Vedlegg G: Kode av FDM Flexible 3D

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

using Grasshopper.Kernel;
using Grasshopper.Kernel.Types;
using MathNet.Numerics.LinearAlgebra;
using Rhino.Geometry;

namespace FormFinding
{
    public class NewC_Matrix : GH_Component
    {
        /// <summary>
        /// Initializes a new instance of the NewC_Matrix class.
        /// </summary>
        public NewC_Matrix()
          : base("NewC_Matrix", "FDM",
              "Flexible FDM",
              "Form Finding", "Force Density Method")
        {
        }

        /// <summary>
        /// Registers all the input parameters for this component.
        /// </summary>
        protected override void RegisterInputParams(GH_Component.GH_InputParamManager
pManager)
        {
            pManager.AddLineParameter("Edge Lines", "Le", "Edge Lines",
GH_ParamAccess.list);
            pManager.AddLineParameter("Interoir Lines", "Li", "interior Lines",
GH_ParamAccess.list);
            pManager.AddNumberParameter("Force in z-direction", "Pz", "Force in z-
direction", GH_ParamAccess.item);
            pManager.AddNumberParameter("Force Density", "q", "Force Density",
GH_ParamAccess.item);
            pManager.AddIntegerParameter("Openings", "O", "Openings",
GH_ParamAccess.item);
        }

        /// <summary>
        /// Registers all the output parameters for this component.
        /// </summary>
        protected override void
RegisterOutputParams(GH_Component.GH_OutputParamManager pManager)
        {
            pManager.AddPointParameter("New points", "Pn", "New points",
GH_ParamAccess.list);
            pManager.AddPointParameter("New points", "Pf", "New points",
GH_ParamAccess.list);
            pManager.AddLineParameter("New Lines", "Ni", "New Lines",
GH_ParamAccess.list);
            pManager.AddLineParameter("New Lines", "NL", "New Lines",
GH_ParamAccess.list);
            pManager.AddPointParameter("New points", "Pf", "New points",
GH_ParamAccess.list);
        }
        /// <summary>
        /// This is the method that actually does the work.
        /// </summary>
        /// <param name="DA">The DA object is used to retrieve from inputs and store
in outputs.</param>
        protected override void SolveInstance(IGH_DataAccess DA)
        {
            List<Line> Le = new List<Line>();                              ///
Inndata defineres
            if (!DA.GetDataList(0, Le)) { return; }

            List<Line> Li = new List<Line>();
            if (!DA.GetDataList(1, Li)) { return; }

            double F = double.NaN;
            if (!DA.GetData(2, ref F)) { return; }

            double q = double.NaN;
            if (!DA.GetData(3, ref q)) { return; }

            int O = new int();
            if (!DA.GetData(4, ref O)) { return; }

            List<Point3d> P = new List<Point3d>();
            for (int i = 0; i < Li.Count; i++)
            {
                P.Add(Li[i].PointAt(0));
                P.Add(Li[i].PointAt(1));
            }
            for (int i = 0; i < Le.Count; i++)
            {
                P.Add(Le[i].PointAt(0));
                P.Add(Le[i].PointAt(1));
            }
            List<Point3d> P0R = new List<Point3d>();
            for (int i = 0; i < P.Count; i++)
            {
                var PRc0 = P0R.Count;
                for (int j = i + 1; j < P.Count; j++)
                {
                    var PRc1 = P0R.Count;
                    if (P[i].Equals(P[j]))
                    {
                        if (PRc0 == PRc1)
                        {
                            P0R.Add(P[j]);
                        }
                    }
                }
            }
            for (int i = 0; i < P0R.Count; i++)
            {
                P.Remove(P0R[i]);
            }

            var M = Matrix<double>.Build;

            var C = M.Dense(Le.Count + Li.Count, P.Count, 0);              /// C-
matrise settes opp
```

```
            for (int i = Le.Count; i < (Le.Count + Li.Count); i++)          /// Cn 
delen
            {
                for (int j = 0; j < P.Count; j++)
                {
                    if (Li[i-Le.Count].PointAt(0) == P[j])                  /// Ved 
startpunktet settes verdien lik -1
                    {
                        C[i, j] = -1;
                    }
                    if (Li[i-Le.Count].PointAt(1) == P[j])                  /// Ved 
endepunktet settes verdien lik +1
                    {
                        C[i, j] = 1;
                    }
                }
            }
            for (int i = 0; i < Le.Count; i++)
            {
                for (int j = 0; j < P.Count; j++)                           /// Cf 
delen
                {
                    if (Le[i].PointAt(0) == P[j])                           /// Ved 
startpunket settes verdien lik -1
                    {
                        C[i, j] = -1;
                    }
                    if (Le[i].PointAt(1) == P[j])                           /// Ved 
endepunktet settes verdien lik +1
                    {
                        C[i, j] = 1;
                    }
                }
            }

            List<Vector<double>> Vl = new List<Vector<double>>();           /// C 
deles inn i Cn og Cf
            for (int i = 0; i < (P.Count - Le.Count - O); i++)
            {
                Vl.Add(C.Column(i));
            }
            var Cn = M.DenseOfColumnVectors(Vl);
            List<Vector<double>> Vl2 = new List<Vector<double>>();
            for (int i = (P.Count - Le.Count - O); i < P.Count; i++)
            {
                Vl2.Add(C.Column(i));
            }
            var Cf = M.DenseOfColumnVectors(Vl2);

            List<double> x = new List<double>();                            /// xn og 
xf settes opp ut i fra punktene
            for (int i = 0; i < P.Count; i++)
            {
                x.Add(P[i].X);
            }
            List<double> Vxnl = new List<double>();
            for (int i = 0; i < (P.Count - Le.Count - O); i++)
            {
                Vxnl.Add(x[i]);
            }
            var Xn = M.DenseOfColumnMajor(Vxnl.Count, 1, Vxnl.ToArray());
            List<double> Vxfl = new List<double>();
            for (int i = (P.Count - Le.Count - O); i < P.Count; i++)
            {
                Vxfl.Add(x[i]);
            }
            var Xf = M.DenseOfColumnMajor(Vxfl.Count, 1, Vxfl.ToArray());

            List<double> y = new List<double>();                            /// yn og
yf settes opp ut i fra punktene
            for (int i = 0; i < P.Count; i++)
            {
                y.Add(P[i].Y);
            }
            List<double> Vynl = new List<double>();
            for (int i = 0; i < (P.Count - Le.Count - O); i++)
            {
                Vynl.Add(y[i]);
            }
            var Yn = M.DenseOfColumnMajor(Vynl.Count, 1, Vynl.ToArray());
            List<double> Vyfl = new List<double>();
            for (int i = (P.Count - Le.Count - O); i < P.Count; i++)
            {
                Vyfl.Add(y[i]);
            }
            var Yf = M.DenseOfColumnMajor(Vyfl.Count, 1, Vyfl.ToArray());

            List<double> z = new List<double>();                            /// zn og
zf settes opp ut i fra punktene
            for (int i = 0; i < P.Count; i++)
            {
                z.Add(P[i].Z);
            }
            List<double> Vznl = new List<double>();
            for (int i = 0; i < (P.Count - Le.Count - O); i++)
            {
                Vznl.Add(z[i]);
            }
            var Zn = M.DenseOfColumnMajor(Vznl.Count, 1, Vznl.ToArray());
            List<double> Vzfl = new List<double>();
            for (int i = (P.Count - Le.Count - O); i < P.Count; i++)
            {
                Vzfl.Add(z[i]);
            }
            var Zf = M.DenseOfColumnMajor(Vzfl.Count, 1, Vzfl.ToArray());

            List<double> px = new List<double>();                           /// Last i
x-retning settes lik 0
            for (int i = 1; i <= Xn.RowCount; i++)
            {
                px.Add(0);
            }
            var Px = M.DenseOfColumnMajor(px.Count, 1, px.ToArray());

            List<double> py = new List<double>();                           /// Last i
y-retning settes lik 0
            for (int i = 1; i <= Yn.RowCount; i++)
            {
                py.Add(0);
            }
            var Py = M.DenseOfColumnMajor(py.Count, 1, py.ToArray());

            List<double> pz = new List<double>();                           /// Last i
z-retning settes lik F
```

```
            for (int i = 1; i <= Yn.RowCount; i++)
            {
                pz.Add(F);
            }
            var Pz = M.DenseOfColumnMajor(pz.Count, 1, pz.ToArray());

            var X = M.DenseOfColumnMajor(x.Count, 1, x.ToArray());          /// X-
koordinatene som matrise
            var Y = M.DenseOfColumnMajor(y.Count, 1, y.ToArray());          /// Y-
koordinatene som matrise
            var Z = M.DenseOfColumnMajor(z.Count, 1, z.ToArray());          /// Z-
koordinatene som matrise
            var u0 = C * X;                                                 ///
Utregning av l0
            var u1 = u0.ToColumnMajorArray();
            var u2 = u0.RowCount;
            for (int i = 0; i < u2; i++)
            {
                u1[i] = Math.Abs(u1[i]);
            }
            var U0 = M.DenseOfDiagonalArray(u1.Length, u1.Length, u1);

            var v0 = C * Y;
            var v1 = v0.ToColumnMajorArray();
            var v2 = v0.RowCount;
            for (int i = 0; i < v2; i++)
            {
                v1[i] = Math.Abs(v1[i]);
            }
            var V0 = M.DenseOfDiagonalArray(v1.Length, v1.Length, v1);

            var w0 = C * Z;
            var w1 = w0.ToColumnMajorArray();
            var w2 = w0.RowCount;
            for (int i = 0; i < w2; i++)
            {
                w1[i] = Math.Abs(w1[i]);
            }
            var W0 = M.DenseOfDiagonalArray(w1.Length, w1.Length, w1);

            var L0 = (U0.PointwisePower(2) + V0.PointwisePower(2) +
W0.PointwisePower(2)).PointwisePower(0.5);
            var l0 = L0.Diagonal().ToColumnMatrix();

            var Q = M.DenseDiagonal(Le.Count+Li.Count, Le.Count + Li.Count, q); /// Q
settes opp med diagonalen lik q

            var CnT = (Cn.Transpose());                                     /// Cn
transponeres

            var Dn = CnT * Q * Cn;                                          ///
Utregning av Dn, Df og invers av Dn
            var Df = CnT * Q * Cf;
            var Dni = (Dn.Inverse());

            var XN = Dni * (Px - (Df * Xf));                                ///
Utregning av de nye punktene
            var YN = Dni * (Py - (Df * Yf));
            var ZN = Dni * (Pz - (Df * Zf));

            var XNo = XN.Column(0);                                         /// Gjør
matrisene om til vektorer
```

```
            var YNo = YN.Column(0);
            var ZNo = ZN.Column(0);

            var Nc = XNo.Count;

            List<Point3d> newpoints3 = new List<Point3d>();
            List<Point3d> newpoints = new List<Point3d>();                  /// Utdata
#0, de nye punktene
            for (int i = 0; i < Nc; i++)
            {
                newpoints.Add(new Point3d(XNo[i], YNo[i], ZNo[i]));
                newpoints3.Add(new Point3d(XNo[i], YNo[i], ZNo[i]));
            }

            List<Point3d> newpoints2 = new List<Point3d>();                 /// Utdata
#1, opplagerpunktene
            for (int i = 0; i < Xf.RowCount; i++)
            {
                newpoints2.Add(new Point3d(Xf.Column(0)[i], Yf.Column(0)[i],
Zf.Column(0)[i]));
                newpoints3.Add(new Point3d(Xf.Column(0)[i], Yf.Column(0)[i],
Zf.Column(0)[i]));
            }

            List<Line> NLi = new List<Line>();                              /// Utdata
#2, indre linjer
            for (int i = 0; i < Li.Count; i++)
            {
                for (int j = 0; j < P.Count; j++)
                {
                    if (Li[i].PointAt(0) == P[j])
                    {
                        for (int k = 0; k < P.Count; k++)
                        {
                            if (Li[i].PointAt(1) == P[k])
                            {
                                NLi.Add(new Line(newpoints3[j], newpoints3[k]));
                            }
                        }
                    }
                }
            }

            List<Line> NLe = new List<Line>();                              /// Utdata
#3, ytre linjer
            for (int i = 0; i < Le.Count; i++)
            {
                for (int j = 0; j < P.Count; j++)
                {
                    if (Le[i].PointAt(0) == P[j])
                    {
                        for (int k = 0; k < P.Count; k++)
                        {
                            if (Le[i].PointAt(1) == P[k])
                            {
                                NLe.Add(new Line(newpoints3[j], newpoints3[k]));
                            }
                        }
                    }
                }
            }
            DA.SetDataList(0, newpoints);                                   /// Utdata
defineres
            DA.SetDataList(1, newpoints2);
            DA.SetDataList(2, NLi);
            DA.SetDataList(3, NLe);
            DA.SetDataList(4, P);
        }

        /// <summary>
        /// Provides an Icon for the component.
        /// </summary>
        protected override System.Drawing.Bitmap Icon
        {
            get
            {
                //You can add image files to your project resources and access them
like this:
                // return Resources.IconForThisComponent;
                return null;
            }
        }

        /// <summary>
        /// Gets the unique ID for this component. Do not change this ID after
release.
        /// </summary>
        public override Guid ComponentGuid
        {
            get { return new Guid("c49213d6-9d31-49cf-8e77-00a3c12bdca1"); }
        }
    }
}
```