LINEAR PROGRAMMING PROJECT

GROUP 2 REPORT FOR OPTIMIZATION F24 COURSE

Dmitry Beresnev
AIDS-MS1, Innopolis University
d.beresnev@innopolis.university

Vsevolod Klyushev
AIDS-MS1, Innopolis University
v.klyushev@innopolis.university

1 Introduction

Initial problem is formulated as following:

$$\begin{aligned} &\min_{x' \in \mathbb{R}^p} \|Ax' - y'\|_1 \\ &\text{s.t. } 0 \leq x' \leq 1 \end{aligned} \tag{1}$$

where $A \in \mathbb{R}^{m \times p}$ with $m \geq p$ — message encoding matrix, y' — received encoded (noisy) message, x' — encoded initial message to be find.

2 Notations

Notation	Meaning
e_i	unit vector with 1 at index i and all other zeroes
1_n	vector of n ones
I_n	identity matrix of size $n \times n$
0_n	vector of n zeroes
$0_{m \times n}$	zero matrix of size $m \times n$
x_i (or $(Ax)_i$)	i-th component of vector x (or Ax)

3 Q1: Linear problem formulation

Initial problem (eq. (1)) is not linear as cost function $\|Ax' - y'\|_1 = \sum_{i=1}^{m} |(Ax')_i - y'_i|$, is not linear. However, this objective function is **piecewise linear convex** function. Therefore, each element $|(Ax')_i - y'_i| = \max((Ax')_i - y'_i, y'_i - (Ax')_i)$ can be substituted with new variable z' with the following additional constraints: $z_i \geq (Ax')_i - y'_i$ and $z_i \geq y'_i - (Ax')_i$.

So the following problem is **linear** and is equivalent to the initial one:

$$\begin{aligned} &\min_{x' \in \mathbb{R}^p, z \in \mathbb{R}^m} \sum_{i=1}^{m} z_i \\ &\text{s.t. } x' \geq 0 \\ &\quad x' \leq 1 \\ &\quad z_i \geq (Ax')_i - y'_i,\ i = 1 \dots m \\ &\quad z_i \geq y'_i - (Ax')_i,\ i = 1 \dots m \end{aligned} \tag{2}$$

4 Q2: Linear problem in standard form

For the easier and more evident derivation of standard form of Equation (2), linear problem will be firstly rewritten in geometric form, and only then — in standard. The obtained linear optimization problem in standard form will be equivalent to initial problem (eq. (1)).

4.1 Geometric form

The equivalent **geometric** form of Equation (2) is

$$\begin{aligned} &\min_{z' \in \mathbb{R}^{p+m}} c^T z' \\ &\text{s.t.} \underbrace{\left(\begin{array}{c:c} I_p & 0_{p\times m} \\ \hdashline -I_p & 0_{p\times m} \\ \hdashline -A & I_m \\ \hdashline A & I_m \end{array}\right)}_{A'} z' \geq \underbrace{\begin{pmatrix} 0_p \\ -1_p \\ -y' \\ y' \end{pmatrix}}_{b'}, \end{aligned} \tag{3}$$

where $c = \sum_{i=p+1}^{p+m} e_i \in \mathbb{R}^{(p+m)}$, $b' \in \mathbb{R}^{2p+2m}$ and $A' \in \mathbb{R}^{(2p+2m)\times(p+m)}$.

The first p components of z' correspond to the components of x', and the next m components correspond to the components of z from Equation (2). Rows and columns of A' representation in Equation (3) are separated in blocks for clarity: the vertical separation is for x' and z correspondingly, and the horizontal separations denote corresponding constraints from Equation (2).

4.2 Standard form

Note that $z' = (x', z)^T$ from Equation (3) is already non-negative, because $x' \geq 0$ by problem definition and $z \geq 0$ by construction[1]. Therefore, to convert Equation (3) to standard form, only introduction of slack variables is needed to get rid of inequality sign. The equivalent **standard** form of Equation (3) is

$$\begin{aligned} &\min_{\tilde{x} \in \mathbb{R}^{2p+3m}} c^T \tilde{x} \\ &\text{s.t.} \underbrace{\left(\begin{array}{c:c:c} -I_p & 0_{p\times m} & \\ \hdashline -A & I_m & -I_{p+2m} \\ \hdashline A & I_m & \end{array}\right)}_{A'} \tilde{x} = \underbrace{\begin{pmatrix} -1_p \\ -y' \\ y' \end{pmatrix}}_{b'}, \\ &\tilde{x} \geq 0, \end{aligned} \tag{4}$$

where $c = \sum_{i=p+1}^{p+m} e_i \in \mathbb{R}^{(2p+3m)}$, $b' \in \mathbb{R}^{p+2m}$ and $A' \in \mathbb{R}^{(p+2m)\times(2p+3m)}$. Here, $-I_{p+2m}$ represents necessary slack variables.

The first p components of $\tilde{x}$ correspond to the components of x', the next m components correspond to the components of z from Equation (2) and the last $(p+2m)$ components correspond to slack variables s. Rows and columns of A' representation in Equation (4) are again separated in blocks for clarity: the vertical separations are for x', z and s correspondingly, and the horizontal separations are related to corresponding constraints from Equation (3) (except first one, as non-negativity in standard from is separate constraint).

[1] Intuitively, z substitutes the absolute value, so is non-negative. Formally, from Equation (2), $z \geq t$ and $z \geq -t$ for some t. So if $t \geq 0$, then $z \geq t \geq 0$, and if $t \leq 0$ then $z \geq -t \geq 0$

5 Q3: Message decryption

In the sake of research interest, the message decrypted by solving three different problems: the least squares (assuming no noise at all), linear optimization program (LOP) in geometric form and linear optimization problem in standard form.

5.1 Least Squares

As it is previously mentioned, this approach is quite naive as assumes no noise in received signal y'. The function *scipy.linalg.lstsq* is used, so the encoded message is found as a solution to the problem $\min_{x' \in \mathbb{R}^p} \|Ax' - y'\|_2$. The resulting function is demonstrated on Listing 1.

5.2 LOP in geometric form

In this case, the encoded message is found as a solution to the problem Equation (3) using the function *scipy.optimize.linprog*. The resulting function is demonstrated on Listing 2. As one can notice, the construction of A and b completely reflects representations of A' and b' from Equation (3). Also, as was mentioned in Section 4.1, we are interested only in the first p components of the solution.

5.3 LOP in standard form

The encoded message is found as a solution to the problem Equation (4) using the function *scipy.optimize.linprog*. The resulting function is demonstrated on Listing 3. As one can notice, the construction of A and b completely reflects representations of A' and b' from Equation (4). Also, as was mentioned in Section 4.2, we are interested only in the first p components of the solution.

5.4 Results

As expected, result of naive solution is a total mess, while the solution of LOP of both forms led to meaningful message:

You can claim your personal reward by going to Student affairs, giving you code=1083 and ask for you reward

Also note that computation time of standard LOP (334 seconds) is 14% greater than of geometric LOP (293 seconds)[2].

6 Q4: Check if the obtained solution is a polyhedron vertex

In this section the linear program in geometric form is considered, as the verifying that the point is a vertex of polyhedron in geometric form is easier than in case of polyhedron in standard form.

For polyhedron in geometric form, the following is true: for the feasible solution $x \in \mathbb{R}^n$, if polyhedron has n linearly independent constraints active (or tight) at x, then x — vertex of this polyhedron. For polyhedron defined in Equation (3), it is needed to verify that there are $(p + m)$ tight linearly independent constraints $A'z^* \geq b'$, where $z^* \in \mathbb{R}^{p+m}$ — solution of linear problem in geometric form obtained with Listing 2.

Using code snippet presenting on Listing 4, we determined that obtained solution z^* is indeed vertex of polyhedron in geometric form: there are exactly $(p + m)$ linearly independent constraints (what means that vertex even is not degenerate). Note that *np.isclose()* function is used as Python language can not guarantee numeric precision of values close to zero, so values like 10^{-10} should be indeed treated as 0.

7 Q5: Custom message experiments

To determine maximum level of noise up to what the message can be decrypted, the binary search algorithm was used (Listing 5). The maximum level of noise was calculated for three messages of different sizes. In addition to message length, the **signal dimension factor** — the scalar k in equation $m = kp$ for encoding matrix $A \in \mathbb{R}^{m \times p}$ (in initial problem we are given $m = 4p$).

[2]Possible explanation is that standard LOP has bigger dimension of target vector than geometric LOP. Namely, as in given case $p = 856, m = 4p = 3424$, $\tilde{x}$ from Equation (4) is from $\mathbb{R}^{14p} = \mathbb{R}^{11984}$ while z' from Equation (3) is only from $\mathbb{R}^{5p} = \mathbb{R}^{4280}$.

The results (Figure 1) are not surprising. The bigger signal dimension factor is the larger the decoded encoded message is, so the message transfer should be more robust to the noise. For example, even with signal dimension factor equals 1, considered messages can be transferred with about 10% disturbed data.

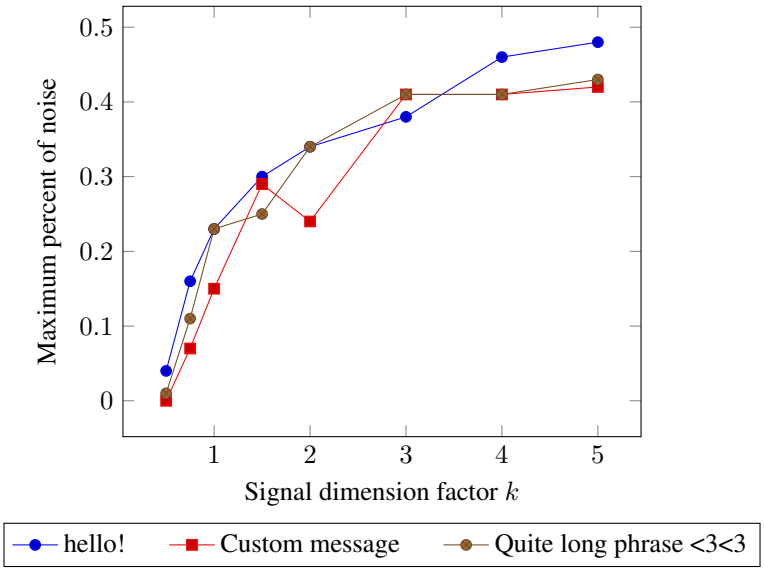

Figure 1: Maximum level of noise up to what the messages can be decrypted. Signal dimension factor k is defined from equation $m = kp$ for encoding matrix $A \in \mathbb{R}^{m \times p}$

8 Q6: Dikin's method

To start the Dikin's method, we first need to get a point from the feasible set of our problem. Since the dimension of our problem is quite large, we decided to use the 'dirty' method of getting the starting point. To do this, we had to solve the following problem in a standard form (from lecture slides):

$$\begin{aligned} &\min_{\tilde{x} \in \mathbb{R}^{2p+3m}} 0 \\ &\text{s.t. } A'\tilde{x} = b \\ &\tilde{x} \geq \varepsilon, \end{aligned} \tag{5}$$

where $c \in \mathbb{R}^{2p+3m} = 0$, $\varepsilon > 0$ — small constant, and other notations are the same as in Equation (4). The code with solution of this problem is presented in Listing 6.

Implementation of Dikin's algorithm is demonstrated on Listing 7. Note that proposed version of method contains one important modification — step size *alpha*, which regulates how far the algorithms go along found direction. During experiments, vanilla algorithm (with *alpha=1*) showed a little progress as the decoded message have not changed. Therefore we decided to increase step size to get at least some change in the decoded message.

After 10 iterations with *alpha=100*, which took about 5 minutes, the algorithm could not get close to a real solution. At the same time, the solution of linear program in geometric and standard forms using the SciPy made it possible to decrypt the message within 3 minutes.

Note that with such choice of *alpha* there is **no guarantee that algorithm converges**. However, we suppose that some progress still better than no progress at all.

In conclusion, SciPy approaches (Listings 2 and 3) look more preferable (at least for the given problem).

9 Q7: Integer programming

Imposing binary (integer) variables SciPy is done via adding parameter *integrality* to function *scipy.optimize.linprog*. Specifically, *integrality* determines for each variable whether it is integer or continuous: 0 means that parameter is continuous, 1 — integer.

Let us for this section consider LOP in geometric form (Equation (3)), as it is solved faster than LOP in standard form. So the integrality for z' would be $w = (1_p\ 0_m)^T$, as only part of z' denoting the x should be restricted to be integers. Moreover, first two 'row blocks' of A' can be removed, and instead the *bounds* parameter of *scipy.optimize.linprog* can be used. The bounds would be $(0, 1)$ for first p coordinates of z' (as $0 \leq x \leq 1$) and $(0, +\inf)$ for the last m coordinates (as $z \geq 0$). The resulting function is demonstrated on Listing 8. Note, that the maximum running time is restricted, as it may take a lot of time to complete (especially for not short messages and high percent of noise).

The results (Figure 2) show, that introducing integer restrictions increases the maximum percent of noise up to what the messages can be decrypted. However, time complexity of optimization problem with integer constraints is **significantly higher** (up to 10 times) than without integer constraints.

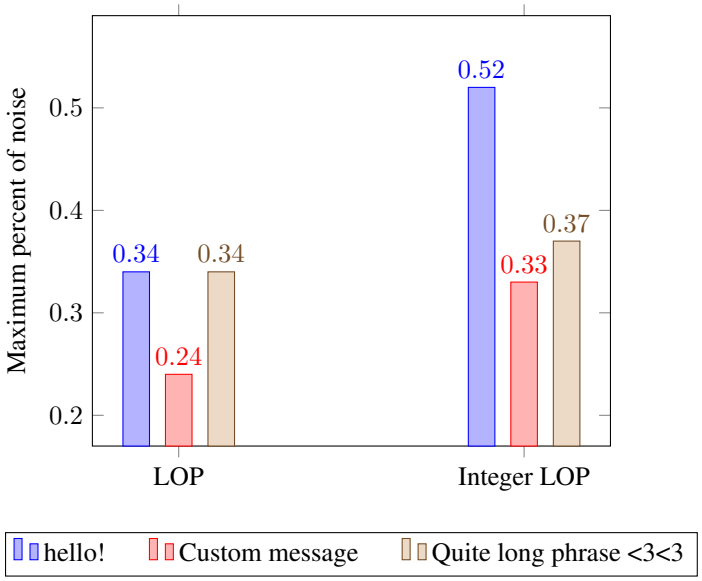

Figure 2: Maximum level of noise up to what the messages can be decrypted (signal dimension factor $k = 2$). *LOP* means that no integer constraints were applied to the parameters vector, and *Integer LOP* means that integer programming was applied

Supplementary material

A Listings for Q3

```python
def extract_message_naive(encoding_matrix, noisy_signal):
  res, *_ = scipy.linalg.lstsq(a=encoding_matrix,b=noisy_signal)
  return res, res
```

Listing 1: Naive message decryption. The output is tuple of solution of the problem, and decrypted message itself

```python
def extract_message_geometric(encoding_matrix, noisy_signal):

  # Size of A
  (m, p) = encoding_matrix.shape

  c = np.zeros(p+m)
  c[p : p + m] = np.ones(m)

  b = np.concat([np.zeros(p), -np.ones(p), -noisy_signal, noisy_signal])

  A = np.concat(
      [
          np.concat([np.identity(p), np.zeros((p, m))], axis=1),
          np.concat([-np.identity(p), np.zeros((p, m))], axis=1),
          np.concat([-encoding_matrix, np.identity(m)], axis=1),
          np.concat([encoding_matrix, np.identity(m)], axis=1),
      ]
  )

  # We add minuses, because from scipy documentation
  # A_ub x <= b_ub
  # But in geometric from we have constraints
  # A_ub x >= b_ub
  res = scipy.optimize.linprog(c, A_ub=-A, b_ub=-b, method="highs")

  return res.x, res.x[:p]
```

Listing 2: Message decryption based on solution of LOP in geometric form. The output is tuple of solution of the problem, and decrypted message itself

```python
def extract_message_standard(encoding_matrix, noisy_signal):

    # Size of A
    (m, p) = encoding_matrix.shape

    c = np.zeros(2 * p + 3 * m)
    c[p : p + m] = np.ones(m)

    b = np.concat([-np.ones(p), -noisy_signal, noisy_signal])

    A = np.concat(
        [
            np.concat(
                [
                    np.concat([-np.identity(p), np.zeros((p, m))], axis=1),
                    np.concat(
                        [-encoding_matrix, np.identity(m)],
                        axis=1,
                    ),
                    np.concat(
                        [
                            encoding_matrix,
                            np.identity(m),
                        ],
                        axis=1,
                    ),
                ]
            ),
            -np.identity(p + 2 * m),
        ],
        axis=1,
    )

    res = scipy.optimize.linprog(c, A_eq=A, b_eq=b, method="highs")

    return res.x, res.x[:p]
```

Listing 3: Message decryption based on solution of LOP in standard form. The output is tuple of solution of the problem, and decrypted message itself

B Listings for Q4

```python
def check_geom_solution_is_vertex(encoding_matrix: np.ndarray, noisy_signal: np.ndarray, solution: np.ndarray) -> bool:

    # In order to check, whether solution is a vertex, we need to use A and b from geometric form problem:
    (m, p) = encoding_matrix.shape
    b = np.concat([np.zeros(p), -np.ones(p), -noisy_signal, noisy_signal])

    A = np.concat(
        [
            np.concat([np.identity(p), np.zeros((p, m))], axis=1),
            np.concat([-np.identity(p), np.zeros((p, m))], axis=1),
            np.concat(
                [-encoding_matrix, np.identity(m)],
                axis=1,
            ),
            np.concat(
                [
                    encoding_matrix,
                    np.identity(m),
                ],
                axis=1,
            ),
        ]
    )

    (M, N) = A.shape

    # Check which constraints are tight with some tolerance
    close_sol_geom = np.isclose(A @ solution - b, 0, atol=1e-10).sum()
    mask_sol_geom = np.isclose(A @ solution - b, 0, atol=1e-10)

    tight_constraints = A[mask_sol_geom]

    rank = int(np.linalg.matrix_rank(tight_constraints))
    return bool(rank >= N) # True!
```

Listing 4: Determining number of tight linear independent constraints for solution of linear problem in geometric form to ccheck whetther it it is a vertex of corresponding polyhedron

C Listings for Q5

```python
for _ in range(max_iter):
    if up_bound - low_bound <eps:
        break
    current = (low_bound + up_bound)/2
    noise_custom = noisy_channel(y_custom, percent_error=current, seed=seed)

    decoded_custom, decoded_float_custom, __ =  extract_decoded_message(
        encoding_matrix_custom, noise_custom, dimensions_custom, extract_fn
    )

    if  decoded_custom == message_custom:
        low_bound = current
    else:
        up_bound = current
```

Listing 5: Binary search used to determine maximum level of noise up to what the message can be decrypted. *extract_fn* is either *extract_message_geometric* or *extract_message_standard*

D Listings for Q6

```python
def get_dikin_initial_point(
    encoding_matrix: np.ndarray, noisy_signal: np.ndarray, epsilon: float
) -> tuple[np.ndarray, np.ndarray]:

    # Size of A
    (m, p) = encoding_matrix.shape

    c = np.zeros(2 * p + 3 * m)

    b = np.concat([-np.ones(p), -noisy_signal, noisy_signal])

    A = np.concat(
        [
            np.concat(
                [
                    np.concat([-np.identity(p), np.zeros((p, m))], axis=1),
                    np.concat(
                        [-encoding_matrix, np.identity(m)],
                        axis=1,
                    ),
                    np.concat(
                        [
                            encoding_matrix,
                            np.identity(m),
                        ],
                        axis=1,
                    ),
                ]
            ),
            -np.identity(p + 2 * m),
        ],
        axis=1,
    )

    res = scipy.optimize.linprog(
        c,
        A_eq=A,
        b_eq=b,
        bounds=[(epsilon, None) for _ in range(2 * p + 3 * m)],
        method="highs",
    )

    return res.x, res.x[:p]
```

Listing 6: Extraction of dirty initial point for Dikin's method using standard form of linear optimization problem

```python
def extract_message_dikin(
    encoding_matrix: np.ndarray,
    noisy_signal: np.ndarray,
    x0: np.ndarray,
    alpha: float = 1.0,
) -> tuple[np.ndarray, np.ndarray]:

    # Setup
    (m, p) = encoding_matrix.shape

    c = np.zeros(2 * p + 3 * m)
    c[p : p + m] = np.ones(m)

    A = np.concat(
        [
            np.concat(
                [
                    np.concat([-np.identity(p), np.zeros((p, m))], axis=1),
                    np.concat(
                        [-encoding_matrix, np.identity(m)],
                        axis=1,
                    ),
                    np.concat(
                        [
                            encoding_matrix,
                            np.identity(m),
                        ],
                        axis=1,
                    ),
                ]
            ),
            -np.identity(p + 2 * m),
        ],
        axis=1,
    )

    x = x0.copy()

    # Dikin's method
    for k in range(10):
        H = np.diag(1 / (x**2 + 1e-7))
        H_inv = np.linalg.inv(H)
        A_H_inv = A @ H_inv
        nu = -1 * np.linalg.inv(A_H_inv @ A.T) @ A_H_inv @ c

        s = -1 * H_inv @ (c + A.T @ nu)
        mu = 1 / np.sqrt(s.T @ H @ s)

        delta_x = mu * s

        x = x + alpha * delta_x

    return x, x[:p]
```

Listing 7: Message decryption based on Dikin's method. The output is tuple of solution of the problem, and decrypted message itself

E Listings for Q7

```python
def extract_message_integer(encoding_matrix, noisy_signal):

    (m, p) = encoding_matrix.shape

    c = np.zeros(p+m)
    c[p : p + m] = np.ones(m)

    integrality = np.ones(p+m) - c
    bounds = [*[(0,1) for _ in range(p)], *[(0,None) for _ in range(m)]]

    b = np.concat([ -noisy_signal, noisy_signal])

    A = np.concat(
        [
            np.concat(
                [-encoding_matrix, np.identity(m)],
                axis=1,
            ),
            np.concat(
                [
                    encoding_matrix,
                    np.identity(m),
                ],
                axis=1,
            ),
        ]
    )

    # We add minuses, because from scipy documentation
    # A_ub x <= b_ub
    # But in geometric from we have constraints
    # A_ub x >= b_ub
    res = scipy.optimize.linprog(c, A_ub=-A, b_ub=-b,
                      options={"maxiter": 20, "time_limit": 90}, method="highs",
                      integrality=integrality, bounds=bounds)

    return res.x, res.x[:p]
```

Listing 8: Message decryption based on solution of integer LOP in geometric form. The output is tuple of solution of the problem and decrypted message itself

F Apologies

We sincerely apologize for the bad formatting of the appendix, in particular for the alignment of the listings blocks.