1 Theoretical problems

1.1 Convexity proposition

The proposition 2.3 says that the following two statements are equivalent:

1. f is convex.
2. For each $x_1, x_2, \ldots, x_k \in D$ and $\alpha_1, \alpha_2, \ldots, \alpha_k \in [0,1]$ satisfying $\sum_{i=1}^{k} \alpha_i = 1$ we have
$$f\left(\sum_{i=1}^{k} \alpha_i x_i\right) \leq \sum_{i=1}^{k} \alpha_i f(x_i). \tag{1}$$

We prove this equivalence by proving the implications in both directions.
($\Longrightarrow$) Assuming that f is convex, it holds that $\forall x_i, x_j \in D$ for $i, j \in \{1, 2, \ldots, k\}$ and $t \in [0,1]$:
$$f(tx_i + (1-t)x_j) \leq tf(x_i) + (1-t)f(x_j). \tag{2}$$
The base case for the induction proof is for $k = 2$:

$$\begin{aligned} f(\alpha_1 x_1 + \alpha_2 + x_2) &= f(\alpha_1 x_1 + (1-\alpha_1)x_2) \\ &\leq \alpha_1 f(x_1) + (1-\alpha_1)f(x_2) = \alpha_1 f(x_1) + \alpha_2 f(x_2). \end{aligned}$$

Let (1) hold for $k-1$. The inductive step equals:

$$\begin{aligned} f\left(\sum_{i=1}^{k} \alpha_i x_i\right) = f\left(\sum_{i=1}^{k-1} \alpha_i x_i + \alpha_k x_k\right) &= f\left((1-\alpha_k)\frac{\sum_{i=1}^{k-1} \alpha_i x_i}{1-\alpha_k} + \alpha_k x_k\right) \\ &\leq (1-\alpha_k) f\left(\sum_{i=1}^{k-1} \frac{\alpha_i}{1-\alpha_k} x_i\right) + \alpha_k f(x_k) \\ &\leq (1-\alpha_k) \sum_{i=1}^{k-1} \frac{\alpha_i}{1-\alpha_k} f(x_i) + \alpha_k f(x_k) = \sum_{i=1}^{k} \alpha_i f(x_i). \end{aligned}$$

We were able to use (2) because $\sum_{i=1}^{k-1} \frac{\alpha_i}{1-\alpha_k} = \frac{1}{1-\alpha_k} \sum_{i=1}^{k-1} \alpha_i = 1$. This shows that (1) holds for all $k \geq 2$.
($\Longleftarrow$) Proof in the other direction is trivial by setting $k = 2$, $t = \alpha_1$ and observing $1 - t = \alpha_2$:

$$\begin{aligned} f(tx_1 + (1-t)x_2) &= f(\alpha_1 x_1 + \alpha_2 x_2) \\ &\leq \alpha_1 f(x_1) + \alpha_2 f(x_2) = tf(x_1) + (1-t)f(x_2). \end{aligned}$$

■

1.2 Lipschitz, smooth and strongly convex

First, we compute the gradient of our function:

$$\nabla f(x,y) = \begin{bmatrix} 2x + e^x - y \\ 2y - x \end{bmatrix}.$$

To find the constant L we need to bound the norm of the gradient. We work with its square for easier computation:

$$\|\nabla f(x,y)\|^2 = f_x^2 + f_y^2 = 5x^2 + 5y^2 - 8xy + 4xe^x - 2ye^x + e^{2x}.$$

To find the maximum, we solve $\nabla\|\nabla f(x,y)\|^2 = \mathbf{0}$, where

$$\nabla\|\nabla f(x,y)\|^2 = \begin{bmatrix} 10x - 8y + 4e^x + 4xe^x - 2ye^x + 2e^{2x} \\ 10y - 8x - 2e^x \end{bmatrix}.$$

Using symbolic computation Python package `SymPy`, we obtain one real solution:

$$x_1 \doteq -0.43, \quad y_1 \doteq -0.22,$$

but even though $(x_1, y_1) \in K$, this is a local minimum because

$$\nabla^2\|\nabla f(x,y)\|^2 = \begin{bmatrix} 10 + 8e^x + 4xe^x - 2ye^x + 4e^{2x} & -8 - 2e^x \\ -8 - 2e^x & 10 \end{bmatrix}$$

and

$$\nabla^2\|\nabla f(x,y)\|^2\big|_{(x_1,y_1)} \doteq \begin{bmatrix} 16.03 & -9.30 \\ -9.30 & 10 \end{bmatrix} \succ 0$$

by the Sylvester's criterion. This means that the maximum of the norm is attained on the domain boundary. The left boundary equals

$$\|\nabla f(-2,y)\|^2 = 5y^2 + y(16 - 2e^{-2}) + 20 + e^{-4} - 8e^{-2},$$

and the right one:

$$\|\nabla f(2,y)\|^2 = 5y^2 + y(-16 - 2e^2) + 20 + e^4 + 8e^2.$$

Both of these are quadratics with positive leading coefficients and their maximum is one of the boundary points. The top boundary equals

$$\|\nabla f(x,2)\|^2 = 5x^2 + 20 - 16x + 4xe^x - 4e^x + e^{2x},$$

and the bottom one:

$$\|\nabla f(x,-2)\|^2 = 5x^2 + 20 + 16x + 4xe^x + 4e^x + e^{2x}.$$

We can see that only two terms differ. All other terms are always positive. Terms $16x$ and $-16x$ are symmetric, but $4e^x$ from the bottom boundary is always positive, so its maximum value is higher. We can maximise its value by maximising x, which is achieved at the boundary. We can conclude that the maximum norm of the gradient will be attained in one of the corners of the domain.

Let us compute the gradient norms in the corners:

$$\begin{aligned}\|\nabla f(-2,-2)\| &\doteq 2.73,\\ \|\nabla f(2,-2)\| &\doteq 14.67,\\ \|\nabla f(-2,2)\| &\doteq 8.39,\\ \|\nabla f(2,2)\| &\doteq 9.60.\end{aligned}$$

This means that

$$L = \|\nabla f(2,-2)\| = \sqrt{72 + 12e^2 + e^4}.$$

To compute the constants α and β, we need to find the smallest and the largest eigenvalue of the Hessian, which equals

$$\nabla^2 f(x,y) = \begin{bmatrix} 2+e^x & -1 \\ -1 & 2 \end{bmatrix}.$$

Its characteristic polynomial is

$$\det\big(\nabla^2 f(x,y) - \lambda \mathbf{I}\big) = (2+e^x-\lambda)(2-\lambda) - 1 = \lambda^2 + \lambda(-4-e^x) + 3 + 2e^x.$$

By the quadratic formula, we compute the eigenvalues as

$$\lambda_{1,2}(x) = \frac{4 + e^x \pm \sqrt{4+e^{2x}}}{2}.$$

We can check graphically that both of these functions are increasing on $[-2,2]$. Thus, we can just check the boundaries again:

$$\begin{aligned}\lambda_1(-2) &\doteq 3.07, \quad \lambda_1(2) \doteq 9.52,\\ \lambda_2(-2) &\doteq 1.07, \quad \lambda_2(2) \doteq 1.87.\end{aligned}$$

It follows that

$$\alpha = \lambda_2(-2) = \frac{4 + e^{-2} - \sqrt{4+e^{-4}}}{2}$$

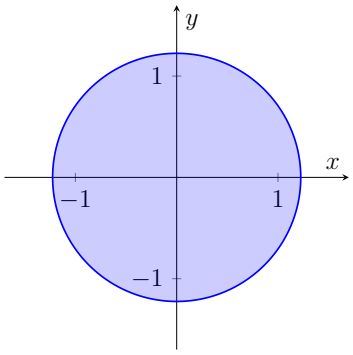
and

$$\beta = \lambda_1(2) = \frac{4 + e^2 + \sqrt{4+e^4}}{2}.$$

Because $\alpha > 0$, it also follows that f is convex.

1.3 Projections to closest point

To project to the closest point in the circle $x^2 + y^2 \leq 1.5$, we normalise outside vectors to unit length and multiply them with the radius:



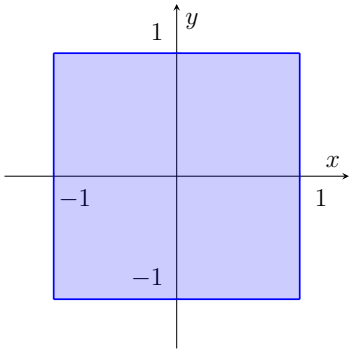
$$f_1(\mathbf{x}) = \begin{cases} \mathbf{x}, & \|\mathbf{x}\|^2 \leq 1.5, \\ \frac{\mathbf{x}\sqrt{1.5}}{\|\mathbf{x}\|}, & \text{else.} \end{cases}$$

To project to the square domain $[-1, 1] \times [-1, 1]$, we clip both coordinates to the interval $[-1, 1]$:

$$f_2(x, y) = (g(x, -1, 1), g(y, -1, 1)),$$

where g is the clip function:

$$g(x, a, b) = \min\{\max\{x, a\}, b\}.$$

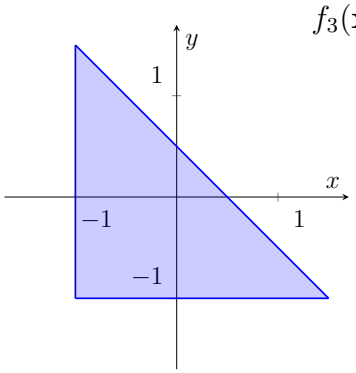


Finally, to project points to the triangle with vertices $(-1, -1)$, $(1.5, -1)$ and $(-1, 1.5)$, we first project vertices above the hypotenuse to the line $y = 0.5 - x$ and then clip the coordinates componentwise like with the previous domain. We define the line vector $\mathbf{s} = [1, -1]^\mathsf{T}$, the offset vector $\mathbf{o} = \left[\frac{1}{4}, \frac{1}{4}\right]^\mathsf{T}$ and the line normal vector $\mathbf{n} = [1, 1]^\mathsf{T}$. The line projection equals:

$$h(\mathbf{x}) = \frac{\mathbf{x}^\mathsf{T}\mathbf{s}}{\mathbf{s}^\mathsf{T}\mathbf{s}}\mathbf{s} + \mathbf{o},$$

and the final projection:

$$f_3(\mathbf{x}) = \begin{cases} g(h(\mathbf{x}), -1, 1.5), & (\mathbf{x} - \mathbf{o})^\mathsf{T}\mathbf{n} > 0. \\ g(\mathbf{x}, -1, 1.5), & \text{else.} \end{cases}$$



1.4 Gradient descent

Let us first compute the gradient of the function,

$$\nabla f(x, y) = \begin{bmatrix} 2x \\ 4y \end{bmatrix},$$

and its value in the initial point:

$$\nabla f(\mathbf{x}_1) = \begin{bmatrix} 2 \\ 4 \end{bmatrix}.$$

From the gradient descent update rule, we obtain the formula for the second point:

$$\mathbf{x}_2 = \mathbf{x}_1 - \gamma \nabla f(\mathbf{x}_1) = \begin{bmatrix} 1 - 2\gamma \\ 1 - 4\gamma \end{bmatrix},$$

where γ is the learning rate.

1. To compute the minimal function value that can be achieved with one step, we compute the value

   $$f(\mathbf{x}_2) = (1 - 2\gamma)^2 + 2(1 - 4\gamma)^2 = 36\gamma^2 - 20\gamma + 3,$$

   and its derivative:

   $$\frac{d}{d\gamma} f(\mathbf{x}_2) = 72\gamma - 20.$$

   It equals 0 when $\gamma = \frac{5}{18}$, from which it follows that

   $$\min f(\mathbf{x}_2) = \frac{2}{9}.$$

   The extremum is the minimum because the leading coefficient of the quadratic is positive.

2. We know that the minimum equals $\mathbf{x}^* = \mathbf{0}$ since f is a paraboloid centred in the coordinate origin. The squared distance after one step is

   $$\|\mathbf{x}_2 - \mathbf{x}^*\|^2 = \|\mathbf{x}_2\|^2 = (1 - 2\gamma)^2 + (1 - 4\gamma)^2 = 20\gamma^2 - 12\gamma + 2,$$

   and its derivative:

   $$\frac{d}{d\gamma} \|\mathbf{x}_2\|^2 = 40\gamma - 12.$$

   Here, it equals 0 when $\gamma = 0.3$ and we get that the minimum distance is

   $$\min \|\mathbf{x}_2\| = \sqrt{0.2}.$$

2 Programming problems

2.1 Projected gradient descent

We implemented the PGD algorithm in Python. Similarly to the GD update, we move in the opposite direction of the gradient, but additionally project the new estimate to our domain.

```
def projected_gradient_descent(x_1, gradient, learning_rate, domain, steps):
    """Perform steps of the projected gradient descent algorithm."""
    lr, adaptive, *_ = learning_rate
    xs = [x_1]
    for i in range(steps):
        if adaptive:
            lr = learning_rate[0] / (i + 2)
        x = domain(xs[-1] - lr * gradient(xs[-1]))
        xs.append(x)
    return np.array(xs)
```

Because we know our function is Lipschitz, smooth and strongly convex, we can use the computed constants L, α and β to calculate the optimal learning rates from Theorem 3.3:

$$\begin{aligned}\gamma_1 &= \frac{\|\mathbf{x}_1 - \mathbf{x}^*\|}{L\sqrt{T}} \doteq 0.028,\\ \gamma_2 &= \frac{1}{\beta} \doteq 0.105,\\ \gamma_3^{(k)} &= \frac{2}{\alpha(k+1)} \in [0.171, 0.939].\end{aligned}$$

We are doing 10 steps, so $T = 11$, and in the last iteration $k = 10$. The true minimum is in the same location as the minimum of the norm we obtained in Problem 2. Symbolically we obtain the approximation

$$\mathbf{x}^* \doteq [-0.43, -0.22]^\mathsf{T}.$$

Each learning rate from the theorem gives us an upper bound on some expression of the function values obtained during iterations:

$$\begin{aligned}LHS_1 = f\left(\frac{1}{T}\sum_{i=1}^{T}\mathbf{x}_i\right) - f(\mathbf{x}^*) &\leq \frac{L\|\mathbf{x}_1 - \mathbf{x}^*\|}{\sqrt{T}} = RHS_1,\\ LHS_2 = f(\mathbf{x}_{k-1}) - f(\mathbf{x}^*) &\leq \frac{\beta}{2}\left(\frac{\beta-\alpha}{\beta}\right)^{2k}\|\mathbf{x}_1 - \mathbf{x}^*\|^2 = RHS_2,\\ LHS_3 = f\left(\sum_{i=1}^{T}\frac{2i}{T(T+1)}\mathbf{x}_i\right) - f(\mathbf{x}^*) &\leq \frac{2L^2}{\alpha(T+1)} = RHS_3.\end{aligned}$$

The results for the three domains we used are shown in Tables 1, 2 and 3. For the same learning rate, they are similar across different domains, but we can see that the circular domain performs a small amount better for γ_1 and γ_3. In terms of getting close to the actual minimum, the last learning rate is the best for all domains. Again, the circular domain is the best overall. This is probably due to the fact that this learning rate decreases over iterations.
Even though the last learning rate is the best, the upper bound is not the tightest, due to the large value of L which gets squared. The tightest bound is for γ_2, which does not perform as good as γ_3, but better than γ_1.

Learning rate	$\mathbf{x}_{11}$	$\|\mathbf{x}_{11} - \mathbf{x}^*\|$	$\|f(\mathbf{x}_{11}) - f(\mathbf{x}^*)\|$	RHS_i	LHS_i
γ_1	$[-0.51, 0.38]$	0.604928	0.411721	1.032280	5.937289
γ_2	$[-0.36, -0.08]$	0.152882	0.015181	0.015181	0.799212
γ_3	$[-0.43, -0.22]$	0.002572	0.000004	0.000063	33.676038

Table 1: Projected gradient descent results with the domain defined by the circle $x^2 + y^2 \leq 1.5$.

Learning rate	$\mathbf{x}_{11}$	$\|\mathbf{x}_{11} - \mathbf{x}^*\|$	$\|f(\mathbf{x}_{11}) - f(\mathbf{x}^*)\|$	RHS_i	LHS_i
γ_1	$[-0.53, 0.41]$	0.633818	0.464623	1.157913	5.937289
γ_2	$[-0.36, -0.08]$	0.152882	0.015181	0.015181	0.799212
γ_3	$[-0.43, -0.21]$	0.008311	0.000044	0.003361	33.676038

Table 2: Projected gradient descent results with the domain defined by the square $[-1, 1] \times [-1, 1]$.

Learning rate	$\mathbf{x}_{11}$	$\|\mathbf{x}_{11} - \mathbf{x}^*\|$	$\|f(\mathbf{x}_{11}) - f(\mathbf{x}^*)\|$	RHS_i	LHS_i
γ_1	$[-0.53, 0.41]$	0.633818	0.464623	1.157913	5.937289
γ_2	$[-0.36, -0.08]$	0.152882	0.015181	0.015181	0.799212
γ_3	$[-0.43, -0.21]$	0.011943	0.000091	0.007306	33.676038

Table 3: Projected gradient descent results with the domain defined by the triangle with vertices $(-1, -1)$, $(1.5, -1)$, $(-1, 1.5)$.

All iteration steps can be seen in Figure 1. We can observe that the first learning rate is too small, the second one works fine, but the last one is the best. Even though it oscillates, it gets closer to the minimum because it gets smaller over time.

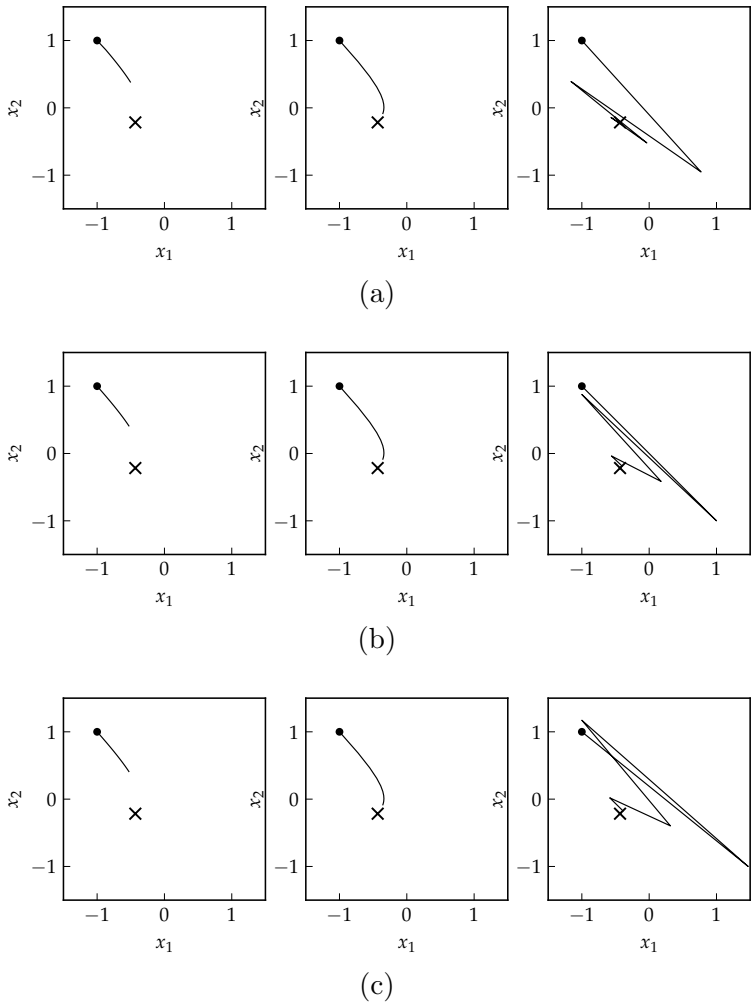


Figure 1: Projected gradient descent process for (a) circle, (b) square and (c) triangle domains, with (left) γ_1, (middle) γ_2 and (left) γ_3. The initial point is marked with a dot, and the optimal one with a cross.

2.2 Global minimum

To be able to use the gradient descent method, we first need to compute the gradient of our function:

$$\frac{\partial f}{\partial z_j} = -\sum_{i=1}^{4} c_i e^k \frac{\partial k}{\partial z_j} = \sum_{i=1}^{4} c_i e^k \cdot 2a_{ij}(z_j - p_{ij}),$$

where

$$k = -\sum_{l=1}^{3} a_{il}(z_l - p_{il})^2.$$

We implement the gradient in the code and use implementation from the previous problem while defining a new domain $[0, 1]^3$.

```
def grad(x):
    exponents = -np.sum(a * (x - p) ** 2, axis=1)
    exp_grad = -2 * a * (x - p)
    return -np.sum((c * np.exp(exponents)).reshape(-1, 1) * exp_grad, axis=0)


def project(x):
    return np.clip(x, 0, 1)
```

First, we select the best learning rate for $\mathbf{x}_1 = [0.5, 0.5, 0.5]^\mathsf{T}$ using 100 000 steps. The best learning rate was $\gamma = 0.01$. We also do a grid search to find the best initial point and its learning rate. We run the gradient descent for both configurations and results are shown in Table 4.

Steps N	$\mathbf{x}_1$	$f(\mathbf{x}_1)$	$\mathbf{x}_{N+1}$	$f(\mathbf{x}_{N+1})$
10	$[0.5, 0.5, 0.5]^\mathsf{T}$	-0.628	$[0.41878, 0.55501, 0.83658]^\mathsf{T}$	-3.78259038157
100	$[0.5, 0.5, 0.5]^\mathsf{T}$	-0.628	$[0.22196, 0.55518, 0.85241]^\mathsf{T}$	-3.85586422336
1000	$[0.5, 0.5, 0.5]^\mathsf{T}$	-0.628	$[0.11461, 0.55564, 0.85254]^\mathsf{T}$	-3.86278214781
10	$[0.1, 0.55, 0.85]^\mathsf{T}$	-3.861	$[0.10153, 0.55561, 0.85382]^\mathsf{T}$	-3.86252735338
100	$[0.1, 0.55, 0.85]^\mathsf{T}$	-3.861	$[0.11024, 0.55565, 0.85256]^\mathsf{T}$	-3.86277057779
1000	$[0.1, 0.55, 0.85]^\mathsf{T}$	-3.861	$[0.11461, 0.55564, 0.85254]^\mathsf{T}$	-3.86278214782

Table 4: Results of projected gradient descent for two initial points and multiple different numbers of steps.

After 1000 steps we obtain a minimum function value accurate to 10 decimal places. We get better results using a better initial point, but the difference is minimal for $N > 1000$.

We can conclude that the minimum of this function is achieved close to the point

$$\mathbf{x} \doteq [0.11461, 0.55564, 0.85254]^\mathsf{T},$$

and its value is

$$f(\mathbf{x}) \doteq -3.86278214782.$$