# Vedlegg B: Kode av TNA 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 TNAComponent : GH_Component
    {
        /// <summary>
        /// Initializes a new instance of the MyComponent1 class.
        /// </summary>
        public TNAComponent()
          : base("TNA 2D", "TNA",
              "Hanging chain #1",
              "Form Finding", "Thrust network analysis")
        {
        }

        /// <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("Scale Factor", "r", "Scale Factor",
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 F = double.NaN;                                          /// Input
#2, kraften i z-retning i punktene
            if (!DA.GetData(2, ref F)) { return; }

            double r = double.NaN;                                          /// Input
#2, skaleringsfaktor
            if (!DA.GetData(3, ref r)) { 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 matrsen
            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 matrsen
            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 e = new List<double>();                                         /// Listen
med verdiene som skal brukes i C matrisen
            e.AddRange(d);
            e.AddRange(g);
            var Xl = new List<double>();                                        /// Listen
med verdiene som skal brukes i x matrisen
            Xl.AddRange(Xnl);
            Xl.AddRange(Xfl);
                                                                                /// C- og
x-matrisene blir satt opp for å regne ut T-matrisen
            var C = M.DenseOfColumnMajor((Xnl.Count + 1), Xnl.Count + Xfl.Count,
e.ToArray());
            var x = M.DenseOfColumnMajor(Xnl.Count + Xfl.Count, 1, Xl.ToArray());
            var u = C * x;
            var u1 = u.ToColumnMajorArray();
            var u2 = u.RowCount;
            for (int i = 0; i < u2; i++)
            {
                u1[i] = Math.Abs(u1[i]);
            }
            var U = M.DenseOfDiagonalArray(Xnl.Count + 1, Xnl.Count + 1, u1);
            var lh = M.Dense(Xnl.Count + 1, 1, u1[0]);                              ///
Horizontal trust
            var t = (U.Inverse()) * lh;

            var T = M.DenseOfDiagonalArray(Xnl.Count + 1, Xnl.Count + 1,
t.ToColumnMajorArray());

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

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

            var Xn = Dni * ((Px*r) - (Df * xf));
            var Yn = Dni * ((Py*r) - (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);
            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("cf662aa0-624a-4752-a95c-f32c211d32a9"); }
        }
    }
}
```