Report 2

Team information (B23-ISE-02):

- Team member 1 (leader): Kirill Efimovich (k.efimovich@innopolis.university), grade 5/5
- Team member 2: Arsen Galiev (a.galiev@innopolis.university), grade: 5/5
- Team member 3: Asqar Arslanov (a.arslanov@innopolis.university), grade: 5/5
- Team member 4: Ilya-Linh Nguen (i.nguen@innopolis.university), grade: 5/5

Link to the product:

- https://github.com/quintet-sdr/optimization-pt2

Programming language:

- Rust
- To launch the code: `$ cargo run`

Linear programming problem:

- Maximization
- Approximation accuracy $\epsilon = 2$
- $\alpha_1 = 0.5$
- $\alpha_2 = 0.9$

Tests:

In all tests answers for α_1 and α_2 are the same

(1) **Input:**

- Name: Lab 6 / Problem 1
- Objective function: $F(x_1, x_2) = x_1 + x_2$
- Constraints: $\begin{cases} 2x_1 + 4x_2 \leq 16 \\ x_1 + 3x_2 \geq 9 \\ x_1, x_2 \geq 0 \end{cases}$
- Initial point: $\begin{bmatrix} 0.5 & 3.5 & 1 & 2 \end{bmatrix}$.

Output:

- Decision variables: $\begin{bmatrix} 6 & 1 & 0 & 0 \end{bmatrix}$.
- Maximum values: 7.

(2) **Input:**

- Name: Lab 6 / Problem 2
- Objective function: $F(x_1, x_2, x_3) = 9x_1 + 10x_2 + 16x_3$
- Constraints: $\begin{cases} 18x_1 + 15x_2 + 12x_3 \leq 360 \\ 6x_1 + 4x_2 + 8x_3 \leq 192 \\ 5x_1 + 3x_2 + 3x_3 \leq 180 \\ x_1, x_2, x_3 \geq 0 \end{cases}$
- Initial point: $\begin{bmatrix} 1 & 1 & 1 & 315 & 174 & 169 \end{bmatrix}$.

Output:

- Decision variables: $\begin{bmatrix} 0 & 8 & 20 & 0 & 0 & 96 \end{bmatrix}$.
- Maximum value: 400.

(3) **Input:**

- Name: Lec 6 / Problem 1
- Objective function: $F(x_1, x_2, x_3) = x_1 + 2x_2$
- Constraints: $\begin{cases} x_1 + x_2 + x_3 = 8 \\ x_1, x_2, x_3 \geq 0 \end{cases}$
- Initial point: $\begin{bmatrix} 2 & 2 & 4 \end{bmatrix}$.

Output:

- Decision variables: $\begin{bmatrix} 0 & 8 & 0 \end{bmatrix}$.
- Maximum value: 16.

(4) **Input:**

- Name: Tut 6 / Problem 1
- Objective function: $F(x_1, x_2) = 2x_1 + 5x_2 + 7x_3$

- Constraints: $\begin{cases} 1x_1 + 2x_2 + 3x_3 = 6 \\ x_1, x_2, x_3 \geq 0 \end{cases}$
- Initial point: $\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$.

Output:

- Decision variables: $\begin{bmatrix} 0 & 3 & 0 \end{bmatrix}$.
- Maximum value: 15.

(5) **Input:**

- Name: No solution 1
- Objective function: $F(x_1, x_2) = 3x_1 + 2x_2$
- Constraints: $\begin{cases} x_1 - x_2 \leq 2 \\ -2x_1 + x_2 \leq -1 \\ x_1, x_2 \geq 0 \end{cases}$
- Initial point: $\begin{bmatrix} 1 & 1 \end{bmatrix}$.

Output:

- The problem doesn't have a solution.

(6) **Input:**

- Name: No solution 2
- Objective function: $F(x_1, x_2) = 2x_1 + 1x_2$
- Constraints: $\begin{cases} x_1 - x_2 \leq 10 \\ -2x_1 \leq 40 \\ x_1, x_2 \geq 0 \end{cases}$
- Initial point: $\begin{bmatrix} 0 & 0 \end{bmatrix}$.

Output:

- The problem doesn't have a solution.

Code:

crates/pt2-cli/src/main.rs

```
use std::panic;

use color_eyre::Result;

mod cli;
mod config;

fn main() -> Result<()> {
    color_eyre::install()?;

    cli::enter_alternate_screen()?;
    panic::set_hook(Box::new(|_| {
        cli::enter_alternate_screen().unwrap();
        cli::leave_alternate_screen().unwrap();
    }));

    let result = cli::run();

    cli::leave_alternate_screen()?;
    result
}
```

crates/pt2-cli/src/cli.rs

```
use std::io;

use color_eyre::{eyre::Context, Result};
use crossterm::terminal::{Clear, ClearType, EnterAlternateScreen, LeaveAlternateScreen};

use crate::config::{self, Test};

pub fn run() -> Result<()> {
    let tests = config::read_tests().wrap_err("tests.json not found")?;
    while matches!(prompt(tests.clone())?, Next::Continue) {}
    Ok(())
}

enum Next {
    Continue,
    Break,
}

fn clear() -> io::Result<()> {
    crossterm::execute!(io::stdout(), Clear(ClearType::All))
}

fn prompt(tests: Vec<Test>) -> Result<Next> {
    const ALPHA_1: f64 = 0.5;
    const ALPHA_2: f64 = 0.9;

    for _ in 0..crossterm::terminal::size()?.1 {
        println!()
    }

    let Some(test) = inquire::Select::new("Select a test:", tests)
        .with_vim_mode(true)
        .prompt_skippable()?
    else {
        return Ok(Next::Break);
    };

    for alpha in [ALPHA_1, ALPHA_2] {
        let Ok((lpp, iterations)) = pt2_core::interior_point(
            test.objective_function.clone(),
            &test.constraints,
            test.initial_point.clone(),
            test.eps,
            alpha,
        ) else {
            println!("The method is not applicable.");
            break;
        };

        println!("Alpha: {alpha}");

        println!("Epsilon: {} ({:.eps$})", test.eps, lpp.eps, eps = test.eps);

        println!(
            "Objective function: {:.eps$?}",
            lpp.c.iter().collect::<Box<[_]>>(),
            eps = test.eps,
        );
        println!(
            "Initial point: {:.eps$?}",
```

```
            lpp.x.iter().collect::<Box<[_]>>(),
            eps = test.eps,
        );
        println!("Constraints:{:.eps$}", lpp.big_a, eps = test.eps);

        let last = iterations.last().unwrap();

        let Ok(result) = last else {
            println!("The problem doesn't have a solution.");
            println!();
            continue;
        };

        println!("Result:");
        println!("Maximum: {:.eps$}", result.max, eps = test.eps);
        println!(
            "Decision variables: {:.eps$?}",
            result.decision_variables.iter().collect::<Box<[_]>>(),
            eps = test.eps
        );
        println!();
        println!("{}", "-".repeat(16));
        println!();
    }

    let Some(next) = inquire::Confirm::new("Next test?").prompt_skippable()? else {
        return Ok(Next::Break);
    };
    if !next {
        return Ok(Next::Break);
    };

    Ok(Next::Continue)
}

pub fn enter_alternate_screen() -> io::Result<()> {
    crossterm::execute!(io::stdout(), EnterAlternateScreen)
}

pub fn leave_alternate_screen() -> io::Result<()> {
    crossterm::execute!(io::stdout(), LeaveAlternateScreen)
}
```

crates/pt2-cli/src/config.rs

```rust
use std::fmt::{self, Display, Formatter};
use std::fs::File;
use std::io::BufReader;

use color_eyre::Result;
use serde::Deserialize;

use pt2_core::Constraints;

#[derive(Clone, Deserialize)]
#[serde(rename_all = "kebab-case")]
pub struct Test {
    #[serde(default = "name_default")]
    pub name: Box<str>,
    pub objective_function: Vec<f64>,
    pub constraints: Constraints,
    pub initial_point: Vec<f64>,
    #[serde(alias = "epsilon", default = "eps_default")]
    pub eps: usize,
}

impl Display for Test {
    fn fmt(&self, f: &mut Formatter<'_>) -> fmt::Result {
        write!(f, "{}", self.name)
    }
}

pub fn read_tests() -> Result<Vec<Test>> {
    let tests_file = BufReader::new(File::open("tests.json")?);
    Ok(serde_json::from_reader(tests_file)?)
}

fn name_default() -> Box<str> {
    Box::from("Unnamed")
}

const fn eps_default() -> usize {
    2
}
```

crates/pt2-core/src/lib.rs

```
use lpp::Lpp;

pub use interfaces::{Constraints, Sign};
use interfaces::{InteriorPoint, NotApplicableError};

mod algorithm;
mod interfaces;
mod lpp;

pub fn interior_point(
    objective_function: Vec<f64>,
    constraints: &Constraints,
    initial_point: Vec<f64>,
    eps: usize,
    alpha: f64,
) -> Result<(Lpp, InteriorPoint), NotApplicableError> {
    let lpp = Lpp::try_new(objective_function, constraints, initial_point, eps)?;

    Ok((
        lpp.clone(),
        InteriorPoint {
            done: false,
            x: lpp.x,
            big_a: lpp.big_a,
            c: lpp.c,
            eps: lpp.eps,
            alpha,
        },
    ))
}
```

crates/pt2-core/src/algorithm.rs

```rust
use na::{DMatrix, DVector};

use crate::interfaces::{Auxiliary, InteriorPoint, Iteration, NoSolutionError};

impl Iterator for InteriorPoint {
    type Item = Result<Iteration, NoSolutionError>;

    fn next(&mut self) -> Option<Self::Item> {
        if self.done {
            return None;
        }

        let size = self.x.len();

        let big_d = DMatrix::from_diagonal(&self.x);

        let big_a_tilde = &self.big_a * &big_d;
        let c_tilde = &big_d * &self.c;

        let big_p = {
            let big_i = DMatrix::identity(size, size);
            let big_a_tilde_tr = big_a_tilde.transpose();
            let Some(inverse) = (&big_a_tilde * &big_a_tilde_tr).try_inverse() else {
                self.done = true;
                return Some(Err(NoSolutionError));
            };
            big_i - big_a_tilde_tr * inverse * &big_a_tilde
        };
        let c_p = &big_p * &c_tilde;

        let Some(nu) = c_p
            .iter()
            .filter_map(|it| (it < &0.).then_some(it.abs()))
            .max_by(|a, b| a.partial_cmp(b).unwrap())
        else {
            self.done = true;
            return Some(Err(NoSolutionError));
        };
        let x_tilde = DVector::from_element(size, 1.) + (self.alpha / nu) * &c_p;

        let new_x = &big_d * &x_tilde;
        if (&new_x - &self.x).norm() < self.eps {
            self.done = true;
        }

        self.x = new_x;

        Some(Ok(Iteration {
            auxiliary: Auxiliary {
                big_d,
                big_a_tilde,
                c_tilde,
                big_p,
                c_p,
                nu,
                x_tilde,
            },
            decision_variables: self.x.clone_owned(),
            max: self.x.dot(&self.c),
        }))
```

```
    }
}
```

crates/pt2-core/src/interfaces.rs

```rust
use na::{DMatrix, DVector};
use serde::Deserialize;
use thiserror::Error;

#[derive(Error, Debug)]
#[error("method is not applicable")]
pub struct NotApplicableError;

#[derive(Error, Debug)]
#[error("problem has no solution")]
pub struct NoSolutionError;

pub struct Auxiliary {
    pub big_d: DMatrix<f64>,
    pub big_a_tilde: DMatrix<f64>,
    pub c_tilde: DVector<f64>,
    pub big_p: DMatrix<f64>,
    pub c_p: DVector<f64>,
    pub nu: f64,
    pub x_tilde: DVector<f64>,
}

pub struct Iteration {
    pub auxiliary: Auxiliary,
    pub decision_variables: DVector<f64>,
    pub max: f64,
}

pub struct InteriorPoint {
    pub(crate) done: bool,
    pub(crate) x: DVector<f64>,
    pub(crate) big_a: DMatrix<f64>,
    pub(crate) c: DVector<f64>,
    pub(crate) eps: f64,
    pub(crate) alpha: f64,
}

pub type Constraints = Box<[(Box<[f64]>, Sign, f64)]>;

#[derive(Clone, Deserialize)]
pub enum Sign {
    #[serde(rename = "<=")]
    Le,
    #[serde(rename = "==", alias = "=")]
    Eq,
    #[serde(rename = ">=")]
    Ge,
}

impl Sign {
    pub fn compare<Lhs, Rhs>(&self, a: &Lhs, b: &Rhs) -> bool
    where
        Lhs: PartialOrd<Rhs>,
    {
        let cmp_function = match self {
            Self::Le => PartialOrd::le,
            Self::Eq => PartialEq::eq,
            Self::Ge => PartialOrd::ge,
        };
```

```
            cmp_function(a, b)
        }
    }
```

crates/pt2-core/src/lpp.rs

```rust
use na::{DMatrix, DVector};

use crate::{interfaces::NotApplicableError, Constraints, Sign};

#[derive(Clone)]
pub struct Lpp {
    pub x: DVector<f64>,
    pub big_a: DMatrix<f64>,
    pub c: DVector<f64>,
    pub eps: f64,
}

impl Lpp {
    pub fn try_new(
        objective_function: Vec<f64>,
        constraints: &Constraints,
        initial_point: Vec<f64>,
        eps: usize,
    ) -> Result<Lpp, NotApplicableError> {
        let (n, m) = get_n_and_m(constraints).ok_or(NotApplicableError)?;

        if constraints
            .iter()
            .any(|row| row.0.len() != objective_function.len())
        {
            return Err(NotApplicableError);
        }

        let initial_point_is_feasible = constraints.iter().all(|(coefficients, sign, rhs)| {
            let constraint_sum: f64 = coefficients
                .iter()
                .zip(&initial_point)
                .map(|(coeff, x)| coeff * x)
                .sum();

            sign.compare(&constraint_sum, rhs)
        });

        if !initial_point_is_feasible {
            return Err(NotApplicableError);
        }

        let no_slack_rows = constraints
            .iter()
            .enumerate()
            .filter_map(|(i, (_, sign, _))| matches!(sign, Sign::Eq).then_some(i));
        let no_slack_cols = no_slack_rows.map(|j| m + j).collect::<Box<[_]>>();
        let slack_cols_count = n - no_slack_cols.len();

        if initial_point.len() != m + slack_cols_count {
            return Err(NotApplicableError);
        }

        let x = DVector::from_vec(initial_point);
        let big_a = {
            let left_part_row_elements = constraints
                .iter()
                .flat_map(|(coefficients, _, _)| coefficients)
                .copied();
```

```
            let right_part_diagonal_elements = &DVector::from_vec(
                constraints
                    .iter()
                    .map(|(_, sign, _)| match sign {
                        Sign::Le => 1.,
                        Sign::Eq => 0.,
                        Sign::Ge => -1.,
                    })
                    .collect(),
            );

            let mut big_a = DMatrix::from_row_iterator(n, m, left_part_row_elements)
                .resize_horizontally(m + n, 0.);

            big_a
                .view_mut((0, m), (n, n))
                .set_diagonal(right_part_diagonal_elements);

            big_a.remove_columns_at(&no_slack_cols)
        };
        let c = DVector::from_vec(objective_function).resize_vertically(m + slack_cols_count, 0.);
        let eps = up_to_n_dec_places(i32::try_from(eps).map_err(|_| NotApplicableError)?);

        Ok(Lpp { x, big_a, c, eps })
    }
}

fn get_n_and_m(constraints: &Constraints) -> Option<(usize, usize)> {
    Some((constraints.len(), constraints.first()?.0.len()))
}

fn up_to_n_dec_places(n: i32) -> f64 {
    0.1_f64.powi(n) / 2.
}
```