# 1 Nelder-Mead method

In this problem, we compare the performance of the implemented Nelder-Mead method to performance of the methods from the previous homework, where we implemented gradient descent and Newton methods. In the previous homework, we used two starting points for each function. Here, we select the one for which the methods performed better. We run the Nelder-Mead method with multiple different diameter values.

1. The first function is

$$f_a(x, y, z) = (x - z)^2 + (2y + z)^2 + (4x - 2y + z)^2 + x + y.$$

The results of gradient descent and Newton methods are shown in Table 1. All methods converge after enough iterations. The Newton method converges first, already after 2 iterations, with BFGS close behind with 10 iterations.

In Table 2, we can see that Nelder-Mead does find the minimum, but the number of iterations needed depends on the simplex radius. If the radius is too large, the best simplex point needs more iterations to start updating. If the simplex is too small, it starts updating sooner but needs more iterations to move to the optimum. The optimal radius probably also depends on the distance to the minimum, which is $\mathbf{x}^* \approx [-0.17, -0.23, 0.17]^\mathsf{T}$ in this case.

| Stopping | **GD** | **Polyak** | **Nesterov** | **AdaGrad** | **Newton** | **BFGS** |
|---|---|---|---|---|---|---|
| $N = 2$ | $-0.035$ | $-0.037$ | $-0.037$ | $-0.032$ | $-0.198$ | 30.3 |
| $N = 5$ | $-0.073$ | $-0.077$ | $-0.077$ | $-0.056$ | $-0.198$ | 47.6 |
| $N = 10$ | $-0.113$ | $-0.119$ | $-0.119$ | $-0.082$ | $-0.198$ | $-0.198$ |
| $N = 100$ | $-0.198$ | $-0.198$ | $-0.198$ | $-0.180$ | $-0.198$ | $-0.198$ |
| $t = 0.1$ s | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ |
| $t = 1$ s | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ |
| $t = 2$ s | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ |

Table 1: Minimal function values, obtained by running gradient descent and Newton methods on $f_a$ with $\mathbf{x}_1 = [0, 0, 0]^\mathsf{T}$.

| Stopping | $r = 0.001$ | $r = 0.01$ | $r = 0.02$ | $r = 0.05$ | $r = 0.1$ | $r = 0.5$ |
|---|---|---|---|---|---|---|
| $N = 2$ | $-0.003$ | $-0.023$ | $-0.043$ | $-0.096$ | $-0.075$ | 0.000 |
| $N = 5$ | $-0.010$ | $-0.071$ | $-0.064$ | $-0.114$ | $-0.168$ | 0.000 |
| $N = 10$ | $-0.049$ | $-0.167$ | $-0.196$ | $-0.195$ | $-0.191$ | $-0.172$ |
| $N = 100$ | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ |
| $t = 0.1$ s | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ |
| $t = 1$ s | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ |
| $t = 2$ s | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ | $-0.198$ |

Table 2: Minimal function values, obtained using the Nelder-Mead method with different radii on $f_a$ with $\mathbf{x}_1 = [0, 0, 0]^\mathsf{T}$.

2. The second function is

$$f_b(x, y, z) = (x-1)^2 + (y-1)^2 + 100(y-x^2)^2 + 100(z-y^2)^2.$$

In Table 3, we can see that the Newton method again works the best. Here, BFGS diverges with too few iterations. Other gradient descent methods also find the minimum but need additional iterations.

The results of the Nelder-Mead method are shown in Table 4. Again, the performance depends on the radius. Even though the optimum is at $\mathbf{x}^* = [1, 1, 1]^\mathsf{T}$, the performance drops for $r > 0.1$. In all cases, the minimum is found in 100 iterations. For $r = 0.05$ and 0.1, the method already comes close after 10 iterations.

| Stopping | **GD** | **Polyak** | **Nesterov** | **AdaGrad** | **Newton** | **BFGS** |
|---|---|---|---|---|---|---|
| $N = 2$ | 8.091 | 7.926 | 7.941 | 8.155 | 0.035 | / |
| $N = 5$ | 4.814 | 4.416 | 4.454 | 5.941 | 0.000 | / |
| $N = 10$ | 2.113 | 1.755 | 1.787 | 4.056 | 0.000 | / |
| $N = 100$ | 0.018 | 0.018 | 0.018 | 0.102 | 0.000 | / |
| $t = 0.1$ s | 0.0002 | 0.0007 | 0.0009 | 0.0076 | 0.0000 | 0.0000 |
| $t = 1$ s | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| $t = 2$ s | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |

Table 3: Minimal function values, obtained by running gradient descent and Newton methods on $f_b$ with $\mathbf{x}_1 = [1.2, 1.2, 1.2]^\mathsf{T}$.

| Stopping | $r = 0.001$ | $r = 0.01$ | $r = 0.02$ | $r = 0.05$ | $r = 0.1$ | $r = 0.5$ |
|---|---|---|---|---|---|---|
| $N = 2$ | 11.209 | 8.011 | 5.123 | 3.619 | 1.999 | 11.600 |
| $N = 5$ | 9.884 | 0.821 | 0.345 | 0.062 | 0.256 | 1.781 |
| $N = 10$ | 1.717 | 0.816 | 0.345 | 0.051 | 0.082 | 0.456 |
| $N = 100$ | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| $t = 0.1$ s | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| $t = 1$ s | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| $t = 2$ s | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |

Table 4: Minimal function values, obtained using the Nelder-Mead method with different radii on $f_b$ with $\mathbf{x}_1 = [1.2, 1.2, 1.2]^\mathsf{T}$.

3. The last function is

$$f_c(x, y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2.$$

Table 5 shows that the Newton methods fail to find the minimum. The gradient descent methods do find it but need more than 100 iterations. AdaGrad even needs 1 s to find it.

The Nelder-Mead method works better, see Table 6. The minimum is found in 100 iterations or even less for optimal radius value. Since $\mathbf{x}^* = [3, 0.5]^\mathsf{T}$, the larger radius values work better, with optimal being around $r = 2$.

| Stopping | **GD** | **Polyak** | **Nesterov** | **AdaGrad** | **Newton** | **BFGS** |
|---|---|---|---|---|---|---|
| $N = 2$ | 6.75 | 6.49 | 6.56 | 13.7 | 2.15 | / |
| $N = 5$ | 3.97 | 3.66 | 3.70 | 13.3 | 2.49 | / |
| $N = 10$ | 2.04 | 1.82 | 1.83 | 12.8 | 1332.2 | / |
| $N = 100$ | 0.08 | 0.07 | 0.07 | 9.3 | 9.9 | / |
| $t = 0.1$ s | 0.000 | 0.000 | 0.000 | 0.061 | 9.9 | / |
| $t = 1$ s | 0.000 | 0.000 | 0.000 | 0.000 | 9.9 | / |
| $t = 2$ s | 0.000 | 0.000 | 0.000 | 0.000 | 9.9 | / |

Table 5: Minimal function values, obtained by running gradient descent and Newton methods on $f_c$ with $\mathbf{x}_1 = [1, 1]^\mathsf{T}$.

| Stopping | $r = 0.05$ | $r = 0.1$ | $r = 0.5$ | $r = 1.0$ | $r = 2.0$ | $r = 3.0$ |
|---|---|---|---|---|---|---|
| $N = 2$ | 11.570 | 9.243 | 1.078 | 0.703 | 1.078 | 11.013 |
| $N = 5$ | 1.916 | 1.463 | 1.078 | 0.703 | 1.078 | 0.238 |
| $N = 10$ | 1.194 | 0.616 | 0.159 | 0.045 | 0.010 | 0.036 |
| $N = 100$ | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| $t = 0.1$ s | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| $t = 1$ s | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| $t = 2$ s | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |

Table 6: Minimal function values, obtained using the Nelder-Mead method with different radii on $f_c$ with $\mathbf{x}_1 = [1, 1]^\mathsf{T}$.

# 2 Black box optimization

## 2.1 Nelder-Mead

For each of the three functions, we look for the minimum with an initial guess $\mathbf{x}_1 = [0, 0, 0]^\mathsf{T}$. We run the Nelder-Mead method until the difference between function values of the best and the worst point is less than $1e^{-11}$. The results of optimization for all functions are shown in Table 7. We were able to find the minimum for all functions in around 10 minutes. The optimization for the second function takes the most iterations. This function also happened to be the most difficult to optimize using gradient descent.

| $i$ | $f_i(\mathbf{x}^*)$ | $\mathbf{x}^*$ | Iterations | Time [min:s] |
|---|---|---|---|---|
| 1 | 0.790091360 | [0.36789946, 0.91367979, 0.00913816] | 81 | 10:33 |
| 2 | 0.790091360 | [0.00913924, 0.91367866, 0.36790025] | 103 | 11:32 |
| 3 | 0.790091360 | [0.91367968, 0.00913563, 0.36789706] | 71 | 9:37 |

Table 7: Results of black-box optimization using the Nelder-Mead method.

The benefit of this method was that the optimization was not highly dependent on the chosen radius parameter. In contrast, using gradient descent was more difficult because an incorrect learning rate quickly caused the function values to explode.

## 2.2 Gradient descent

To use gradient descent methods, we need a gradient but we cannot compute it analytically because we do not have the function expression. Instead, we use the forward difference approximation:

$$\nabla f(\mathbf{x}) \approx \frac{1}{h} \begin{bmatrix} f(x_1 + h, x_2, x_3) - f(\mathbf{x}) \\ f(x_1, x_2 + h, x_3) - f(\mathbf{x}) \\ f(x_1, x_2, x_3 + h) - f(\mathbf{x}) \end{bmatrix},$$

for $h = 1e^{-8}$. We use the forward difference instead of the centered difference because we decrease the number of function evaluations by half.

To find the best method, we compare *gradient descent*, *AdaGrad*, *Nesterov gradient descent* and *RMSProp*.

For the first function, we already managed to find the minimum using the ordinary gradient descent with a learning rate of 0.1 in 54 iterations. Each iteration took a few seconds more than the Nelder-Mead one, but the minimum was found in 9 minutes due to fewer iterations.

For the second function, none of the methods managed to find the minimum in a reasonable time. The convergence nearly stopped after 20 minutes. Nesterov was also not able to improve the convergence speed because the function value exploded for all but very small momentum values around $1e^{-8}$.

For the third function, we again managed to find the minimum. This time, AdaGrad performed well with a learning rate of 1. It found the minimum in only 36 iterations, which took around 7 minutes and a half.