MAT II - Nelder Mead

Sebastijan Trojer
MAT II 2024/25 , FRI, UL
st5804@student.uni-lj.si

I. Introduction

In this project, we were tasked with implementing the Nelder-Mead method comparing it with various gradient descend methods, and also performing optimization of 3 black-box functions. Lastly, we used the corner point method to solve the problem defined at lectures.

II. Methodology and results

A. Nelder-Mead Implementation

Nelder-Mead method uses a simplex with $n+1$ vertices, where n is the number of dimensions the function we optimize has. For convenience, we will use $m = n+1$. The simplex is then used to minimize the function based on a few basic steps: Each iteration the vertices are sorted by function values $f(x_i)$, x_1 being the point with the lowest value, and x_m the point with the largest value, and the centroid c of all except the worst point is computed: $c = \frac{1}{n}\sum_{i=1}^{n} x_i$. Then the vertices are moved according to the following rules:

1) **Reflection:** $x_r = c + \alpha(c - x_m)$. The worst point is reflected across the centroid. The point is accepted if $f(x_1) \leq f(x_r) < f(x_n)$. If $f(x_r) < f(x_1)$, we try expansion.
2) **Expansion:** $x_e = c + \gamma(x_r - c)$. The reflected point is pushed even further away from the centroid, since we believe that that direction is the direction of the largest descent. If $f(x_e) < f(x_r)$, we accept x_e, otherwise we accept x_r.
3) **Contraction:** There are two cases:
   - **Outside contraction:** Used when $f(x_n) \leq f(x_r) < f(x_m)$. Then $x_{oc} = c + \beta(x_r - c)$ is tested.
   - **Inside contraction:** Used when $f(x_r) \geq f(x_m)$. Then $x_{ic} = c - \beta(c - x_m)$ is tested.

   If the contracted point is better than x_m, we accept it. Otherwise, we shrink.
4) **Shrink:** All points except x_1 are replaced with $x_i = x_1 + \delta(x_i - x_1)$.

We used the standard parameter values: $\alpha = 1, \beta = 0.5, \gamma = 2, \delta = 0.5$.

Our implementation supports functions with arbitrary dimensions. We define a simplex class that stores the vertices and contains methods for the steps described above. The main loop simply applies the appropriate conditions until convergence. To detect convergence we rely on 2 different measures: The main stopping criteria is whether $f(x_1)$ doesn't change for more than 10^{-13} in ten iterations. Another criteria is the diameter of the simplex, however this value is harder to determine so we rely on the first condition.

B. Comparison with Gradient Descend methods

The Nelder-Mead method was compared to some gradient descend methods, specifically GD, Polyak GD, Nesterov GD, AdaGrad GD, Newton and BFGS, using 3 different functions. For each function, a vertex of the simplex had the same coordinates as the GD starting point. In the following sections we compare function values vs. iterations, total time and final result quality. The functions are the following:

1) $f_1(x,y,z) = (x-z)^2 + (2y+z)^2 + (4x-2y+z)^2 + x + y$
   Starting point: $(1,1,0)$.
2) $f_2(x,y,z) = (x-1)^2 + (y-1)^2 + 100(y-x^2)^2 + 100(z-y^2)^2$
   Starting point: $(-1, 1.2, 1.2)$.
3) $f_3(x,y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$
   Starting point: $(4.5, 4.5)$.

The initial simplexes for each function were the following:

1) For $f_1(x,y,z)$:
$$\begin{bmatrix} 1 & 1 & 0 \\ 2 & 1 & 0 \\ 1 & 2 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$
2) For $f_2(x,y,z)$:
$$\begin{bmatrix} -1.0 & 1.2 & 1.2 \\ 0.0 & 1.2 & 1.2 \\ -1.0 & 2.2 & 1.2 \\ -1.0 & 1.2 & 2.2 \end{bmatrix}$$
3) For $f_3(x,y)$:
$$\begin{bmatrix} 4.5 & 4.5 \\ 5.5 & 4.5 \\ 4.5 & 5.5 \end{bmatrix}$$

Below, we report the minimum value found and the corresponding input values for each function that were computed using SymPy.

- $f_1(x, y, z)$:
  - Minimum value -0.1979 @ $(-\frac{1}{6}, -\frac{11}{48}, \frac{1}{6})$
- $f_2(x, y, z)$:
  - Minimum value 0 @ (1, 1, 1)
- $f_3(x, y)$:
  - Minimum value 0 @ (3, 0.5)

The function values after 2, 5, 10, and 100 steps for each function and method are shown in Table I. Nelder–Mead typically requires more initial steps to approach the minimum or register improvement, but overall converges among the fastest, along with Newton and BFGS. After 100 steps, these methods consistently outperformed the others and Nelder-Mead has the additional advantage of being derivative-free. It also requires minimal hyperparameter tuning -— only the initial simplex – whereas gradient-based methods depend critically on the learning rate and momentum parameters (μ), making them more sensitive and harder to tune. Since Nelder–Mead accepts only non-worsening points by design, its monotonic convergence behavior likely contributes to its superior long-term performance.

Method	Steps	$f_1(x, y, z)$	$f_2(x, y, z)$	$f_3(x, y)$
f_{min}		**-0.1979**	**0.000**	**0.000**
GD	2	6.0553	8.8490	766.59
Polyak GD		5.8736	8.6296	**317.36**
Nesterov GD		5.9159	8.6664	337.89
AdaGrad		1.9098	10.8858	58009.33
Newton		**-0.1979**	9.0585	11938.21
BFGS		5.0201	4.2317	46320.01
Nelder-Mead		11.0000	13.7999	9416.95
GD	5	3.7295	5.8598	573.31
Polyak GD		3.4668	5.5520	234.18
Nesterov GD		3.4908	5.6033	251.26
AdaGrad		0.7652	4.2466	30478.91
Newton		**-0.1979**	4.2339	295.17
BFGS		0.6332	4.2138	3154.62
Nelder-Mead		1.6250	13.799	**54.25**
GD	10	1.9656	4.5401	391.01
Polyak GD		1.7119	4.4221	180.14
Nesterov GD		1.7301	4.4394	191.64
AdaGrad		0.0395	4.2304	16621.87
Newton		**-0.1979**	1141.01	14.59
BFGS		0.5838	4.2091	22.12
Nelder-Mead		0.6291	8.6600	**2.4658**
GD	100	-0.1466	4.2105	41.4483
Polyak GD		-0.1644	4.2099	27.3437
Nesterov GD		-0.1641	4.2099	28.1310
AdaGrad		**-0.1979**	4.1327	1315.53
Newton		**-0.1979**	$\mathbf{8.43}\times10^{-28}$	14.2031
BFGS		**-0.1979**	$\mathbf{9.77}\times10^{-29}$	572.69
Nelder-Mead		**-0.1979**	**2.8452**	$\mathbf{3.82}\times10^{-12}$

TABLE I

Results of different methods on the functions. For GD methods, the learning rate and μ were fixed at $0.01 \cdot 0.99^k$ and 0.1 respectively. For f_2 and f_3 the base learning rate was reduced by a factor of 100. For AdaGrad the learning rate was selected separately – for f_1 and f_2 it was set to 1 and for f_3 it was set to 0.2. The true minimum was computed using sympy and derivatives from the original homework.

An outlier is the Newton method, which converged in a single step for $f_1(x, y, z)$ – a special case since this function is quadratic. However, Newton's method is generally expensive due to the need to compute and invert the Hessian, which is why quasi-Newton methods like BFGS are often preferred in practice.

Method	$f_1(x, y, z)$	$f_2(x, y, z)$	$f_3(x, y)$
GD	6.406	5.891	6.738
Polyak GD	**5.652**	**5.322**	6.730
Nesterov GD	10.310	10.237	11.400
AdaGrad	16.158	15.669	16.224
Newton	26.088	22.719	25.665
BFGS	35.165	32.575	14.119
Nelder-Mead	6.552	18.007	**5.845**

TABLE II

Average execution times for 1000 steps for each method and function (10 repetitions).

Table II shows the average execution times for 1000 steps (each method was run 10 times) for each method and function. The method with fastest execution time is the Polyak GD, however as shown in Table I it does not converge the fastest. The Nelder-Mead method has the best ratio between computation speed and convergence, as the computation speed is close to that of Polyak, however, the performance is better, as Nelder-Mead consistently got closer to the minimum than Polyak in the same amount of steps. The other best performing methods, Newton and BFGS are significantly slower.

C. Black-box optimization

The black-box optimization was performed using Nelder-Mead method. We were given three 3-dimensional black-box functions, all guaranteed to be defined in the $[-10, 10]$ range for each parameter. The functions were optimized to the first 12 decimal places, after that the optimization was treated as converged. The initial simplex for the functions was constructed starting from the point $(-5, 0, 0)$, with the following vertices:

$$\begin{bmatrix} -5 & 0 & 0 \\ 5 & 0 & 0 \\ -5 & 10 & 0 \\ -5 & 0 & 10 \end{bmatrix}$$

These starting values were chosen, so that the initial simplex covered a large area of the $[-10, 10]^3$ domain.

Table III shows the minimum function values, minimum points and number of steps needed for convergence for all three functions. f_3 converged the fastest while f_1 needed the longest to converge. The reason for that might be that the initial vertices were closer to the minimum for f_2 and f_3 while relatively far for f_1, and f_1 might be symmetric/circular, making the simplex spin around it.

Figures 1 and 2 show the trajectories of x_1 vertex of the simplexes for each function (left) and the function values per iteration (right). Notice that f_1 appears to be significantly steeper than f_3, whose function values all move

Function	$f_{min}(x)$	x	y	z	Steps
f_1	0.603002360000	0.364460325392	0.023644608565	0.300236428268	150
f_2	21.952490173338	1.920214765663	1.799885616013	2.005920612167	42
f_3	1.017324218801	-0.235900878906	0.004577636719	5.467224121094	37

TABLE III
FUNCTION VALUES AND POINTS OF MINIMUMS FOR THE BLACK BOX FUNCTIONS.

around 1, whereas values for f_1 are orders of magnitude apart. The trajectories also show, that f_1 and f_2 shared the same first best vertex, while f_3 had a different one.

Since the function calls were slow, the Nelder-Mead method was optimized to minimize the number of function calls required for an iteration.

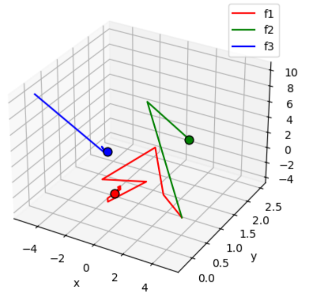

Fig. 1. Trajectory of x_1 simplex vertices in 3D.

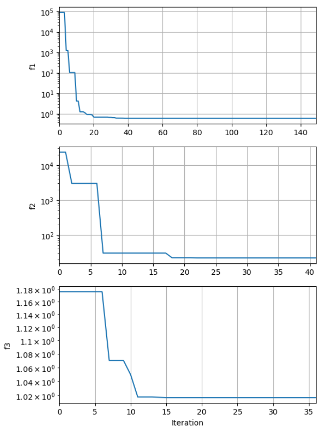

Fig. 2. Log-scaled function values per step.

Since the functions are black-boxes and do not provide explicit gradients, a gradient descent (GD) approach can still be approximated using numerical gradient estimation:

1) Choose an initial point $x_1 \in [-10, 10]^3$.
2) At each iteration k, approximate the gradient $\nabla f(x_k)$ using finite differences:
$$\frac{\partial f}{\partial x_i}(x_k) \approx \frac{f(x_k + he_i) - f(x_k)}{h}$$
where h is a small step size and e_i the unit vector along the i-th coordinate.
3) Update the iterate by a step in the negative gradient direction:
$$x_{k+1} = x_k - \alpha_k \nabla f(x_k)$$
where α_k is a suitable learning rate.

This approach however would be extremely slow, since the numerical gradient estimation requires multiple function evaluations per step.

D. *Jeklo Ruše Problem*

Lastly, we solved the Jeklo Ruše problem described at lectures. It is defined by the following system of equations:

$$\max_{x_1, x_2} \quad 3x_1 + 2x_2$$

subject to the constraints

$$\begin{cases} 3x_1 + x_2 \leq 600, \\ 2x_1 + 2x_2 \leq 480, \\ x_1 \geq 0, \\ x_2 \geq 0. \end{cases}$$

The feasible region is defined by the intersection of the half-planes given above. To solve the problem we find all intersection points of the constraint lines by solving pairs of equalities:

$$\begin{cases} 3x_1 + x_2 = b_1, \\ 2x_1 + 2x_2 = b_2, \end{cases}$$

where b_i are the right-hand side values from the constraints. For each pair, solve the linear system to find candidate vertices x.

Only points satisfying all inequalities (feasibility) are considered. The optimal solution is the feasible vertex maximizing the objective $3x_1 + 2x_2$. The optimal solution is $x_1 = 180$ and $x_2 = 60$.