```
using System;
using System.Collections.Generic;
using System.Linq;

using MathNet.Numerics.Data.Matlab;
using static MathNet.Numerics.LinearAlgebra.CreateMatrix;
using static MathNet.Numerics.LinearAlgebra.CreateVector;
using mat = MathNet.Numerics.LinearAlgebra.Matrix<double>;
using vec = MathNet.Numerics.LinearAlgebra.Vector<double>;
using MathNet.Numerics.LinearAlgebra.Double.Solvers;
using MathNet.Numerics.LinearAlgebra.Solvers;
using MathNet.Numerics.LinearAlgebra;

namespace MMP
{
    public struct Vertex1
    {
        public double X;
        public int I;
    }

    public struct Interval1
    {
        public double A, B;
        public double StepSize(int n)
        {
            return (B - A) / (n - 1);
        }
        public IEnumerable<Vertex1> Grid(int n)
        {
            double h = StepSize(n);
```

```
        double x = A;
        for (int i = 0; i < n; i++)
        {
            yield return new Vertex1 { X = x, I = i };
            x += h;
        }
    }
}
public struct Vertex
{
    public double X, Y;
    public int Ix, Iy;
}

public struct Edge
{
    public int A { get; }
    public int B { get; }
    public Edge(int A, int B)
    {
        if (A < B)
        {
            this.A = A;
            this.B = B;
        }
        else
        {
            this.A = B;
            this.B = A;
        }
    }
```

```
}
public struct Triangle
{
    public int A, B, C;
    public IEnumerable<Edge> Edges
    {
        get
        {
            yield return new Edge(A, B);
            yield return new Edge(B, C);
            yield return new Edge(C, A);
        }
    }
}
class Program
{
    static void _Main(string[] args)
    {
        var tri = new Triangle[]
        {
               new Triangle { A=0, B=1, C=3},
               new Triangle { A=1, B=2, C=3}
        };

        var q = from T in tri
                select T.Edges;

        IEnumerable<Edge> U = new Edge[0];
        foreach (var item in q)
        {
            foreach (var e in item)
```

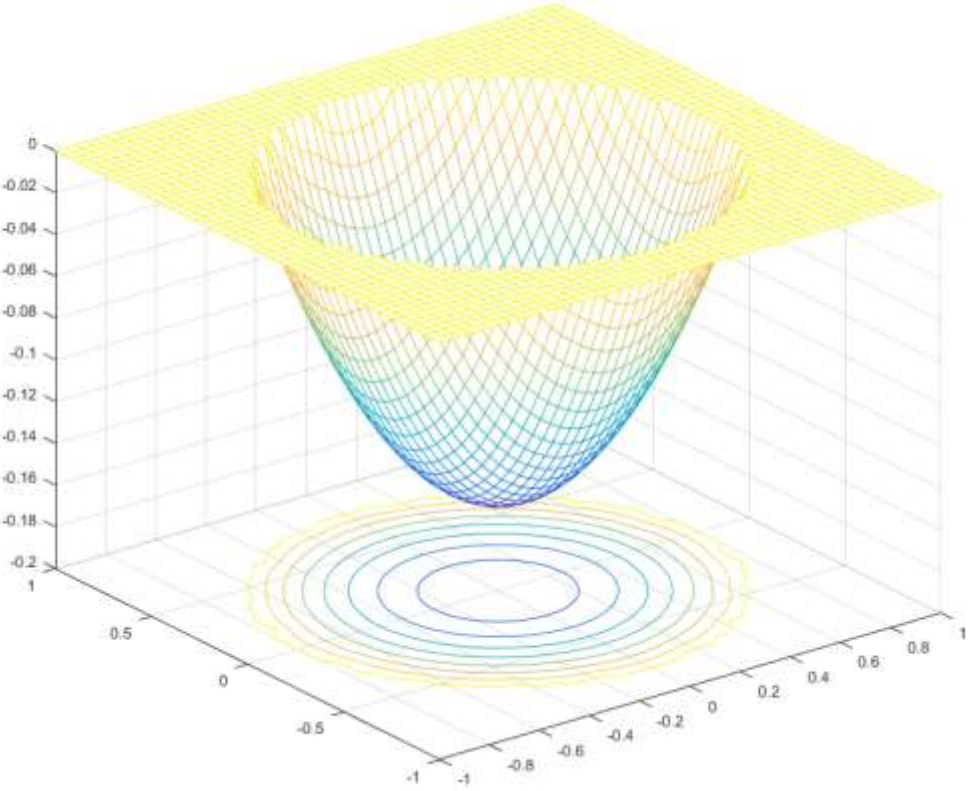
```
        {
            Console.WriteLine($"{e.A} {e.B}");
        }
        U = U.Union(item);
    }

    Console.WriteLine();
    foreach (var e in U)
    {
        Console.WriteLine($"{e.A} {e.B}");
    }
    Console.WriteLine();

    var e1 = new Edge(0, 1);
    var e2 = new Edge(0, 1);
    var b = e1.Equals(e2);
    Console.WriteLine(b);
}

static void Main()
{
    int n = 51;
    var I = new Interval1 { A = -1, B = 1 };

    Func<double, double, double> f = (x1, x2) => x1 * x1 + x2 * x2;

    var Omega = from x in I.Grid(n)
                from y in I.Grid(n)
                where f(x.X, y.X) < 0.6
                select new Vertex { X = x.X, Y = y.X, Ix = x.I, Iy = y.I };
```

```
var Box = from x in I.Grid(n)
          from y in I.Grid(n)
          select new Vertex { X = x.X, Y = y.X, Ix = x.I, Iy = y.I };

int np = Omega.Count();
int[,] Dof = new int[n, n];
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        Dof[i, j] = -1;

int ix = 0;
foreach (var v in Omega)
{
    Dof[v.Ix, v.Iy] = ix++;
}

mat A = Sparse<double>(np, np);
int[] stencil = new int[4];

for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
    {
        int d = Dof[i, j];
        if (d > -1)
        {
            A[d, d] = 4;
            stencil[0] = Dof[i - 1, j];
            stencil[1] = Dof[i + 1, j];
            stencil[2] = Dof[i, j - 1];
            stencil[3] = Dof[i, j + 1];
            foreach (var nb in stencil)
```

```
                if (nb > -1) A[d, nb] = -1;
            }
        }

    mat X = Dense<double>(n, n);
    mat Y = Dense<double>(n, n);
    mat Z = Dense<double>(n, n);

    foreach (var v in Box)
    {
        X[v.Ix, v.Iy] = v.X;
        Y[v.Ix, v.Iy] = v.Y;
    }

    //Write(A);
    //MatlabWriter.Write("A.mat", A, "A");

    const double ConvergenceBoundary = 1e-5;
    const int MaximumIterations = 1000;


    var monitor = new Iterator<double>(
        new IterationCountStopCriterion<double>(MaximumIterations),
        new ResidualStopCriterion<double>(ConvergenceBoundary),
        new DivergenceStopCriterion<double>(),
        new FailureStopCriterion<double>());
    var solver = new BiCgStab();

    double h = I.StepSize(n);
    vec b = Vector<double>.Build.Dense(A.RowCount, -h * h);
    var u = A.SolveIterative(b, solver, monitor);
```

```
            for (int i = 0; i < n; i++)
                for (int j = 0; j < n; j++)
                {
                    if (Dof[i, j] > -1)
                        Z[i, j] = u[Dof[i, j]];
                }

            mat[] data = { X, Y, Z };
            string[] names = { "X", "Y", "Z" };
            MatlabWriter.Write("poisson.mat", data, names);

        }
    }
}
```

```
clear;clc;close;
load poisson.mat
meshc(X,Y,Z);
```


0
-0.02
-0.04
-0.06
-0.08
-0.1
-0.12
-0.14
-0.16
-0.18
-0.2
1
0.5
0
-0.5
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1