LINEAR PROGRAMMING PROJECT

GROUP 2 REPORT FOR OPTIMIZATION F24 COURSE

Dmitry Beresnev
MS-DS1, Innopolis University
d.beresnev@innopolis.university

Vsevolod Klyushev
MS-DS1, 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 deviation 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

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

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

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

where $c = \sum_{i=p+1}^{p+m} e_i \in \mathbb{R}^{(2p+3m)}$, $b' \in \mathbb{R}^{p+2m}$, $A' \in \mathbb{R}^{(p+2m)\times(2p+3m)}$, and $S^{a,b}$ — slack variable matrix of size $(b-a+1)\times(p+2m)$ with rows $S_i^{a,b} = e_{a+i-1}$, which 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.

```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

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.

```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

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). Moreover, the separate function *build_slack* to construct slack matrices $S^{a,b}$ (eq. (4)) is introduced. Also, as was mentioned in Section 4.2, we are interested only in the first p components of the solution.

```python
def build_slack(a, b, p, m):
    slack_matrix = np.zeros((b - a + 1, p + 2 * m))
    for i in range(b - a + 1):
        slack_matrix[i][a + i - 1] = 1
    return slack_matrix


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

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

...

[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 $\mathbb{R}^{5p} = \mathbb{R}^{4280}$.

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 4). 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 $m = 4p$).

```
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 4: 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*

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.

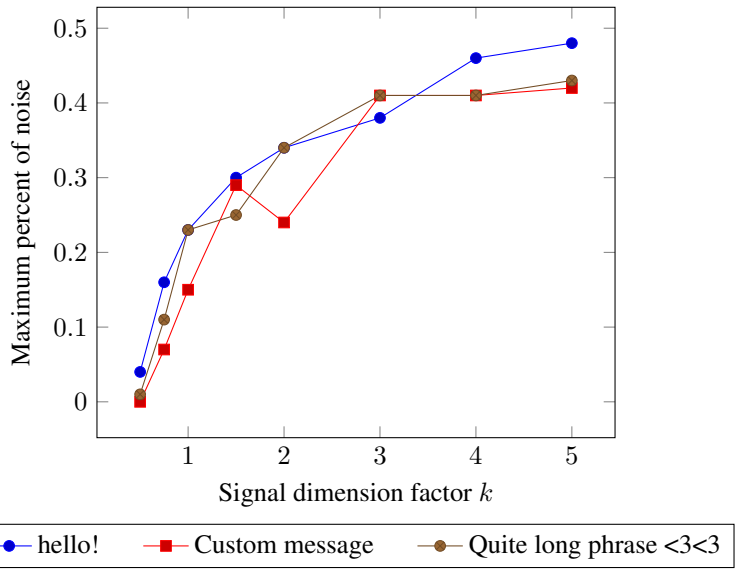

Figure 1: Maximum level of noise up to what the messages can be decrypted. Signal dimension factor k is defined from equation $m = kp$ for encoding matrix $A \in \mathbb{R}^{m \times p}$

8 Q6: Dikin's method

...

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 5.

```
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,
                                 method="highs", integrality=integrality, bounds
=bounds)

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

Listing 5: Message decryption based on solution of integer LOP in geometric form. The output is tuple of solution of the problem, and decrypted message itself

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.

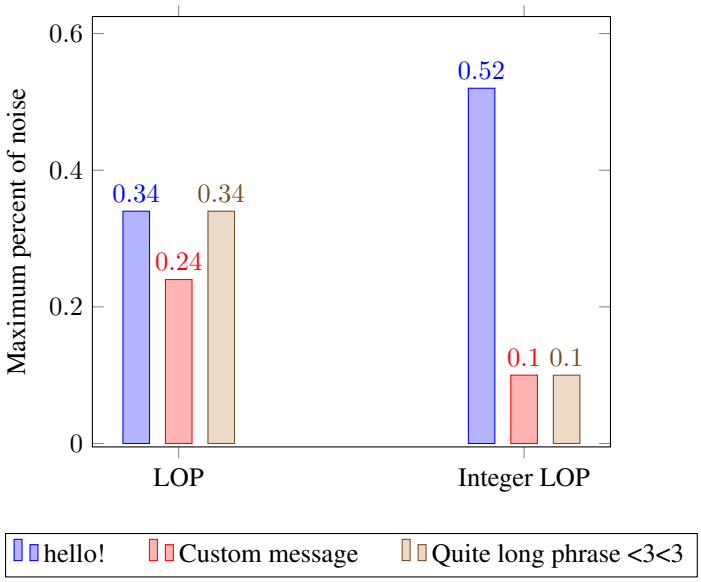

Figure 2: Maximum level of noise up to what the messages can be decrypted (signal dimension factor $k = 2$). *LOP* means that no integer constraints were applied to the parameters vector, and *Integer LOP* means that integer programming was applied