Vedlegg A: Kode av FDM 2D

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

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

// In order to load the result of this wizard, you will also need to
// add the output bin/ folder of this project to the list of loaded
// folder in Grasshopper.
// You can use the _GrasshopperDeveloperSettings Rhino command for that.

namespace FormFinding
{
    public class FDMComponent : GH_Component
    {
        /// <summary>
        /// Each implementation of GH_Component must provide a public 
        /// constructor without any arguments.
        /// Category represents the Tab in which the component will appear, 
        /// Subcategory the panel. If you use non-existing tab or panel names, 
        /// new tabs/panels will automatically be created.
        /// </summary>
        public FDMComponent()
          : base("FDM 2D", "FDM",
              "Hanging chain #2",
              "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.AddNumberParameter("Distance", "D", "Length of chain", 
GH_ParamAccess.item);
            pManager.AddNumberParameter("Segments", "S", "Number of segments", 
GH_ParamAccess.item);
            pManager.AddNumberParameter("Force in z-direction", "Pz", "Forces in each 
point", GH_ParamAccess.item);
            pManager.AddNumberParameter("Force Density", "q", "Force Density", 
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 can be used to retrieve data from input
parameters and
        /// to store data in output parameters.</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 F = double.NaN;                                          /// Input #2,
kraften i z-retning i punktene
            if (!DA.GetData(2, ref F)) { return; }

            double q = double.NaN;                                          /// Input #3,
krafttetthet
            if (!DA.GetData(3, ref q)) { 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);
            }

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

            Xnl.RemoveAt(0);
            Xnl.RemoveAt(Xnl.Count - 1);

            var M = Matrix<double>.Build;

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

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

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

            var xn = M.DenseOfColumnMajor(Xnl.Count, 1, Xnl.ToArray());
            var xf = M.DenseOfColumnMajor(Xfl.Count, 1, Xfl.ToArray());

            var yn = M.Dense(Xnl.Count, 1, 0);                                    /// y-
koordinatene i punktene starter alle i 0
            var yf = M.Dense(Xfl.Count, 1, 0);

            var Px = M.Dense(Xnl.Count, 1, 0);
            var Py = M.Dense(Xnl.Count, 1, F);

            var Q = M.DenseDiagonal(Xnl.Count + 1, Xnl.Count + 1, q);
/// Q blir satt opp med q som diagonalen

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

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

            var Xn = Dni * (Px - (Df * xf));
            var Yn = Dni * (Py - (Df * yf));

            var SSS = Yn.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(Xnl[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);                                   /// Utdata
defineres
            DA.SetDataList(1, newpoints2);
        }
        /// <summary>
        /// Provides an Icon for every component that will be visible in the User Interface.
        /// Icons need to be 24x24 pixels.
        /// </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>
        /// Each component must have a unique Guid to identify it.
        /// It is vital this Guid doesn't change otherwise old ghx files
        /// that use the old ID will partially fail during loading.
        /// </summary>
        public override Guid ComponentGuid
        {
            get { return new Guid("43c3a7e4-ccbc-48f7-b98e-35ea9dd4a4ec"); }
        }
    }
}
```