1. Tiesinių lygčių sprendimas

1.1. Užduotis

Duota tiesinių lygčių sistema [A][X] = [B] ir jos sprendimui nurodytas Gauso-Zeidelio metodas.

$$\begin{cases} 9x_1 + 3x_2 - x_3 + 2x_4 = 65 \\ 3x_1 + 11x_2 - 2x_3 - 2x_4 = 27 \\ -x_1 - 2x_2 + 6x_3 - x_4 = -23 \\ 2x_1 - 2x_2 - x_3 + 9x_4 = 39 \end{cases}$$

1.2. Lygčių sistemos sprendimas

Lygčių sistemos sprendimui panaudotas Gauso-Zeidelio metodas:

$$x_i^{(k+1)} = \frac{1}{\alpha_i}\left(\tilde{b}_i - \sum_{j=1}^{i-1} \tilde{a}_{ij} x_j^{(k+1)} - \sum_{j=i}^{n} \tilde{a}_{ij} x_j^{(k)}\right), \quad i = 1:n$$

Pradinės *alfa* reikšmės [1, 1, 1, 1].

Gautas rezultatas:

```
X1 = 6.13607
X2 = 0.862131
X3 = -2.09912
X4 = 2.54494
```

Rezultatas gautas MatLab aplinkoje:

```
X1 = 6.1361
X2 = 0.8621
X3 = -2.0991
X4 = 2.5449
```

Įsistačius gautus atsakymus į pradinę lygtį gaunami tokie atsakymai:

Gautas rez:	Tikimasi:
65	65
27	27
-23	-23
35.5515	39

Gauti rezultatai minimaliai skiriasi nuo MatLab aplinkoje gautų rezultatų. O gautus rezultatus panaudojus pradinėje lygčių sistemoje tik viena lygtis turi klaidingą atsakymą. Tad galima teigti, kad gautos x reikšmės yra teisingos.

1.3. Kodas

```
public void GausoZeidelioMetodas()
       {
            Matrix<double> M = Matrix<double>.Build.DenseOfArray(new double[,] {
                { 9,  3, -1,  2},
                { 3, 11, -2, -2},
                {-1, -2,  6, -1},
                { 2,  2, -1,  9}
            });
            Vector<double> B = Vector<double>.Build.DenseOfArray(new double[]
            {
                65, 27, -23, 39
            });

            Vector<double> alpha = Vector<double>.Build.DenseOfArray(new double[]
            {
                1, 1, 1, 1
            });
            int n = 4;

            var atld = Matrix<double>.Build
                .DenseOfDiagonalVector(M.Diagonal().DivideByThis(1))
                .Multiply(M)
                .Subtract(Matrix<double>.Build.DenseOfDiagonalVector(alpha));

            var btld = Matrix<double>.Build
                .DenseOfDiagonalVector(M.Diagonal().DivideByThis(1))
                .Multiply(B);

            var x = Vector<double>.Build.DenseOfArray(new double[] { 0, 0, 0, 0 });
            var x1 = x.Clone();

            for (int i = 0; i < maxIteraciju; i++)
            {
                for (int j = 0; j < n; j++)
                {
                    x1[j] = (btld[j] - atld.Row(j) * x1) / alpha[j];
                }

                form.richTextBox1.AppendText(x1.ToString());
                var tikslumas = (x1 - x).Norm(2) / (x.Norm(2) + x1.Norm(2));

                if (tikslumas < eps)
                {
                    return;
                }

                x = x1.Clone();
            }
        }
```

2. Netiesiunių lygčių sprendimas

2.1. Užduotis

Duotos netiesinių lygčių sistemos:

$$\begin{cases} \dfrac{10x_1}{x_2^2+1} + x_1^2 - x_2^2 = 0 \\ x_1^2 + 2x_2^2 - 32 = 0 \end{cases}$$

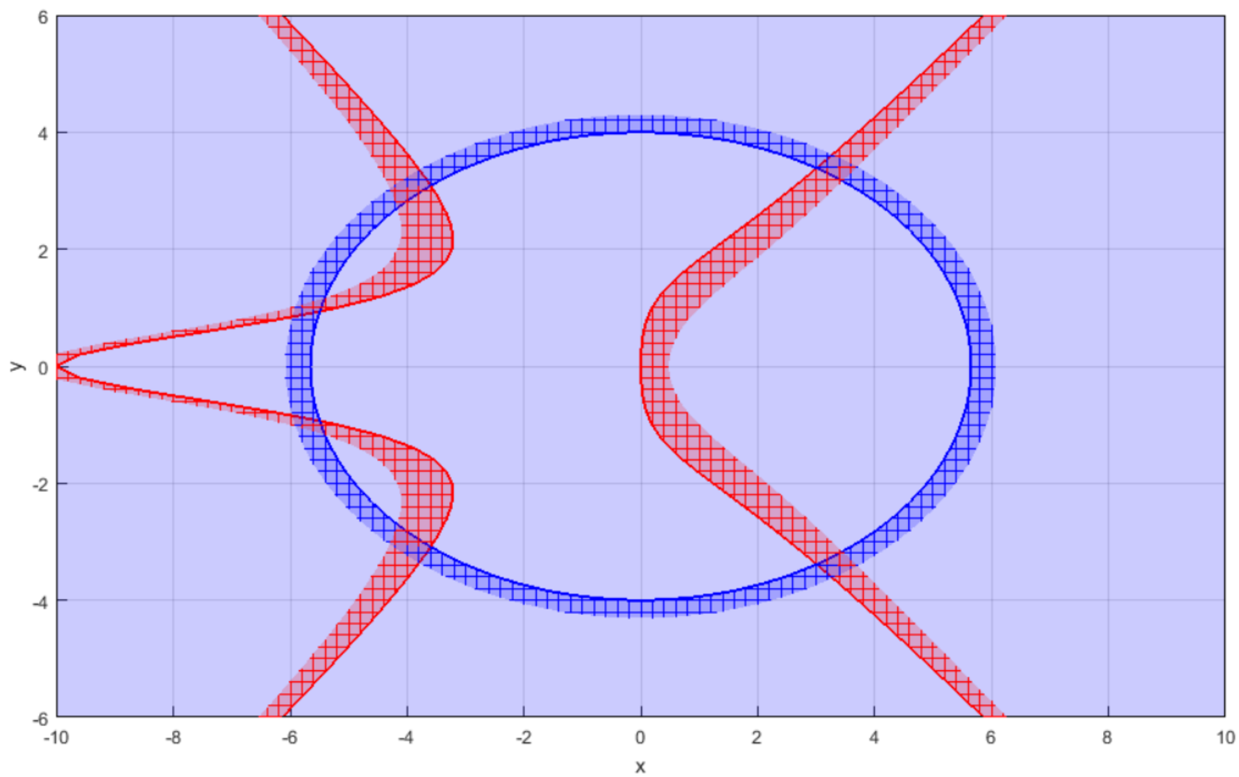
ir

$$\begin{cases} x_1 + 4x_2 + x_3 - 22 = 0 \\ x_2 x_3 - 2x_3 - 18 = 0 \\ -x_2^2 + 2x_4^3 - 3x_1 x_4 + 335 = 0 \\ 2x_3 - 12x_2 + 2x_4 + 58 = 0 \end{cases}$$

Niutono metodas.

2.2. Lygčių sistemų sprendimas

Pirmos lygčių sistemos grafinis atvaizdavimas:



Išvados: Iš lygčių sistemos sprendimo grafiniu būdu vaizdo matome, kad plokštumų kreivės plokštumoje z = 0 susikerta 6 vietose, todėl iš šio vaizdo galima daryti prielaidą, kad ši lygčių sistema galimai turės 6 skirtingas šaknis.

Skaitiniai metodai ir algoritmai (P170B115). Nerijus Dulkė, IFF-6/11, Varianto nr. 15

Pirmos lygčių sistemos sprendimas Niutono metodu:

Pradinis artinys	Sprendinys	Iteracijų skaičius	MatLab sprendinys
[1,-1]	[3.00962042991304, -3.38690012163409]	7	
[1,1]	[3.00962042991304, 3.38690012163409]	7	
[-2,-2]	[-3.59793171343044, -3.08665574574583]	6	
[-4,-1]	[-5.50055533047788, -0.933780235481406]	5	
[-4,1]	[-5.50055533047788, 0.933780235481406]	5	
[-2,2]	[-3.59793171343044, 3.08665574574583]	6	

Išvados: Iš rezultatų lentelės matome, jog Niutono metodu rastos tokios šaknys, kokios buvo matomos ir grafiniame lygties sprendimo variante.

Antros lygčių sistemos sprendimas Niutono metodu:

Pradinis artinys	Sprendinys	Iteracijų skaičius	MatLab sprendinys
X1 = 1 X2 = 1 X3 = -1 X4 = -1	X1 = 41.7591559390323 X2 = 0.0228401465215402 X3 = -19.8505165251185 X4 = -9.01244259575228	20	

Kodas:

```
public void NiutonoMetodas(bool isF2)
        {
            double F1_1(double[] p) => 10 * p[0] / (p[1] * p[1] + 1) + p[0] * p[0] - p[1] 
* p[1];
            double F1_2(double[] p) => p[0] * p[0] + 2 * p[1] * p[1] - 32;
            double F2_1(double[] p) => p[0] + 4 * p[1] + p[2] - 22;
            double F2_2(double[] p) => p[1] * p[2] * 2 * p[2] - 18;
            double F2_3(double[] p) => -1 * p[1] * p[1] + 2 * Math.Pow(p[3], 3) - 3 * 
p[0] * p[3] + 335;
            double F2_4(double[] p) => 2 * p[2] - 12 * p[1] + 2 * p[3] + 58;

            double a = 1;
            var x = !isF2
                ? new[] {1d, -1d}
                : new[] {1d, 1d, -1d, -1d};

            for (int i = 0; i < 500; i++)
            {
                form.richTextBox1.AppendText($"Iteracija: {i + 1}\n");

                var final = x.ToArray();
                var func = !isF2
                    ? new[] {F1_1(x), F1_2(x)}
                    : new[] {F2_1(x), F2_2(x), F2_3(x), F2_4(x)};
```

```
                // Jakobo matrica
                var jacMatrix = !isF2
                    ? new NumericalJacobian()
                        .Evaluate(new Func<double[], double>[] {F1_1, F1_2}, x)
                        .ToMatrix()
                    : new NumericalJacobian()
                        .Evaluate(new Func<double[], double>[] {F2_1, F2_2, F2_3, F2_4},
x)
                        .ToMatrix();


                var deltaX = jacMatrix.Solve(func.ToVector());
                x = (final.ToVector() - a * deltaX).ToArray();

                if (!isF2)
                {
                    form.richTextBox1.AppendText($"x:{x[0],20}, y:{x[1],20}\n");
                }
                else
                {
                    form.richTextBox1.AppendText($"x1:{x[0],20}, x2:{x[1],20},
x3:{x[2],20}, x4:{x[3],20}\n");
                }

                if ((x.ToVector() - final.ToVector()).Norm(2) < 1e-8)
                    break;
            }
        }
```

3. Optimizavimas