Vedlegg C: Kode av DR 2D

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

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

namespace FormFinding
{
    public class DRComponent : GH_Component
    {
        /// <summary>
        /// Initializes a new instance of the DRComponent class.
        /// </summary>
        public DRComponent()
          : base("DRComponent", "DR",
              "Hanging chain",
              "Form Finding", "Dynamic Relaxation")
        {
        }

        /// <summary>
        /// Registers all the input parameters for this component.
        /// </summary>
        protected override void RegisterInputParams(GH_Component.GH_InputParamManager
pManager)
        {
            pManager.AddNumberParameter("Distance", "D", "Length of chain",
GH_ParamAccess.item);
            pManager.AddNumberParameter("Segments", "S", "Number of segments",
GH_ParamAccess.item);
            pManager.AddNumberParameter("Iterations", "I", "Number of iterations",
GH_ParamAccess.item);
            pManager.AddNumberParameter("Force", "F", "Force", GH_ParamAccess.item);
            pManager.AddNumberParameter("Iterations", "EA", "Number of iterations",
GH_ParamAccess.item);
            pManager.AddNumberParameter("Internal Force", "f", "Force",
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);
        }

        /// <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)
        {
            double dis = double.NaN;                                        /// Input
#0, avstand mellom punkter
            if (!DA.GetData(0, ref dis)) { return; }

            double seg = double.NaN;                                        /// Input
#1, antall segmenter
            if (!DA.GetData(1, ref seg)) { return; }

            double it = double.NaN;                                        /// Input
#1, antall segmenter
            if (!DA.GetData(2, ref it)) { return; }

            double F = double.NaN;                                        /// Input
#1, antall segmenter
            if (!DA.GetData(3, ref F)) { return; }

            double EA = double.NaN;                                        /// Input
#1, antall segmenter
            if (!DA.GetData(4, ref EA)) { return; }

            double f0n = double.NaN;                                        /// Input
#1, antall segmenter
            if (!DA.GetData(5, ref f0n)) { return; }

            List<double> Xnl = new List<double>();                          ///
Definering av x-verdiene til punktne
            for (double i = (-dis / 2); i <= (dis / 2 + 0.00000000001); i += (dis /
seg))
            {
                Xnl.Add(i);
            }
            var lXn0 = Xnl.Count;

            List<double> Xfl = new List<double>();                          ///
Definering av x-verdiene til opplagerene
            Xfl.Add(Xnl[0]);
            Xfl.Add(Xnl[lXn0 - 1]);
            var lXf = Xfl.Count;

            List<double> Yfl = new List<double>();                          ///
Definering av x-verdiene til opplagerene
            Yfl.Add(0);
            Yfl.Add(0);
            var lYf = Yfl.Count;

            Xnl.RemoveAt(0);
            Xnl.RemoveAt(lXn0 - 2);
            var lXn = Xnl.Count;

            var M = Matrix<double>.Build;

            List<double> d = new List<double>();                            /// Listen
med verdiene som skal brukes i Cn matrsen
            d.Add(1);
            d.Add(1);
            for (int i = 0; i < (lXn - 1); i++)
            {
                for (int j = 0; j < lXn; j++)
```

```
                {
                    d.Add(0);
                }

                d.Add(-1);
                d.Add(1);
            }
            List<double> g = new List<double>();                                /// Listen
med verdiene som skal brukes i Cf matrsen
            g.Add(-1);
            for (int j = 0; j < (lXn * 2); j++)
            {
                g.Add(0);
            }
            g.Add(-1);

            var Cn = M.DenseOfColumnMajor((lXn + 1), lXn, d.ToArray());     ///
Matrisene blir satt opp fra listene som er laget
            var Cf = M.DenseOfColumnMajor((lXn + 1), lXf, g.ToArray());

            var xn = M.DenseOfColumnMajor(lXn, 1, Xnl.ToArray());
            var xf = M.DenseOfColumnMajor(lXf, 1, Xfl.ToArray());
            var yn = M.Dense(lXn, 1, 0);                                        /// y-
koordinatene i punktene starter alle i 0
            var yf = M.Dense(lXf, 1, 0);

            var e0 = new List<double>();
            e0.AddRange(d);
            e0.AddRange(g);
            var Xl = new List<double>();
            Xl.AddRange(Xnl);
            Xl.AddRange(Xfl);
            var C = M.DenseOfColumnMajor((lXn + 1), lXn + lXf, e0.ToArray());
            var x = M.DenseOfColumnMajor(lXn + lXf, 1, Xl.ToArray());
            var u = C * x;
            var u1 = u.ToColumnMajorArray();
            var U = M.DenseOfDiagonalArray(u1.Length, u1.Length,
u.ToColumnMajorArray());
            var u2 = u.RowCount;
            for (int i = 0; i < u2; i++)
            {
                u1[i] = Math.Abs(u1[i]);
            }
            var L0 = M.DenseOfDiagonalArray(lXn + 1, lXn + 1, u1);
            var l0 = M.DenseOfColumnMajor(u1.Length, 1, u1);
            var L = L0;

            var Yl = new List<double>();
            var y = M.DenseOfColumnMajor(lXn + lXf, 1, Yl.ToArray());
            var v = C * y;
            var va = v.ToColumnMajorArray();
            var v2 = v.RowCount;
            for (int i = 0; i < v2; i++)
            {
                va[i] = Math.Abs(va[i]);
            }
            var V = M.DenseOfDiagonalArray(u1.Length, u1.Length,
v.ToColumnMajorArray());

            var f0 = M.Dense(lXn + 1, 1, f0n);
            var EAV = M.Dense(lXn + 1, 1, EA);
            var e = (L - L0) * (l0.PointwisePower(-1));
```

```
            var A = 0.5;

            var q = L.Inverse() * (f0 + EA * e);
            var Q = M.DenseOfDiagonalArray(lXn + 1, lXn + 1, q.Column(0).AsArray());

            var CnT = (Cn.Transpose());                                         ///
Utregninger
            var Dn = CnT * Q * Cn;
            var Df = CnT * Q * Cf;
            var Dni = (Dn.Inverse());

            var py = M.Dense(lXn, 1, F);
            var ry = py - Dn * yn - Df * yf;
            var rx = -Dn * xn - Df * xf;

            var dt = 1;
            var m = 0.5 * dt * Cn.Transpose().PointwiseAbs() * (L.Inverse() * f0 +
L0.Inverse() * EAV);
            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 v0 = M.Dense(lXn, 1, 0);

            var v1 = A0 * v0 + B0 * dt * M0.Inverse() * ry;
            var y1 = yn + dt * v1;
            var v11 = A0 * v0 + B0 * dt * M0.Inverse() * rx;
            var x1 = xn + dt * v11;

            var rrr = ry;
            var vvv = v1;
            var yyy = y1;
            var rrrx = rx;
            var vvvx = v11;
            var xxx = x1;
            for (int j = 1; j < it; j++)
            {
                Xl.Clear();
                Xl.AddRange(xxx.Column(0).ToArray());
                Xl.AddRange(Xfl);
                x = M.DenseOfColumnMajor(Xl.Count, 1, Xl.ToArray());
                u = C * x;
                U = M.DenseOfDiagonalArray(u1.Length, u1.Length,
u.ToColumnMajorArray());

                Yl.Clear();
                Yl.AddRange(yyy.Column(0).ToArray());
                Yl.AddRange(Yfl);
                y = M.DenseOfColumnMajor(Yl.Count, 1, Yl.ToArray());
                v = C * y;
                V = M.DenseOfDiagonalArray(u1.Length, u1.Length,
v.ToColumnMajorArray());

                L = (U.PointwisePower(2) + V.PointwisePower(2)).PointwisePower(0.5);

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

                q = L.Inverse() * (f0 + EA * e);
                Q = M.DenseOfDiagonalArray(lXn + 1, lXn + 1, 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 = py - Dn * yyy - Df * yf;
                vvv = A0 * vvv + B0 * dt * M0.Inverse() * rrr;
                yyy = yyy + dt * vvv;
                rrrx = -Dn * xxx - Df * xf;
                vvvx = A0 * vvvx + B0 * dt * M0.Inverse() * rrrx;
                xxx = xxx + dt * vvvx;
            }

            var SSS = yyy.Column(0);
            var TTT = xxx.Column(0);

            List<Point3d> newpoints = new List<Point3d>();                  /// output
#1, de nye punktene
            for (int j = 0; j < SSS.Count; j++)
            {
                newpoints.Add(new Rhino.Geometry.Point3d(TTT[j], 0, SSS[j]));
            }

            List<Point3d> newpoints2 = new List<Point3d>();                 /// output
#2, opplagerene
            newpoints2.Add(new Rhino.Geometry.Point3d(Xfl[0], 0, 0));
            newpoints2.Add(new Rhino.Geometry.Point3d(Xfl[1], 0, 0));

            DA.SetDataList(0, newpoints);
            DA.SetDataList(1, newpoints2);
        }

        /// <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("1afb0744-2858-4268-a92c-4f01dd652745"); }
        }
    }
}
```