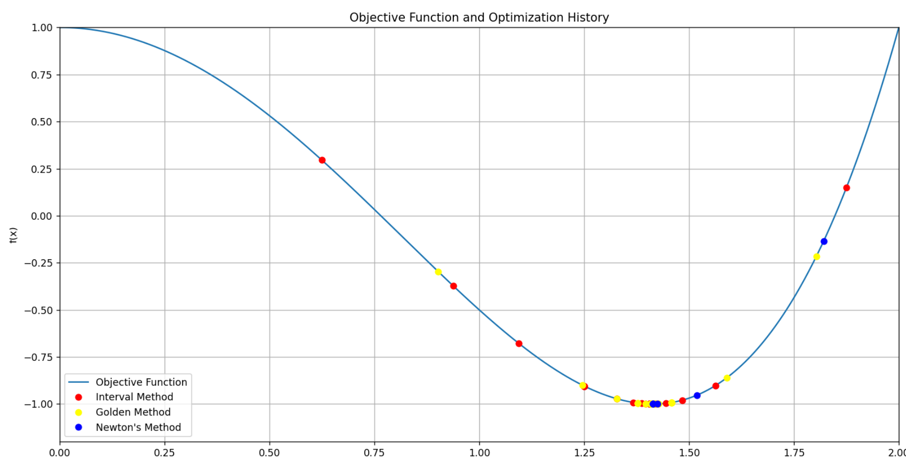

Figure 5: Zoomed in on Figure 4