Diberikan dua buah SPL sebagai berikut:

(a.) SPL 1:

$$\begin{aligned} x_1 + x_2 + 3x_3 &= 10 \\ 3x_1 - x_2 + x_3 &= -2 \\ x_1 + 4x_2 - x_3 &= 4^2 \end{aligned}$$

(b.) SPL 2:

$$\begin{aligned} 2x_1 + 9x_2 - x_3 - 2x_4 &= 1 \\ x_1 + 3x_2 + 2x_3 - x_3 &= 2 \\ x_1 + 6x_3 + 4x_4 &= 3 \\ 8x_1 + x_2 + 3x_3 + 2x_4 &= 0 \end{aligned}$$

1. Bagaimana cara eliminasi Gauss naif dan eliminasi Gauss *pivoting* parsial menyelesaikan kedua SPL atau menentukan solusi eksak kedua SPL tersebut? Jelaskan!

   Langkah pertama dalam menyelesaikan SPL menggunakan metode eliminasi Gauss naif adalah mengubah SPL menjadi matriks

   $$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & ... & + & a_{1n}x_n & = b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & ... & + & a_{2n}x_n & = b_2 \\ \vdots & & \vdots & & \ddots & & \vdots & \vdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + & ... & + & a_{nn}x_n & = b_n \end{array} \quad \rightarrow \quad \left[\begin{array}{cccc|c} a_{11} & a_{12} & ... & a_{1n} & b_1 \\ a_{21} & a_{22} & ... & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & ... & a_{nn} & b_n \end{array}\right]$$

   Kemudian menggunakan operasi baris elementer (OBE), ubah matriks menjadi matriks segitiga atas sebagai berikut

   $$\left[\begin{array}{cccc|c} a'_{11} & a'_{12} & ... & a'_{1n} & b'_1 \\ 0 & a'_{22} & ... & a'_{2n} & b'_1 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & ... & 0 & a'_{nn} & b'_n \end{array}\right]$$

   Kemudian nilai $x_1, x_2, ..., x_n$ dapat diperoleh melalui

   $$\begin{aligned} x_n &= \frac{b_n}{a_n} \\ x_n &= \frac{b_{n-1} - a_{(n-1)n}x_n}{a_{(n-1)(n-1)}} \\ &\vdots \end{aligned}$$

   Terkadang eliminasi Gauss tidak dapat dihitung langsung ketika terdapat 0 di diagonal matriks. Namun hal ini dapat diatasi dengan merubah posisi baris-baris

pada matriks sehingga tidak terdapat 0 di diagonal matriks. proses ini dinamakan *pivoting*. Salah satu *pivoting* yang dapat digunakan adalah *pivoting* parsial yaitu *pivoting* yang hanya dilakukan per kolom. Jika

$$|x_{ii}| < |x_{ji}| \Rightarrow \text{ Tukar baris } i \text{ dan } j, j > i$$

pivoting ini dapat dilakukan seiring OBE pada metode eliminasi Gauss sehingga diperoleh metode Gauss diperbaiki atau Gauss *pivoting* parsial.

Eliminasi Gauss naif dan eliminasi Gauss *pivoting* parsial dapat dilakukan menggunakan algoritma. Dalam hal ini akan digunakan bahasa pemograman "Rust" yaitu sebagai berikut

```rust
// Library yang digunakan:
//      std::fs -> membaca file
use std::fs::*;

// Tipe data "Persamaan"
struct Persamaan{
    matrix : Vec<Vec<f64>>,
    vector : Vec<f64>,
    size : usize
}

// Fungsi terhadap tipe data "Persamaan"
impl Persamaan {

    // Membaca file "koef.txt" dan "konst.txt" ke variabel "Persamaan" baru
    fn read() -> Persamaan{
        let koef = read_to_string("./koef.txt").unwrap();
        let kosnt = read_to_string("./konst.txt").unwrap();
        if koef.lines().count() != kosnt.lines().count(){
            panic!("Difference in Matrix size and Vector size");
        }
        let mut buffer = Persamaan{
            matrix : Vec::new(),
            vector : Vec::new(),
            size : koef.lines().count()
        };
        let mut temp : Vec<&str>;
        let mut matrix_buffer : Vec<f64>;
        for i in 0..buffer.size{
            buffer.vector.push(kosnt.lines().nth(i).unwrap().trim().parse().expect("Failed to parse"));
            temp = koef.lines().nth(i).unwrap().split(' ').collect();
            if temp.len() != buffer.size{
                panic!("Not a square matrix");
            }
            matrix_buffer = Vec::new();
            for value in temp{
                matrix_buffer.push(value.trim().parse().expect("Failed to parse"));
            }
            buffer.matrix.push(matrix_buffer);
        }
        return buffer;
    }

    // Eliminasi Gauss naif tanpa pivoting parsial:
    //      Mengubah matriks pada variabel "Persamaan"
    //      menjadi matriks segitiga atas
    fn gauss_naif(&mut self){
        let mut multiplier : f64;
        for i in 1..self.size{
            for j in i..self.size{
                multiplier = self.matrix[j][i-1]/self.matrix[i-1][i-1];
                self.vector[j] = self.vector[j] - self.vector[i-1] * multiplier;
                for k in 0..self.size{
```

```
                    self.matrix[j][k] = self.matrix[j][k] - self.matrix[i-1][k] * multiplier;
                }
            }
        }
    }

    // Fungsi pivoting pasrial
    fn pivot(&mut self, index: usize){
        for i in index+1..self.size{
            if self.matrix[i][index].abs() > self.matrix[index][index].abs(){
                self.matrix.swap(index, i);
                self.vector.swap(index, i);
            }
        }
    }

    // Elminiasi Gauss naif dengan pivoting parsial
    fn gauss_pivot(&mut self){
        let mut multiplier : f64;
        for i in 1..self.size{
            self.pivot(i-1);
            for j in i..self.size{
                multiplier = self.matrix[j][i-1]/self.matrix[i-1][i-1];
                self.vector[j] = self.vector[j] - self.vector[i-1] * multiplier;
                for k in 0..self.size{
                    self.matrix[j][k] = self.matrix[j][k] - self.matrix[i-1][k] * multiplier;
                }
            }
        }
    }

    // Perhitungan nilai solusi bedasarkan matriks segitiga atas
    fn solve(&self) -> Vec<f64> {
        let max_index = self.size-1;
        let mut buffer = Vec::new();
        buffer.push(self.vector[max_index]/self.matrix[max_index][max_index]);
        let mut sum : f64;
        for i in (0..max_index).rev(){
            sum = 0.0;
            for j in (i + 1 ..= max_index).rev() {
                sum = sum + (self.matrix[i][j] * buffer[max_index - j]);
            }
            buffer.push((self.vector[i] - sum) / self.matrix[i][i])
        }
        buffer.reverse();
        return buffer;
    }
}

// Fungsi validasi solusi
fn validate(hasil: Vec<f64>){
    let mut eval : f64;
    let mut result = true;
    for i in 0..Persamaan::read().size{
        eval = 0.0;
        for j in 0..hasil.len(){
            eval = eval + Persamaan::read().matrix[i][j]*hasil[j];
        }
        result = eval == Persamaan::read().vector[i];
    }
    match result {
        true => println!("Pass!"),
        false => println!("Error!")
    }
}

// Entry-point
fn main() {

    // Perhitunagn menggunakan Gauss naif
```

```
    println!("\nGauss Naif:");
    let mut persamaan = Persamaan::read();
    persamaan.gauss_naif();
    let hasil_naif = persamaan.solve();
    for i in 0..hasil_naif.len(){
        println!("x{} = {}", i + 1, hasil_naif[i]);
    }
    validate(hasil_naif);

    // Perhtingan menggunakan Gauss pivoting parsial
    println!("\nGauss Pivot Parsial");
    persamaan = Persamaan::read();
    persamaan.gauss_pivot();
    let hasil_pivot = persamaan.solve();
    for i in 0..hasil_pivot.len(){
        println!("x{} = {}", i + 1, hasil_pivot[i]);
    }
    validate(hasil_pivot);
}
```

Hasil perhitungan untuk SPL 1 dan SPL 2 adalah sebagai berikut

(a) SPL 1:

```
Gauss Naif:
x1 = -1
x2 = 2
x3 = 3
Dengan:
Persamaan 1 = 10
Persamaan 2 = -2
Persamaan 3 = 4
Kesimpulan: Pass!

Gauss Pivot Parsial
x1 = -0.9999999999999999
x2 = 2.0000000000000004
x3 = 3
Dengan:
Persamaan 1 = 10
Persamaan 2 = -2
Persamaan 3 = 4.000000000000002
Kesimpulan: Error!
```

Dari hasil diatas, terdapat perbedaan antara hasil dari metode Gauss naif dengan metode Gauss *pivoting* parsial serta solusi yang diperoleh metode Gauss *pivoting* pasrial tidak valid. Namun kesalahan ini sangat kecil dan disebabkan oleh akurasi *floating point* 64 bit yang terbatas. Diperoleh solusi untuk SPL 1 adalah $x_1 = -1, x_2 = 2, x_3 = 3$.

(b) SPL 2:

```
Gauss Naif:
x1 = -0.2243243243243226
x2 = 0.182432432432432
x3 = 0.7094594594594592
x4 = -0.25810810810810814
Dengan:
Persamaan 1 = 1
Persamaan 2 = 2
Persamaan 3 = 3.0000000000000004
```

```
Persamaan 4 = 0.00000000000001265654248072678 5
Kesimpulan: Error!

Gauss Pivot Parsial
x1 = -0.2243243243243243
x2 = 0.18243243243243243
x3 = 0.7094594594594594
x4 = -0.2581081081081081
Dengan:
Persamaan 1 = 0.9999999999999999
Persamaan 2 = 1.9999999999999998
Persamaan 3 = 3
Persamaan 4 = -0.00000000000000011102230246251565
Kesimpulan: Error!
```

Sama seperti pada SPL 1, kedua metode memperoleh solusi yang berbeda dengan keduanya tidak valid. Kesalahan ini juga disebabkan hal yang sama pada SPL 1 dimana kesalahannya sangat kecil. Diperoleh $x_1 = -0.224..., x_2 = 0.182..., x_3 = 0.709, x_4 = -0.258$

2. Bagaimana cara metode iterasi Jacobi dan metode iteratif Gauss-Seidel menyelesaikan masalah solusi kedua SPL tersebut (bagaimana cara memperoleh hampiran solusi kedua SPL)? Jelaskan!

   Misalkan SPL sebagai berikut

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & ... & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & ... & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & \ddots & & \vdots & & \vdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + & ... & + & a_{nn}x_n & = & b_n \end{array}$$

   Maka berlaku

$$x_k = \frac{b_k - \sum_{j=0, j\neq k}^{n} a_{kj}x_j}{a_{kk}}, k = 1, 2, ..., n$$

   Persamaan diatas merupakan dasar dari metode hampiran solusi SPL iteratif Jacobi. Dari perkiraan hampiran awal $\{x_k^{(0)}, k = 1, 2, ..., n\}$, hampiran selanjutnya dapat dihitung melalui

$$x_k^{(i)} = \frac{b_k - \sum_{j=0, j\neq k}^{n} a_{kj}x_j^{(i-1)}}{a_{kk}}, k = 1, 2, ..., n$$

   dengan $x_k^{(i)}$ adalah hampiran solusi untuk x_k pada iterasike i. Namun metode ini hanya akan memperoleh hasil jika iterasi konvergen ke suatu nilai. Baiknya, kekonvergenan iterasi terjamin jika matriks dari SPL merupakan matriks diagonal dominan yaitu matriks yang memenuhi

$$|a_{ii}| > \sum_{j=0, j\neq i}^{n} |a_{ij}|, i = 1, 2, ..., n$$

Sehingga jika suatu SPL dapat diubah menjadi matriks diagonal dominan maka SPL itu dapat diperoleh hampiran solusinya. Sedangkan untuk SPL yang tidak bisa diubah menjadi matriks diagonal dominan, kekonvergenannya tidak terjamin sehingga perolehan hampiran solusinya juga tidak terjamin.

Dari rumus metode iteratif Jacobi, perhatikan bahwa untuk mennghitung $x_k^{(i)}, \exists k \in [1, n]$, diperlukan $x_j^{(i-1)}, \forall j \in [1, n]$. Padahal terkadang nilai $x_l^{(i)}, \exists l \neq k$ sudah diperoleh. Nilai-nilai hampiran solusi ini digunakan pada metode Gauss-Seidel untuk memperbaiki metode iteratif Jacobi agar iterasinya lebih sedikit. Rumus dari metode ini adalah sebagai berikut

$$x_k^{(i)} = \frac{b_k - \sum_{j=0, j\neq k}^{n} a_{kj} x_j^{(i-1)} - \sum_{j=0, j\neq k}^{n} a_{kj} x_j^{(i)}}{a_{kk}}, k = 1, 2, ..., n$$

Seperti pada no. 1, kedua metode ini akan dijalankan menggunakan bahasa pemograman "Rust" yaitus sebagai berikut

```rust
use std::io;
use std::fs::*;

struct Problem{
    ukuran : usize,
    koefisien : Vec<Vec<f64>>,
    konstan : Vec<f64>,
    solusi : Vec<f64>,
    unstable : bool
}

impl Problem {
    fn init() -> Problem {
        let file_koef = read_to_string("./koef.txt").unwrap();
        let file_konst = read_to_string("./konst.txt").unwrap();
        if file_koef.lines().count() != file_konst.lines().count(){
            panic!();
        }
        let mut buffer = Problem{
            ukuran : file_konst.lines().count(),
            koefisien : Vec::new(),
            konstan : Vec::new(),
            solusi : Vec::new(),
            unstable : true
        };
        let mut temp: Vec<&str>;
        let mut koef_buffer: Vec<f64>;
        for i in 0..buffer.ukuran{
            buffer.konstan.push(file_konst.lines().nth(i).unwrap().trim().parse().expect("Parsing failed"));
            temp = file_koef.lines().nth(i).unwrap().split(' ').collect();
            if temp.len() != buffer.ukuran{
                panic!("Not a square matrix")
            }
            koef_buffer = Vec::new();
            for value in temp{
                koef_buffer.push(value.trim().parse().expect("Parsing failed"))
            }
            buffer.koefisien.push(koef_buffer);
            buffer.solusi.push(0.0);
        }
        return buffer;
    }

    fn eval (&self, index: usize) -> f64 {
        let mut sum = 0.0;
        for i in 0..self.ukuran{
            sum += self.solusi[i] * self.koefisien[index][i];
        }
        return sum;
    }

    fn dominan_diagonal(&mut self) {
        let mut buffer = self.koefisien.clone();
        let mut temp = self.konstan.clone();
        let mut max: f64;
        for i in 0..self.ukuran {
```

```
            max = 0.0;
            for line in &buffer {
                if (line[i]).abs() > max.abs() {
                    max = line[i];
                };
            };
            let mut j = 0;
            while j < buffer.len() && buffer[j][i] != max {
                j += 1;
            }
            self.koefisien[i] = buffer[j].clone();
            self.konstan[i] = temp[j];
            buffer.remove(j);
            temp.remove(j);
        }
    }

    fn iteras_jacobi(&mut self, threshold: f64, accuracy: f64) {
        self.unstable = false;
        let buffer = self.solusi.clone();
        for i in 0..self.ukuran{
            self.solusi[i] = self.konstan[i];
            for j in 0..self.ukuran{
                if i != j {
                    self.solusi[i] -= self.koefisien[i][j]*buffer[j];
                }
            }
            self.solusi[i] /= self.koefisien[i][i];
            if (self.solusi[i] - buffer[i]).abs() > threshold || (self.eval(i) - self.konstan[i]).abs() > accuracy {
                self.unstable = true;
            }
        }
    }

    fn gauss_seidel(&mut self, threshold: f64, accuracy: f64) {
        self.unstable = false;
        let buffer = self.solusi.clone();
        for i in 0..self.ukuran{
            self.solusi[i] = self.konstan[i];
            for j in 0..self.ukuran{
                if i != j {
                    self.solusi[i] -= self.koefisien[i][j]*self.solusi[j];
                }
            }
            self.solusi[i] /= self.koefisien[i][i];
            if (self.solusi[i] - buffer[i]).abs() > threshold || (self.eval(i) - self.konstan[i]).abs() > accuracy {
                self.unstable = true;
            }
        }
    }
}

macro_rules! read {
    ($var_name : ident, $typ : ty, $msg : expr) =>  {
        println!();
        println!($msg);
        let mut $var_name = String::new();
        io::stdin()
            .read_line(&mut $var_name)
            .expect("Failed");
        let $var_name : $typ = $var_name.trim().parse().expect("Failed");
    };
}

macro_rules! readmut {
    ($var_name : ident, $typ : ty, $msg : expr) => {
        println!();
        println!($msg);
        let mut $var_name = String::new();
        io::stdin()
            .read_line(&mut $var_name)
            .expect("Failed");
        let mut $var_name : $typ = $var_name.trim().parse().expect("Failed");
    };
}

fn main() {
    read!(galat_iterasi, f64, "Galat iterasi = ");
    read!(galat_akurasi, f64, "Galat mutlak = ");
    read!(max_iter, usize, "Batas iterasi= ");

    println!("\n\nMetode Iterasi Jacobi:");
    let mut spl = Problem::init();
    spl.dominan_diagonal();
    let mut iterasi = 0;
    while spl.unstable && iterasi < max_iter{
        spl.iteras_jacobi(galat_iterasi, galat_akurasi);
```

```
        iterasi += 1;
    }
    println!();
    for i in 0..spl.ukuran {
        println!("x{} = {}", i+1, spl.solusi[i]);
    }
    println!();
    for i in 0..spl.ukuran {
        println!("Persamaan {} = {}\n Dengan galat: {}", i+1, spl.eval(i), (spl.eval(i) - spl.konstan[i]).abs());
    }
    println!("\n Jumlah total iteras: {iterasi}");

    println!("\n\nMetode Gauss-Seidel");
    spl = Problem::init();
    spl.dominan_diagonal();
    iterasi = 0;
    while spl.unstable && iterasi < max_iter{
        spl.gauss_seidel(galat_iterasi, galat_akurasi);
        iterasi += 1;
    }
    println!();
    for i in 0..spl.ukuran {
        println!("x{} = {}", i+1, spl.solusi[i]);
    }
    println!();
    for i in 0..spl.ukuran {
        println!("Persamaan {} = {}\n Dengan galat: {}", i+1, spl.eval(i), (spl.eval(i) - spl.konstan[i]).abs());
    }
    println!("\n Jumlah total iteras: {iterasi}");
}
```

Hasil perhitungan untuk SPL 1 dan SPL 2 adalah sebagai berikut

(a) SPL 1:

```
Galat iterasi =
0.00001

Galat mutlak =
0.00001

Batas iterasi=
1000


Metode Iterasi Jacobi:

x1 = -0.9999986987249274
x2 = 2.0000026804088633
x3 = 2.999998838108366

Persamaan 1 = -1.9999999384752791
Dengan galat: 0.00000006152472087350702
Persamaan 2 = 4.00001318480216
Dengan galat: 0.000013184802160282061
Persamaan 3 = 10.000000496009033
Dengan galat: 0.000004960090329575451

Jumlah total iteras: 15


Metode Gauss-Seidel

x1 = -1.000001147335661
x2 = 2.000000515913434
x3 = 3.0000002104740755

Persamaan 1 = -2.000003747446341
```

```
Dengan galat: 0.0000037474463407960457
Persamaan 2 = 4.000000705844
Dengan galat: 0.000000705844000137290 1
Persamaan 3 = 10
Dengan galat: 0

Jumlah total iteras: 9
```

Diperoleh hasil hampirannya adalah $x_1 = -1.000..., x_2 = 2.000..., x_3 = 3.000...$ untuk kedua metode. Namun metode iteratif Jacobi membutuhkan lebih banyak iterasidibanding metode Gauss-Seidel.

(b) SPL 2:

```
Galat iterasi =
0.00001

Galat mutlak =
0.00001

Batas iterasi=
1000


Metode Iterasi Jacobi:

x1 = -0.22432191178776772
x2 = 0.1824301230830861
x3 = 0.7094648820885382
x4 = -0.25810671310530897

Persamaan 1 = 0.000036048835940771795
Dengan galat: 0.000036048835940771795
Persamaan 2 = 0.9999758282943191
Dengan galat: 0.000024171705680919864
Persamaan 3 = 3.0000405283222253
Dengan galat: 0.000040528322225341 64
Persamaan 4 = 2.0000049347438758
Dengan galat: 0.000004934743875076 9

Jumlah total iteras: 118


Metode Gauss-Seidel

x1 = -0.22432157836388256
x2 = 0.18243012118314197
x3 = 0.7094629084798579
x4 = -0.25810539785474096

Persamaan 1 = 0.000035424002173600755
Dengan galat: 0.000035424002173600755
Persamaan 2 = 0.9999758211501367
Dengan galat: 0.000024178849863276675
Persamaan 3 = 3.000034281096301
Dengan galat: 0.00003428109630121412
Persamaan 4 = 2
Dengan galat: 0
```

```
    Jumlah total iteras: 15
```

Diperoleh hasil hampirannya adalah $x_1 = -0.224..., x_2 = 0.182..., x_3 = 0.709..., x_4 = -0.258...$ untuk kedua metode. Sama seperti pada SPL 1, jumlah iterasi metode Gauss-Seidel lebih sedikit dibanding metode iteratif Jacobi.