Vedlegg E: Kode av TNA 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 TNAComponent3D : GH_Component
    {
        /// <summary>
        /// Initializes a new instance of the TNAComponent3D class.
        /// </summary>
        public TNAComponent3D()
          : base("TNAComponent3D", "TNA3D",
              "TNA in 3D",
              "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.AddPointParameter("Points", "P", "Points", GH_ParamAccess.list);
            pManager.AddNumberParameter("Scale Factor", "r", "Scale Factor",
GH_ParamAccess.item);
            pManager.AddNumberParameter("Force in z-direction", "Pz", "Force in z-
direction", GH_ParamAccess.item);
            pManager.AddIntegerParameter("# U divisions", "U", "Number of divisions in
U-direction", GH_ParamAccess.item);
            pManager.AddIntegerParameter("# V divisions", "V", "Number of divisions in
V-direction", 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.AddPointParameter("New points", "P", "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<Point3d> points = new List<Point3d>();                     ///
Inndata legges til
            if (!DA.GetDataList(0, points)) { return; }

            var M = Matrix<double>.Build;

            double r = double.NaN;
            if (!DA.GetData(1, ref r)) { return; }

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

            List<double> x = new List<double>();                            /// Liste
over verdiene til x-koordinatene
            for (int i = 0; i < points.Count; i++)
            {
                x.Add(points[i].X);
            }

            List<double> y = new List<double>();                            /// Liste
over verdiene til y-koordinatene
            for (int i = 0; i < points.Count; i++)
            {
                y.Add(points[i].Y);
            }

            List<double> z = new List<double>();                            /// Liste
over verdiene til z-koordinatene
            for (int i = 0; i < points.Count; i++)
            {
                z.Add(points[i].Z);
            }

            int V = new int();                                              ///
Inndata: V
            if (!DA.GetData(4, ref V)) { return; }
            int U = new int();                                              ///
Inndata: U
            if (!DA.GetData(3, ref U)) { return; }
            var u = U * (V + 1);                                            ///
Utregning av dimmensjoner for oppsett av matriser
            var v = V * (U + 1);
            var n = (V + 1) * (U + 1);                                      /// Antall
punkter ut fra U og V

            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 Xn1 = X.RemoveRow(n - 1);                                   /// Fjerne
opplagerpunktene for å sette opp Xn
            var Xn2 = Xn1.RemoveRow(n - V - 1);
            var Xn3 = Xn2.RemoveRow(V);
            var Xn = Xn3.RemoveRow(0);
```

```
            var Xf1 = X.Row(0);                                         /// Legge
til opplagerpunktene for å sette opp Xf
            var Xf2 = X.Row(V);
            var Xf3 = X.Row(n - V - 1);
            var Xf4 = X.Row(n - 1);
            var Xf = M.DenseOfRowVectors(Xf1, Xf2, Xf3, Xf4);

            var Yn1 = Y.RemoveRow(n - 1);                               /// Fjerne
opplagerpunktene for å sette opp Yn
            var Yn2 = Yn1.RemoveRow(n - V - 1);
            var Yn3 = Yn2.RemoveRow(V);
            var Yn = Yn3.RemoveRow(0);

            var Yf1 = Y.Row(0);                                         /// Legge
til opplagerpunktene for å sette opp Yf
            var Yf2 = Y.Row(V);
            var Yf3 = Y.Row(n - V - 1);
            var Yf4 = Y.Row(n - 1);
            var Yf = M.DenseOfRowVectors(Yf1, Yf2, Yf3, Yf4);

            var Zn1 = Z.RemoveRow(n - 1);                               /// Fjerne
opplagerpunktene for å sette opp Zn
            var Zn2 = Zn1.RemoveRow(n - V - 1);
            var Zn3 = Zn2.RemoveRow(V);
            var Zn = Zn3.RemoveRow(0);

            var Zf1 = Z.Row(0);                                         /// Legge
til opplagerpunktene for å sette opp Zf
            var Zf2 = Z.Row(V);
            var Zf3 = Z.Row(n - V - 1);
            var Zf4 = Z.Row(n - 1);
            var Zf = M.DenseOfRowVectors(Zf1, Zf2, Zf3, Zf4);


            List<double> g = new List<double>();                        /// Lister
til å finne ut når C-matrisen "foskyves"
            List<double> h = new List<double>();
            for (int i = 0; i <= n; i++)
            {
                g.Add((V + 1) * i);
                h.Add(((V + 1) * i) + 1);
            }

            List<double> I = new List<double>();                        /// Liste
over verdiene i C-matrisen
            for (int i = 1; i <= n; i++)
            {

                if (i > (V + 2))
                {
                    for (int j = 1; j <= (i - V - 2); j++)
                    {
                        I.Add(0);
                    }
                }
                if (i > (V + 1))
                {
                    I.Add(1);
                }
                if (i <= V)
                {
                    for (int j = 1; j <= (i - 1); j++)
```

```
    {
        I.Add(0);
    }
}
if (i > V)
{
    for (int j = 1; j <= V; j++)
    {
        if (i <= (u + 1))
        {
            I.Add(0);
        }
    }
}
if (i > u + 1)
{
    for (int j = 1; j <= (u + V + 1 - i); j++)
    {
        if (i < n)
        {
            I.Add(0);
        }
    }
}
if (i <= u)
{
    I.Add(-1);
}
if (i < u)
{
    for (int j = 1; j <= (u - i); j++)
    {
        I.Add(0);
    }
}
//
for (int j = 1; j <= (i - 2); j++)
{
    I.Add(0);
}
for (int j = 2; j <= (i - 1); j++)
{
    if (h.Contains(j))
    {
        var c = I.Count;
        I.RemoveAt(c - 1);
    }
}
if (h.Contains(i))
{

}
else
{
    I.Add(1);
}
if (g.Contains(i))
{
    if (i < n - 1)
    {
        I.Add(0);
    }
```

```
                }
                else
                {
                    I.Add(-1);
                }
                if (h.Contains(i))
                {
                    if (i < n - 1)
                    {
                        I.Add(0);
                    }
                }
                for (int j = 1; j <= v - i; j++)
                {
                    I.Add(0);
                }
                for (int j = 2; j <= (i - 1); j++)
                {
                    if (h.Contains(j))
                    {
                        if (i < (n - 1))
                        {
                            I.Add(0);
                        }
                    }
                }
                if (i >= (n - U))
                {
                    if (i < (n - 1))
                    {
                        for (int k = 0; k <= (i - n + U); k++)
                        {
                            var c = I.Count;
                            I.RemoveAt(c - 1);
                        }
                    }
                }
                if (i == 1)
                {
                    var c = I.Count;
                    I.RemoveAt(c - 1);
                }
            }

            var C = M.DenseOfColumnMajor((u + v), n, I.ToArray());          /// C blir
satt opp ved help av listen I

            var u0 = C * X;
            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 Lh = (U0.PointwisePower(2) + V0.PointwisePower(2) +
W0.PointwisePower(2)).PointwisePower(0.5);
            var lh = M.Dense(u1.Length, 1, Lh.At(0,0));
            var t = Lh * lh;
            var T = M.DenseOfDiagonalArray(u + v, u + v, t.ToColumnMajorArray());

            var Cn1 = C.RemoveColumn(n - 1);                                /// Deler
C inn i Cn og Cf
            var Cn2 = Cn1.RemoveColumn(n - V - 1);
            var Cn3 = Cn2.RemoveColumn(V);
            var Cn = Cn3.RemoveColumn(0);

            var Cf1 = C.Column(0);
            var Cf2 = C.Column(V);
            var Cf3 = C.Column(n - V - 1);
            var Cf4 = C.Column(n - 1);
            var Cf = M.DenseOfColumnVectors(Cf1, Cf2, Cf3, Cf4);

            List<double> px = new List<double>();                           /// Last i
x-retning settes lik 0
            for (int i = 1; i <= (n - 4); 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 <= (n - 4); 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 <= (n - 4); i++)
            {
                pz.Add(F);
            }
            var Pz = M.DenseOfColumnMajor(pz.Count, 1, pz.ToArray());

            //var Q = M.DenseDiagonal(u + v, u + v, r);

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

            var Dn = CnT * T * Cn;                                          ///
Utregning av Dn, Df og invers av Dn
            var Df = CnT * T * Cf;
```

```
            var Dni = (Dn.Inverse());

            var XN = Dni * ((Px*r) - (Df * Xf));                                ///
Utregning av de nye punktene
            var YN = Dni * ((Py*r) - (Df * Yf));
            var ZN = Dni * ((Pz*r) - (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> newpoints = new List<Point3d>();                      /// Utdata
#0, de nye punktene
            for (int j = 0; j < Nc; j++)
            {
                newpoints.Add(new Rhino.Geometry.Point3d(XNo[j], YNo[j], ZNo[j]));
            }

            List<Point3d> newpoints2 = new List<Point3d>();                     /// Utdata
#1, opplagerpunktene
            for (int j = 0; j < 4; j++)
            {
                newpoints2.Add(new Rhino.Geometry.Point3d(Xf.Column(0)[j],
Yf.Column(0)[j], Zf.Column(0)[j]));
            }

            List<Point3d> newpoints3 = new List<Point3d>();                     /// Utdata
#2, Alle punktene samlet
            newpoints3.Add(newpoints2[0]);
            for (int j = 0; j < V - 1; j++)
            {
                newpoints3.Add(new Rhino.Geometry.Point3d(XNo[j], YNo[j], ZNo[j]));
            }
            newpoints3.Add(newpoints2[1]);
            for (int j = (V - 1); j < (n - V - 3); j++)
            {
                newpoints3.Add(new Rhino.Geometry.Point3d(XNo[j], YNo[j], ZNo[j]));
            }
            newpoints3.Add(newpoints2[2]);
            for (int j = (n - V - 3); j < (n - 4); j++)
            {
                newpoints3.Add(new Rhino.Geometry.Point3d(XNo[j], YNo[j], ZNo[j]));
            }
            newpoints3.Add(newpoints2[3]);

            DA.SetDataList(0, newpoints);                                       /// Utdata
defineres
            DA.SetDataList(1, newpoints2);
            DA.SetDataList(2, newpoints3);
        }

        /// <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("e3753daf-3b2b-49b2-883e-7320b940a644"); }
        }
    }
}
```