Vedlegg H: Kode av DR 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 Flexible_DR : GH_Component
    {
        /// <summary>
        /// Initializes a new instance of the Flexible_DR class.
        /// </summary>
        public Flexible_DR()
          : base("Flexible_DR", "DR",
              "Flexible Dynamic Relaxation",
              "Form Finding", "Dynamic Relaxation")
        {
        }

        /// <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", "A", "Force in z-
direction", GH_ParamAccess.item);
            pManager.AddNumberParameter("Iterations", "I", "Number of iterations",
GH_ParamAccess.item);
            pManager.AddNumberParameter("Force in z-direction", "EA", "Force in z-
direction", GH_ParamAccess.item);
            pManager.AddNumberParameter("Force in z-direction", "EAv", "Force in z-
direction", GH_ParamAccess.list);
            pManager.AddNumberParameter("Force in z-direction", "f0", "Force in z-
direction", GH_ParamAccess.list);
        }

        /// <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);
        }

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

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

            double EA = double.NaN;
            if (!DA.GetData(4, ref EA)) { return; }

            List<double> EAn = new List<double>();
            if (!DA.GetDataList(5, EAn)) { return; }

            List<double> f0n = new List<double>();
            if (!DA.GetDataList(6, f0n)) { return; }

            List<Point3d> Pe = new List<Point3d>();
            for (int i = 0; i < Le.Count; i++)
            {
                Pe.Add(Le[i].PointAt(0));
                Pe.Add(Le[i].PointAt(1));
            }
            List<Point3d> PeR = new List<Point3d>();
            for (int i = 0; i < Pe.Count; i++)
            {
                var PeRc0 = PeR.Count;
                for (int j = i + 1; j < Pe.Count; j++)
                {
                    var PeRc1 = PeR.Count;
                    if (Pe[i] == Pe[j])
                    {
                        if (PeRc0 == PeRc1)
                        {
                            PeR.Add(Pe[j]);
                        }
                    }
                }
            }
            for (int i = 0; i < PeR.Count; i++)
            {
                Pe.Remove(PeR[i]);
            }
            List<Point3d> Pi = new List<Point3d>();
            for (int i = 0; i < Li.Count; i++)
            {
```

```
                Pi.Add(Li[i].PointAt(0));
                Pi.Add(Li[i].PointAt(1));
            }
            List<Point3d> PiR = new List<Point3d>();
            for (int i = 0; i < Pi.Count; i++)
            {
                var PiRc0 = PiR.Count;
                for (int j = i+1; j < Pi.Count; j++)
                {
                    var PiRc1 = PiR.Count;
                    if (Pi[i] == Pi[j])
                    {
                        if (PiRc0 == PiRc1)
                        {
                            PiR.Add(Pi[j]);
                        }
                    }
                }
            }
            for (int i = 0; i < PiR.Count; i++)
            {
                Pi.Remove(PiR[i]);
            }
            List<Point3d> P = new List<Point3d>();
            for (int i = 0; i < Pi.Count; i++)
            {
                P.Add(Pi[i]);
            }
            for (int i = 0; i < Pe.Count; i++)
            {
                P.Add(Pe[i]);
            }
            List<Point3d> P0R = new List<Point3d>();
            for (int i = 0; i < P.Count; i++)
            {
                for (int j = i + 1; j < P.Count; j++)
                {
                    if (P[i] == P[j])
                    {
                        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 = 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;
                    }
                }
            }
            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
startpunket 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;
                    }
                }
            }
            List<Vector<double>> Vl = new List<Vector<double>>();           /// C
deles inn i Cn og Cf
            for (int i = 0; i < (P.Count - Le.Count); 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); 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); 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); 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); 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); 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); 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); i < P.Count; i++)
            {
                Vzfl.Add(z[i]);
            }
            var Zf = M.DenseOfColumnMajor(Vzfl.Count, 1, Vzfl.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 L
            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 L = L0;

                                                                                ///
Definering av krefter og stivhet
            var f0 = M.DenseOfColumnMajor(Le.Count + Li.Count, 1, f0n.ToArray());
            var EAV = M.DenseOfColumnMajor(Le.Count + Li.Count, 1, EAn.ToArray());
            var e = (L - L0) * (L0.Inverse());

            var q = L.Inverse() * (f0 + e * EAV);                               /// Q
settes opp med diagonalen lik q
            var Q = M.DenseOfDiagonalArray(Le.Count + Li.Count, Le.Count + Li.Count,
q.Column(0).AsArray());

            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 pz = 0.5 * A * Cn.Transpose().PointwiseAbs() * l0;           ///
Utregning av krefter
            var pz = M.Dense(Xn.RowCount, 1, A);
            var rz = pz - Dn * Zn - Df * Zf;
            var rx = -Dn * Xn - Df * Xf;
            var ry = -Dn * Yn - Df * Yf;

            var dt = 1;                                                        ///
Utregning av fart
            var m = 0.5 * dt * Cn.Transpose().PointwiseAbs() * (L.Inverse() * f0 +
L0.Inverse() * f0);
            var C0 = 0.5;
            var A0 = (1 - C0 / 2) / (1 + C0 / 2);
            var B0 = (1 + A0) / 2;
            var M0 = M.DenseOfDiagonalArray(m.Column(0).AsArray());
            var v00 = M.Dense(Xn.RowCount, 1, 0);

            var v10 = A0 * v00 + B0 * dt * M0.Inverse() * rz;
            var z1 = Zn + dt * v10;
            var v11 = A0 * v00 + B0 * dt * M0.Inverse() * rx;
            var x1 = Xn + dt * v11;
            var v12 = A0 * v00 + B0 * dt * M0.Inverse() * ry;
            var y1 = Yn + dt * v12;

            var rrr = rz;
            var vvv = v10;
            var zzz = z1;
            var rrrx = rx;
            var vvvx = v11;
            var xxx = x1;
            var rrry = ry;
            var vvvy = v12;
            var yyy = y1;
            List<double> xx = new List<double>();
            List<double> yy = new List<double>();
            List<double> zz = new List<double>();
```

```
            for (int j = 1; j < it; j++)                                        ///
Iterasjoner der koordinatene oppdateres
            {
                xx.Clear();
                yy.Clear();
                zz.Clear();

                for (int i = 0; i < xxx.Column(0).Count; i++)
                {
                    xx.Add(xxx.Column(0)[i]);
                }
                for (int i = 0; i < Xf.Column(0).Count; i++)
                {
                    xx.Add(Xf.Column(0)[i]);
                }
                u0 = C * M.DenseOfColumnMajor(xx.Count, 1, xx);
/// Utregning av L
                u1 = u0.ToColumnMajorArray();
                u2 = u0.RowCount;
                for (int i = 0; i < u2; i++)
                {
                    u1[i] = Math.Abs(u1[i]);
                }
                U0 = M.DenseOfDiagonalArray(u1.Length, u1.Length, u1);

                for (int i = 0; i < yyy.Column(0).Count; i++)
                {
                    yy.Add(yyy.Column(0)[i]);
                }
                for (int i = 0; i < Yf.Column(0).Count; i++)
                {
                    yy.Add(Yf.Column(0)[i]);
                }
                v0 = C * M.DenseOfColumnMajor(yy.Count, 1, yy);
                v1 = v0.ToColumnMajorArray();
                v2 = v0.RowCount;
                for (int i = 0; i < v2; i++)
                {
                    v1[i] = Math.Abs(v1[i]);
                }
                V0 = M.DenseOfDiagonalArray(v1.Length, v1.Length, v1);

                for (int i = 0; i < zzz.Column(0).Count; i++)
                {
                    zz.Add(zzz.Column(0)[i]);
                }
                for (int i = 0; i < Zf.Column(0).Count; i++)
                {
                    zz.Add(Zf.Column(0)[i]);
                }
                w0 = C * M.DenseOfColumnMajor(zz.Count, 1, zz);
                w1 = w0.ToColumnMajorArray();
                w2 = w0.RowCount;
                for (int i = 0; i < w2; i++)
                {
                    w1[i] = Math.Abs(w1[i]);
                }
                W0 = M.DenseOfDiagonalArray(w1.Length, w1.Length, w1);

                L = (U0.PointwisePower(2) + V0.PointwisePower(2) +
W0.PointwisePower(2)).PointwisePower(0.5);
```

```
                //e = (L - L0) * (l0.PointwisePower(-1));
                e = (L - L0) * (L0.Inverse());

                q = L.Inverse() * (f0 + e * EAV);                               /// Q
settes opp med diagonalen lik q
                Q = M.DenseOfDiagonalArray(Le.Count + Li.Count, Le.Count + Li.Count,
q.Column(0).AsArray());

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

                m = 0.5 * dt * Cn.Transpose().PointwiseAbs() * (L.Inverse() * f0 +
L0.Inverse() * EAV);

                M0 = M.DenseOfDiagonalArray(m.Column(0).AsArray());

                rrr = pz - Dn * zzz - Df * Zf;
                vvv = A0 * vvv + B0 * dt * M0.Inverse() * rrr;
                zzz = zzz + dt * vvv;
                rrrx = -Dn * xxx - Df * Xf;
                vvvx = A0 * vvvx + B0 * dt * M0.Inverse() * rrrx;
                xxx = xxx + dt * vvvx;
                rrry = -Dn * yyy - Df * Yf;
                vvvy = A0 * vvvy + B0 * dt * M0.Inverse() * rrry;
                yyy = yyy + dt * vvvy;
            }

            var XNo = xxx.Column(0);
            var YNo = yyy.Column(0);
            var ZNo = zzz.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);
        }

        /// <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("d9c4f8f4-0d11-4c32-a86f-c9a298c8b29c"); }
        }
    }
}
```