AUTODESK **UNIVERSITY** 2015

ES9542

Dynam(o)ite Your Design for Engineers

Dieter Vermeulen, M.Sc. Str. Eng
Autodesk

Learning Objectives

- Learn how to generate complex structural models in Revit using Dynamo
- Learn how to rewire the analysis design in Robot Structural Analysis with Dynamo
- Learn how to introduce structural optimization techniques in the analysis process
- Learn how to apply intelligent behavior to structural analysis models using Dynamo

Description

In the world of structural engineering, we are challenged to make several structural analysis models, to find the best solution, and to be leaders in economic structure design. Robot Structural Analysis and Revit Structure are great solutions that help us with this. This class will show you a whole new way of analyzing your structures. Learn how you can catch the architectural design and add behavior and rules to the structural design in Revit and Robot Structural Analysis in the less time. You will discover how to apply computational design with Dynamo in Robot Structural Analysis Professional. You will also learn how to apply structural optimization techniques to your analysis models in Robot Structural Analysis with Dynamo. Finally, this will lead you to an introduction into the world of genetic algorithms.

Your AU Experts



*Working as a Structural Technical Specialist for the Northern European region at Autodesk, **Dieter Vermeulen** is specialized in the products of the structural solutions portfolio. Within that domain he supports the channel partners and customers with workflow solutions, especially - but not limited to - for structural engineering.*

With Revit, RSA and Dynamo as his sidekicks, he is offering BIM workflow solutions covering the building process from design over analysis to fabrication for steel and concrete constructions.

Dieter has over 15 years of experience in the structural engineering business and started his career in 2000 at Jacobs Engineering as Structural Engineer. Since 2008 he became active in the Autodesk channel and worked as a Sr. BIM Implementation Manager for a Value Added Reseller in Belgium.

Twitter	www.twitter.com/BIM4Struc
LinkedIn	www.linkedin.com/in/dietervermeulen
Wordpress personal blog	www.revitbeyondbim.wordpress.com
Typepad team blog	www.autodesk.typepad.com/bimtoolbox/
YouTube Channel	www.youtube.com/user/RevitbeyondBIM

Table of Contents

Learning Objectives 1
Description 1
Your AU Experts 1
Table of Contents 2
Computational Design 5
What is computational design? 5
Autodesk Dynamo 5
Difference between Autodesk Dynamo and Dynamo Studio 5
Structural Optimization 6
Genetic Optimization 7
Genetic Optimization workflow 8
Getting started 9
Project Description 10
Used software 13
Important notice – Setting in Robot Structural Analysis 15
Case 1 – Complex structural model in Revit 16
General overview of the script 17
Step 1 - Input parameters 18
Step 2 – Outline main truss chords 19
Step 3 – Division of the truss chords 21
Step 4 - Roof beams & bracings 23
Step 5 - Edge trusses 25
Step 6 - Inner trusses 27
Step 7 - Truss columns 29
Step 8 - Colorized display in Dynamo 35
Step 9 – Generation of structural model in Revit 37
Custom node “V bracings” 42
Custom node “Chained line” 42
Extended workflow – Link with Robot Structural Analysis 43
Case 2 – Complex analysis model in Robot Structural Analysis 44
General overview of the script 45
Step 1 – Input parameters 46

Step 2 to 7 – Creation of Dynamo geometry 47
Step 8 – Discretization of arcs for structural analysis model 48
Step 9 – Creation of analysis line model in RSA 49
Step 10 – Creation of load cases in RSA 51
Step 11 – Apply loads to the roof beams 52
Step 12 – Load sections in RSA 54
Step 13 – Assign sections to analytical members 55
Step 14 – Set boundary conditions : Supports and bar end releases 56
Step 15 – Analysis and results reading 60
Case 3 – Manual design exploration of a single complex truss 62
General overview of the script 63
Step 1 – Input parameters 64
Step 2 – Definition main truss chords 65
Step 3 – Generation of the full analysis model 67
Step 4 – Analysis of the model 69
Step 5 – Preparing the script for list input 71
Step 6 – Evaluation of the results 74
Case 4 – Parametric Run Optimization of a single complex truss 78
General overview of the script 79
Step 1 – Input of the base roof geometry 80
Step 2 - Constant input values and constraints 80
Step 3 - Design parameters 81
Step 4 - Load sections to RSA 81
Step 5 - Generation of design options 82
Step 6 - Analysis of the design options 83
Step 7 - Evaluation of the analysis results 86
Step 8 - Export of evaluation results to Excel 87
Simple Truss Weight Score Evaluation 89
Case 5 – Genetic Optimization of a single complex truss 90
Representation of Optimization Problem 91
General overview of the script 93
Step 1 - Input of the base roof geometry 94
Step 2 - Constant input values and constraints 94

Step 3 - Load sections to RSA 94
Step 4 - Generation of initial population list 95
Step 5 - Optimization using Genetic Algorithms 97
Step 6 - Interpretation of optimal result 101
Armadillo Spatial Truss Fitness Function 104
Armadillo Spatial Truss Optimo Evaluation 105
Learning resources 107

Computational Design

What is computational design?

The last few years we all experienced the transition from traditional Computer Aided Design (CAD) to Building Information Modelling (BIM). As our projects are getting more complex, and need more design solutions, the current BIM focused workflow is challenged again. The answer on that is **Computational Design**. This new design approach represents a profound shift in design thinking and methods. Representation is being replaced by simulation, and the crafting of objects is moving towards the generation of integrated systems through designer-authored computational processes.

While there is a particular history of such an approach in the building industry, its relative newness requires the continued progression of new modes of design thinking for the designers and engineers of the 21st century.

Literally, "computational design" means "giving shape with the means of the power of a computer and scripting".

Autodesk Dynamo

Autodesk gives an answer to this new challenge in our design world. This solution is called **Autodesk® Dynamo** (open source) and **Autodesk® Dynamo Studio** (Desktop Subscription). Dynamo lets designers and engineers create visual logic to explore parametric designs and automate tasks. It helps you to solve challenges faster by designing workflows that drive the geometry and behavior of design models. With Dynamo you will extend your designs into interoperable workflows for documentation, fabrication, coordination, simulation, and analysis.

Difference between Autodesk Dynamo and Dynamo Studio

	Dynamo	AUTODESK® DYNAMO STUDIO
Product	Open Source version available in products (i.e. Revit).	Standalone Polished version supported by Autodesk, working outside of Revit.
Key Value	The open source technologies which represent Dynamo's execution engine and graph UI built into Revit allowing better geometry. Allows you to add logic and behavior to Revit	All of the computational power and geometry tools of Dynamo Core. Aimed at the architect, engineer, or design professional who needs access to analysis linked with Dynamo.
Support	No Support	Basic Support
Documentation	Available online	Included
Availability	Click here	Click here
Pricing	Freely Available in Product. A valid Revit license is needed.	New Desktop Subscription Offering

Structural Optimization

In many of the cases design professionals ask themselves if their design (if it is an architectural or structural one) is optimal developed. It is not unusual to apply thumb rules as a solution to this question. Our agenda and our human brain doesn't allow us to comprehend all the scientific methods for an optimized design.

A *structure* in mechanics can be defined as "any assemblage of materials which is intended to sustain loads" (J.E. Gordon). *Optimization* means finding a solution which is the best. Thus, *structural optimization* is the subject of making an assemblage of materials sustaining loads in the best way.

There are several types of **structural optimization**:

- Sizing optimization: find the best solution for the structural thickness (i.e. cross-sectional areas of truss members).
- Shape optimization: find the best solution for the form or contour of some step of the boundary of the structural domain.

Topology optimization: this the most general form of structural optimization, where you try to find the best topology of a structure. (i.e. suppressing 0-tensioned members).

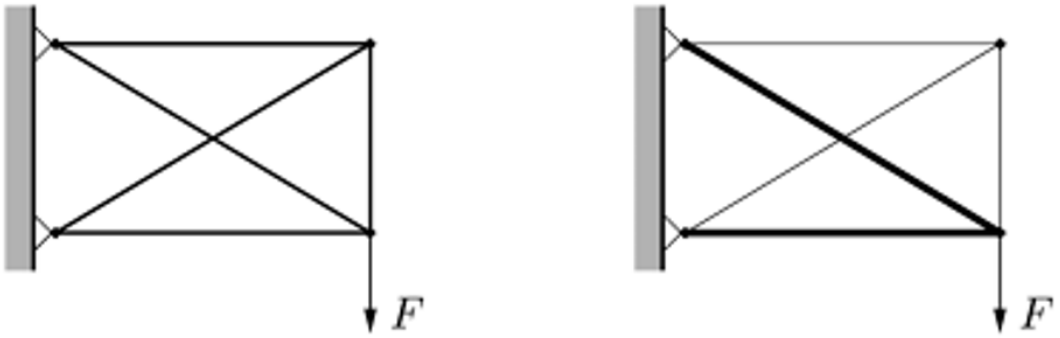


FIG. 1 - TOPOLOGY OPTIMIZATION (LEFT = INITIAL DESIGN, RIGHT = OPTIMIZED DESIGN)

In this class we are only considering a combination of *size* and *shape* optimization. Therefore we will make use of a technology called **Genetic Optimization.**

Genetic Optimization

Genetic Optimization is an optimization technique that makes use of *Genetic Algorithms* (GA). GAs usually operate on solutions that are encoded as genotypic representations (called chromosomes) from their original phenotypic representations (i.e. actual data values). GAs start with a set of initial solutions (population) that are randomly generated in the search space. For each solution in the current population, objective functions defining the optimization problem are evaluated and a fitness value is assigned to reflect its (relative) merit standing in the population. Based on the fitness values, a GA performs a selection operation that reproduces a set of solutions with higher fitness values from the previous generation to fill a mating pool. A crossover operation is then pursued with which two parent solutions in the mating pool are randomly selected and interchange, with a prescribed crossover probability, their respective string components at randomly selected bit locations referred to as cross sites. The resulting new solutions are called children or offspring. This step is meant to hopefully combine better attributes from the parent solutions so that child solutions with improved merits could be created. The next operation in GA is mutation that changes the genotype value at one or more randomly selected bit locations in a child solution with a prescribed mutation probability. This operation serves to possibly recover useful information that may never be accessible through selection and crossover operations and therefore encourages search into a completely new solution space. After these three basic operations, a new generation is created. The search process continues until prescribed stopping criteria are met, for example, when the maximum generation number is reached or improvement of the generation-wise optimized solutions is negligible.

Unlike single-objective problems where the objective function itself may be used as the fitness measure (possibly further with scaling and constraint-handling treatment), a multi-objective optimization algorithm needs a single fitness measure that reflects the overall merit of multiple objectives.

Genetic Optimization workflow

A typical workflow in a structural optimization process is represented in the flowchart below:

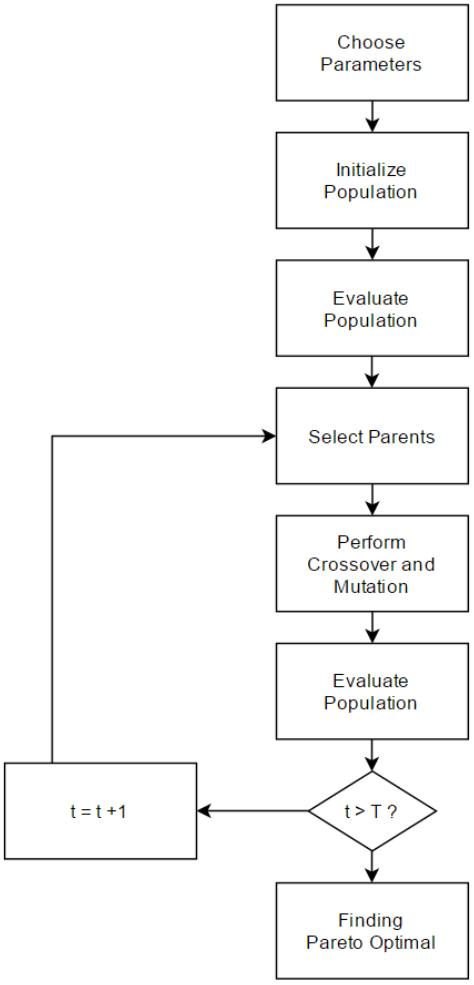
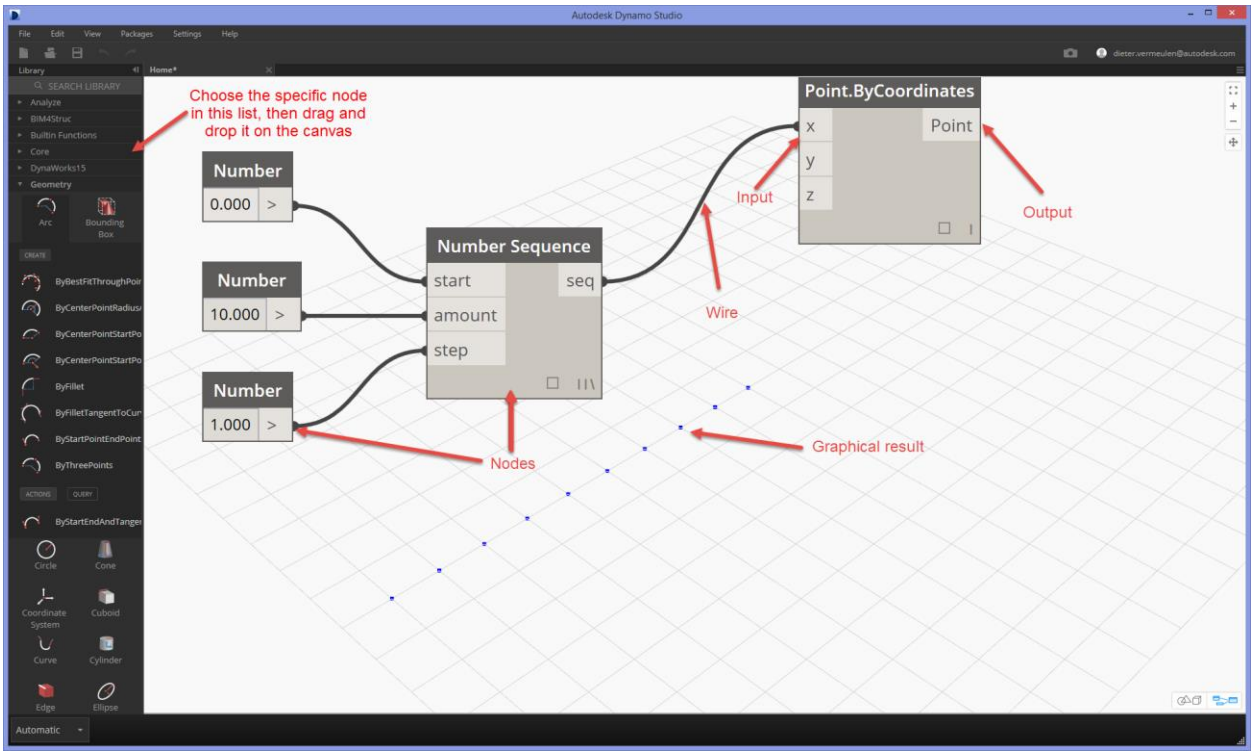


FIG. 2 - GENETIC OPTIMIZATION WORKFLOW

Getting started

For better understanding of this handout, a basic knowledge of Dynamo, Revit and Robot Structural Analysis is advised.

A comprehensive list of learning resources is listed at the end of this handout.

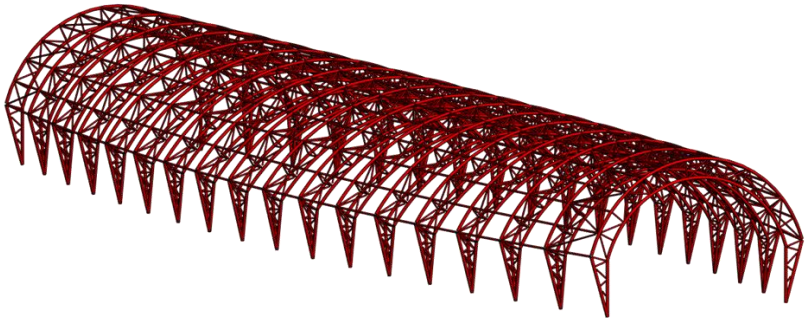


Project Description

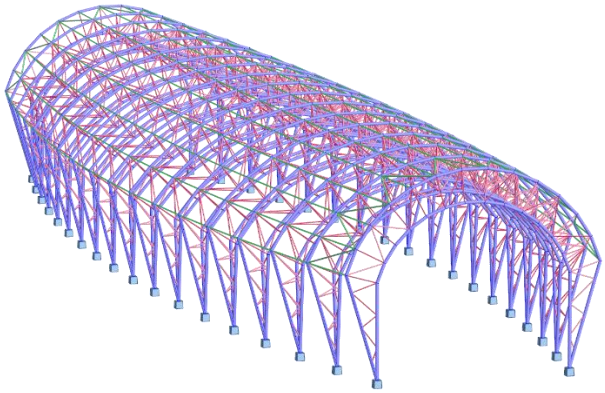
In the next chapters we are going to have a deeper look in how Dynamo, Revit and Robot Structural Analysis will help us to make the transition of “manual exploring of design options” through “parametric runs” to “genetic optimization” of a complex spatial truss girder.

This handout will handle 5 different Dynamo cases. In all of the cases the Truss Height, Truss Width, Number of Trusses along the length, Arc Discretization and the section types used, are set as variables.

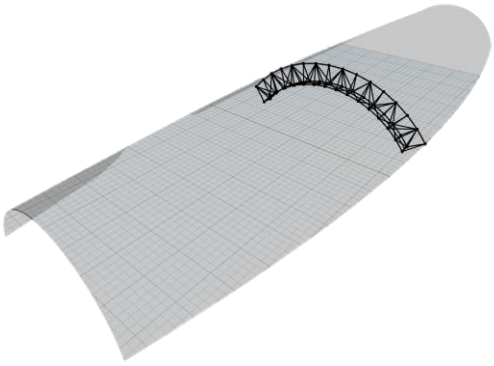
1. In case 1 the structural model in Revit will be set up using the Mass object that represents the roof outlines as base.



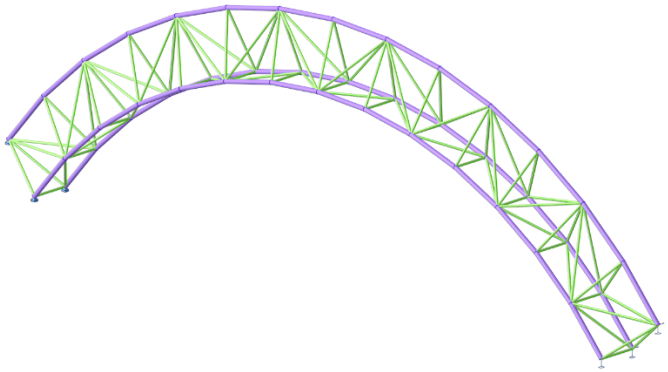
2. In case 2 the same Dynamo script will be reused to setup the analysis model in Robot Structural Analysis, in order to explore multiple design options in the analysis software.



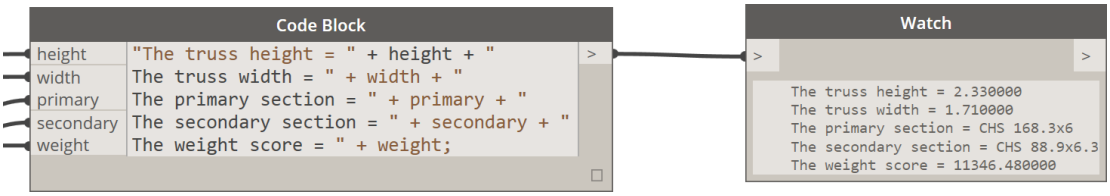
3. In case 3 the Dynamo script is being simplified to get the analysis model of the largest truss.



4. With case 4, the script and analysis model are being prepared for some optimization routine. In this case the optimization is done manually by performing a parametric run on the geometry variables.



5. Finally, case 5 will show how to introduce genetic algorithms in the optimization process, and how we let Dynamo define the optimal solution for the single truss, based on some given constraints.



The base design used for these cases, is coming from a fictive project made by our Autodesk EMEA NE Technical Sales Team, called “Millennium Project”. The structure shown below is part of the “Millennium Railway Station” and represents an ‘Armadillo’ shaped roof structure above the rail platform, with overall dimensions of 100 m length and 40 m width.

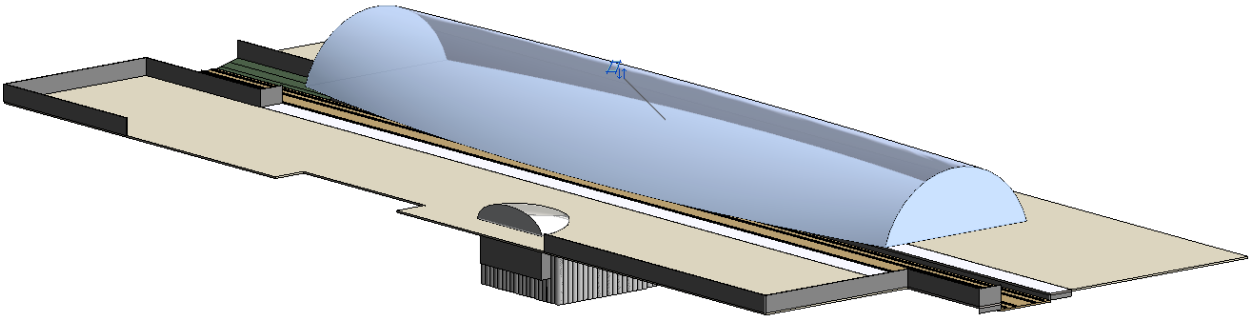


FIG. 3 A - MILLENNIUM RAILWAY STATION - ARMADILLO ROOF – ARCHITECTURAL DESIGN

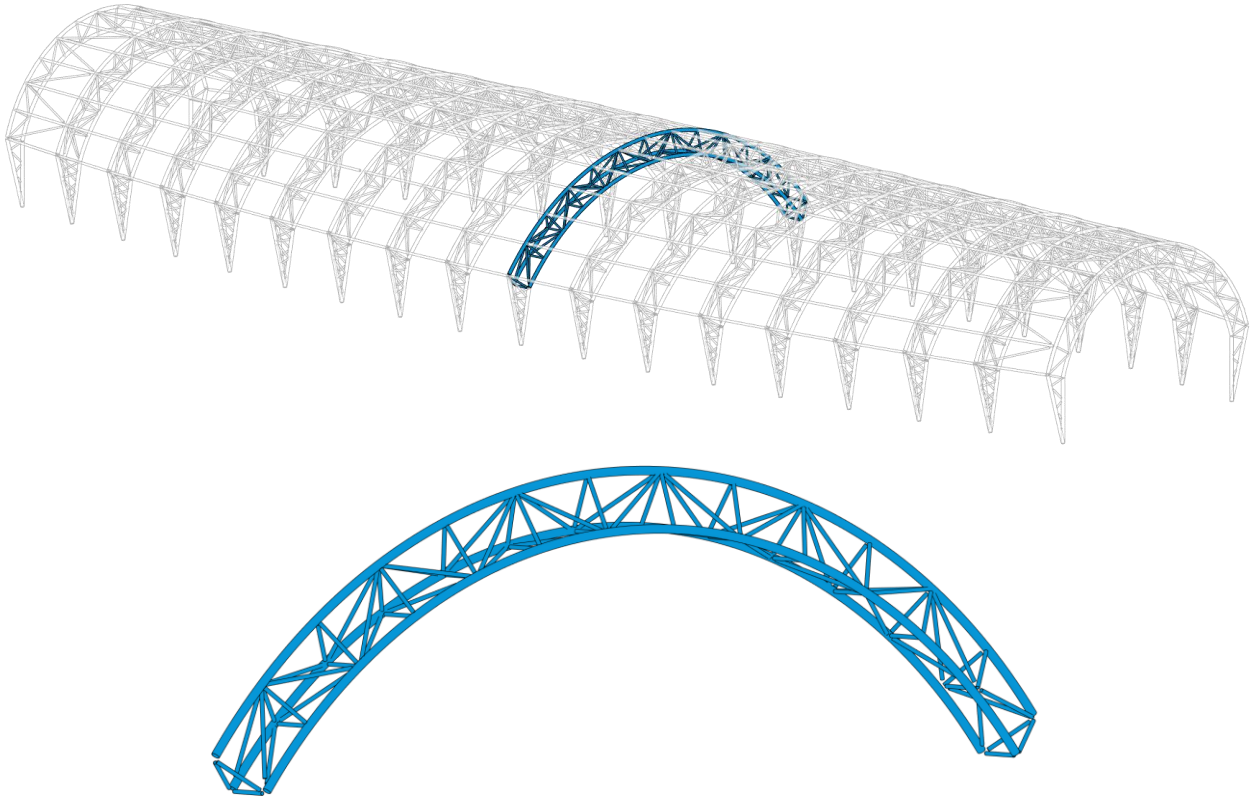


FIG. 4B - MILLENNIUM RAILWAY STATION - ARMADILLO ROOF - SPATIAL TRUSS

Used software

To perform the whole process, several software is used:

- **Autodesk Revit 2016**
  Used for the design of the full steel structure
  More info on : http://www.autodesk.com/products/revit-family/overview

- **Autodesk Robot™ Structural Analysis Professional 2016**
  Used for the structural analysis of the trusses
  More info on: http://www.autodesk.com/products/robot-structural-analysis/overview

- **React Structures Tech Preview 2**
  Alternatively you could also use Autodesk React Structures Tech Preview 2 instead of RSA.
  In that case you need to open Dynamo 0.9 from within the software.
  More info on: http://react.autodesk.com/

- **Autodesk Dynamo 0.9.0.3072**
  Free open-source tool for the computational design and optimization of the structure
  More info on: http://www.dynamobim.org

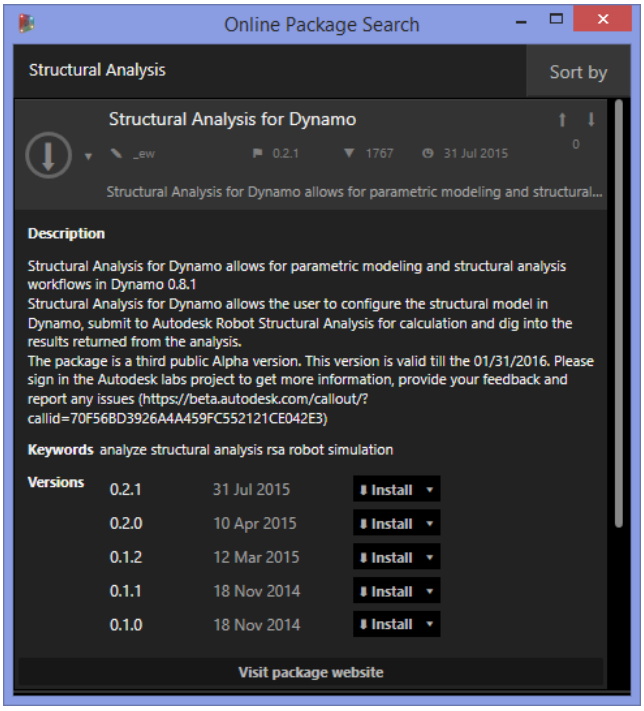
- **Autodesk Dynamo Studio 2016**
  When you don't have a valid Revit license installed, you can also make use of Dynamo Studio 2016.
  More info on: http://www.autodesk.com/products/dynamo-studio/overview

- **Optimo for Dynamo v. 0.1.2**
  Free package (plugin) for Dynamo with Genetic Algorithm nodes for the optimization of design and analysis.
  More info on: https://github.com/BPOpt/Optimo/wiki/0_-Home

- **Structural Analysis for Dynamo v. 0.2.2**
  Free package which makes the connection between Dynamo and Robot Structural Analysis.
  More info on:
  https://beta.autodesk.com/callout/?callid=70F56BD3926A4A459FC552121CE042E3

The Dynamo packages can be found in Dynamo, by performing the next steps:

1. In Dynamo, go to Packages > Search for a Package.
2. In the next dialog find the appropriate package and install using the ‘arrow’ button.

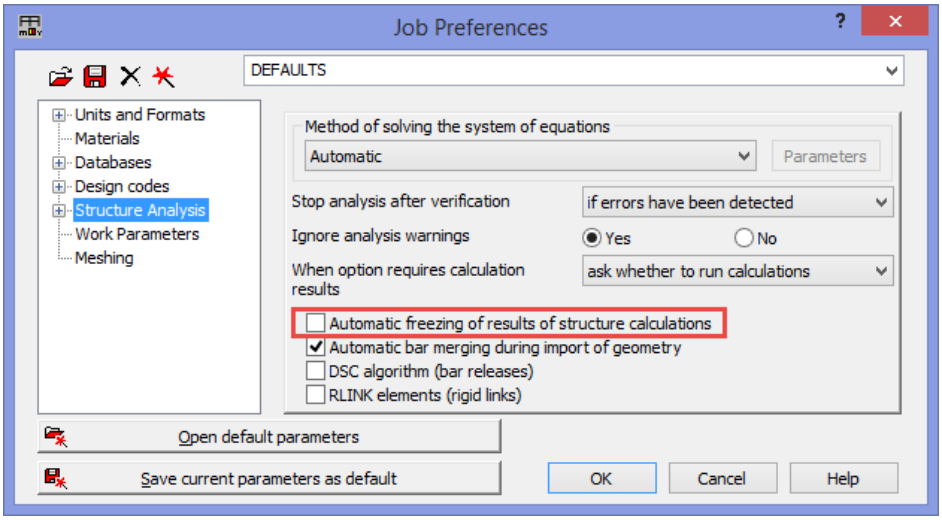


Important notice – Setting in Robot Structural Analysis



Before you run multiple analyses in Robot Structural Analysis, driven by Dynamo, it is advised to change the “automatic freezing of results” setting in the Job Preferences.

The dialog below can be accessed via the menu Tools > Job Preferences.



Case 1 – Complex structural model in Revit

In this first case we will build the complex structural design model of the *Armadillo Roof* in Revit using a parametrized approach with Dynamo, based on the initial design of the architect.

The base geometry of the roof is a Mass object created in Revit by the architectural designer.

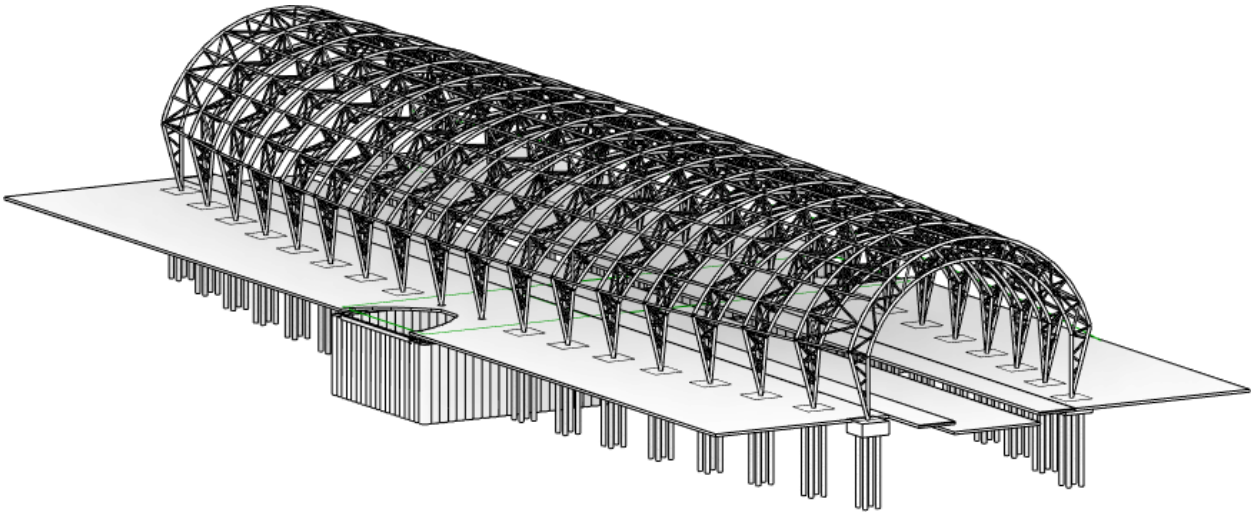


FIG. 5 - MILLENNIUM RAILWAY STATION - ARMADILLO ROOF - STRUCTURAL MODEL

DATASETS

Revit



Start model:
02 Railway Station - Structure - Platform – Start.rvt
Finalized model:
02 Railway Station - Structure - Platform - Dynamo design.rvt

Dynamo

Workspace:
Armadillo Platform - Structural Model.dyn
Custom nodes:
V bracings.dyf
Chained line.dyf

To open the dataset files:

1. Launch Revit 2016
2. Open the Revit file (.rvt)
3. In Revit, go to Add-Ins > Dynamo 0.9 and launch it.
4. Open the Dynamo script (.dyn)
5. Connect the wires as shown in the class and play with the parameters.

General overview of the script

The Dynamo script to generate the structural model in Revit, consists of 9 main parts, as illustrated below:

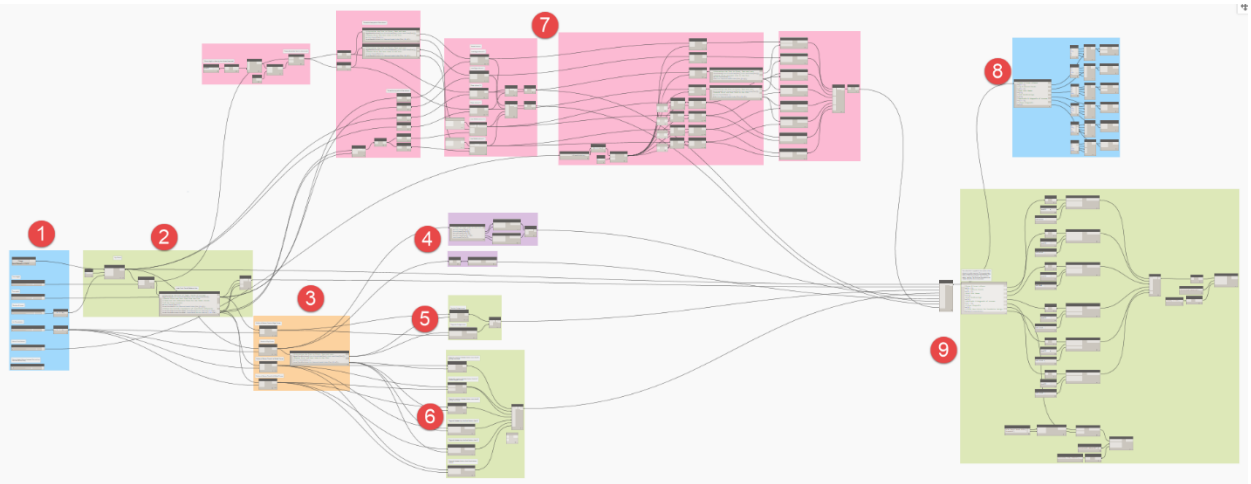


FIG. 6 - DYNAMO SCRIPT STRUCTURAL MODEL - GENERAL OVERVIEW

Step 1 = input parameters, such as the basis roof surface, truss geometry …

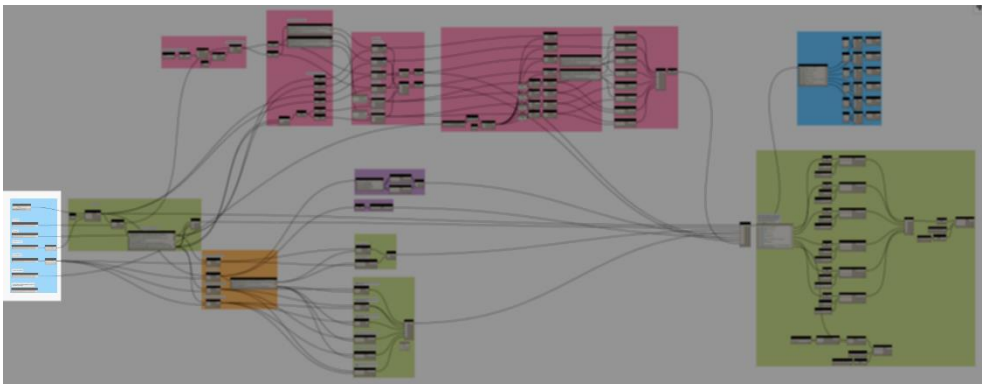
Steps 2-7 = generation of the model outline in Dynamo

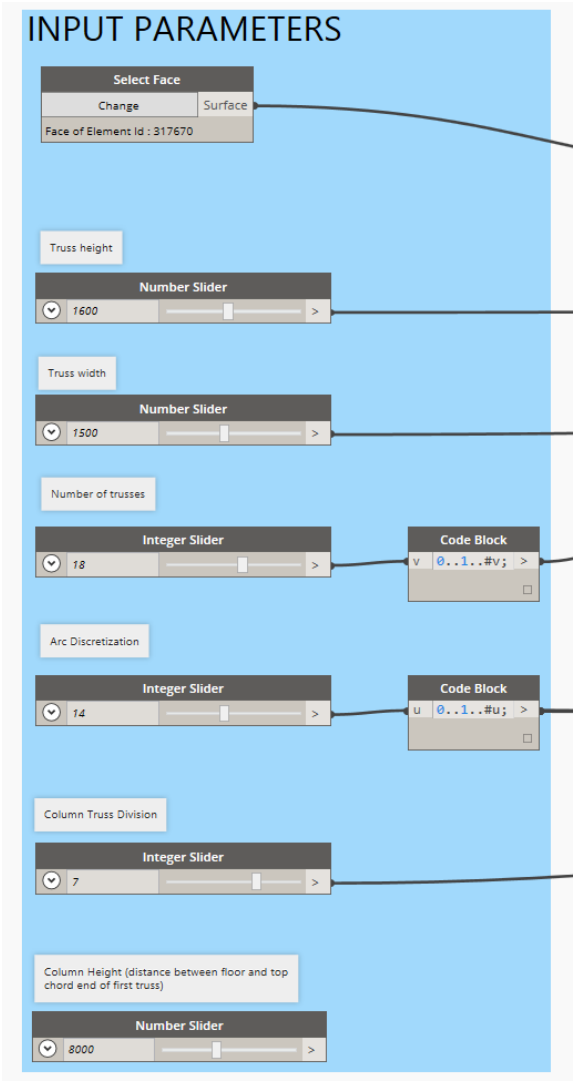
Step 8 = generation of the structural model in Revit, based on the results from 2-7

Step 9 = alternative colorized visualization of the wire model in Dynamo

Each of the steps is described in the chapters below.

Step 1 - Input parameters





In this step, you need to select the Face of the architectural mass object in Revit by clicking the *Select* or *Change* button in the *Select Face* node. From there you can enter the Revit GUI and select the top surface of the mass. This face will be the overall guide for the rest of the computed geometry.

The *Number slider* and *Integer slider* nodes allow to set the Truss height, Truss width, the Number of Trusses along the roof (including edge trusses), the Arc Discretization (which defines the number of panels in one spatial truss), the Column Truss division (which defines the number of panels in one lattice column) and finally the Column height, measured from the base level of the column to the top of the lattice column.

All values are expressed in mm, which is the Project Unit setting in Revit.

The *Integer slider* nodes are then connected with a Code Block to define a range between 0 and 1, with the indicated division rule.

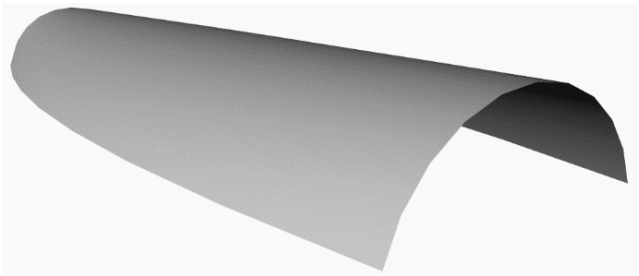
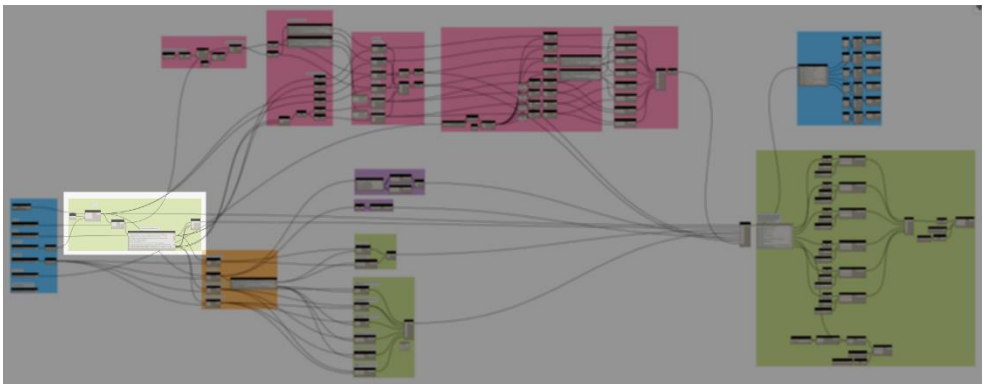
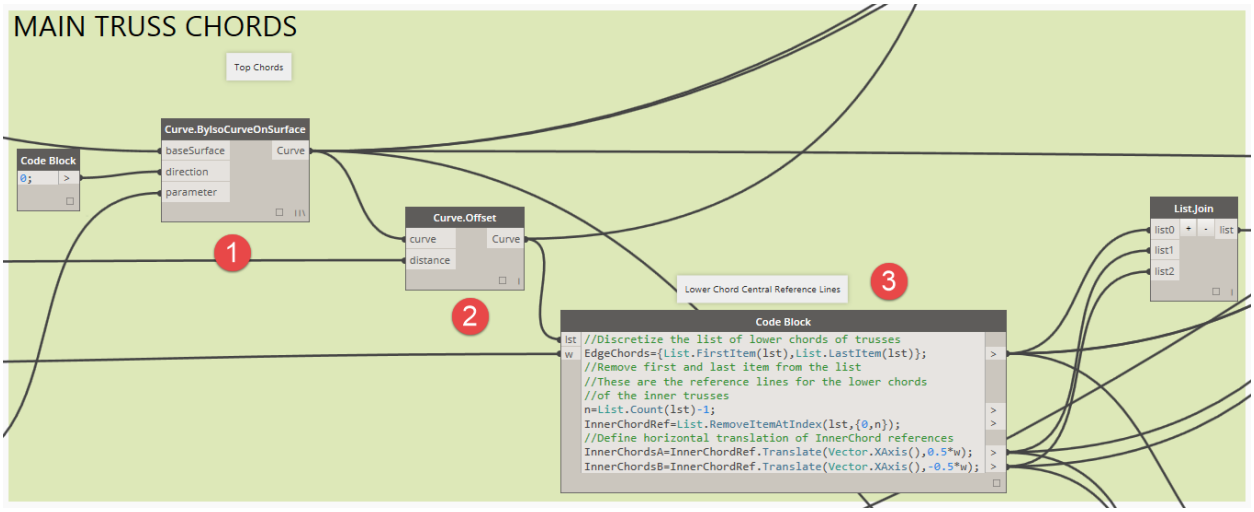


FIG. 7 - FACE SELECTION OF REVIT REFLECTED IN DYNAMO

Step 2 – Outline main truss chords





In this step the outlines for the top chords are defined.

(1) Therefore the isolines of the selected face from Revit are detected with the *Curve.ByIsoCurveOnSurface* node. *Direction* "0" defines the transversal direction of the double curved face. The *parameter* input is connected with the code block range defined for the *Number of Trusses* variable in the input parameters.

(2) The bottom chord reference line can be easily defined by offsetting the resulting curves, with a distance equal to the truss height, using the *Curve.Offset* node.

(3) This generates a list of bottom chord reference lines. For the edge trusses nothing needs to be done, but for the inner trusses – as it concerns spatial trusses – each line needs to be translated horizontally (Vector.XAxis) with half of the truss width.

The result is shown in Fig. 7.

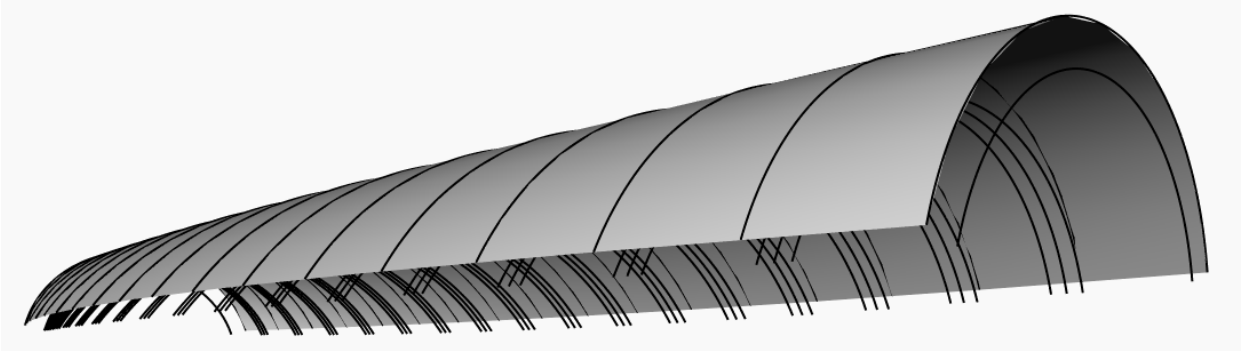
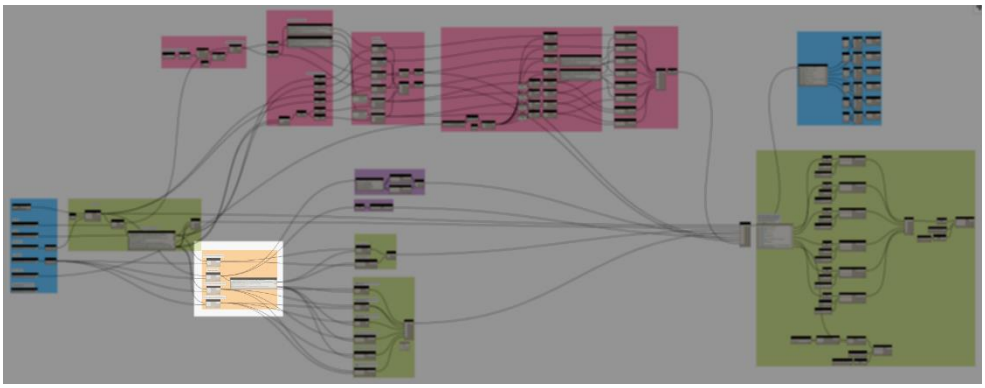
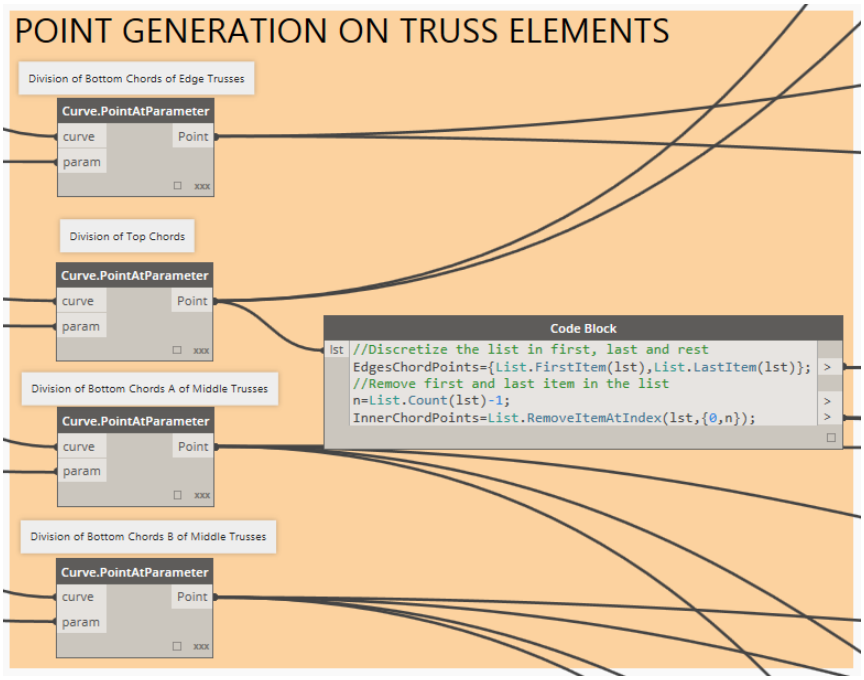


FIG. 8 - MAIN CHORD OUTLINES IN DYNAMO, DEPENDING ON THE SURFACE GEOMETRY

Step 3 – Division of the truss chords





The next step of the script is intended for the division of the main chords. These divisions are realized by generating an indicated number of points on equal distances on the curves. This is done with the *Curve.PointAtParameter* nodes, where the input *param* represents a sequence of numbers between 0 and 1 ("0" representing the start point and "1" representing the end point of the curve). This sequence consists of 'n' number of points, so that the division of the curve equals the indicated number for Arc Discretization as explained in Step 1.

The 4 lists of points that are generated in this step are further on used as placeholders for the diagonals in vertical lines in the spatial truss.

The code block allows to discretize the top chords in lines for the edge trusses (which are 2D trusses in the XZ plane) and the inner trusses (which are 3D spatial trusses).

The result is shown in Fig. 8

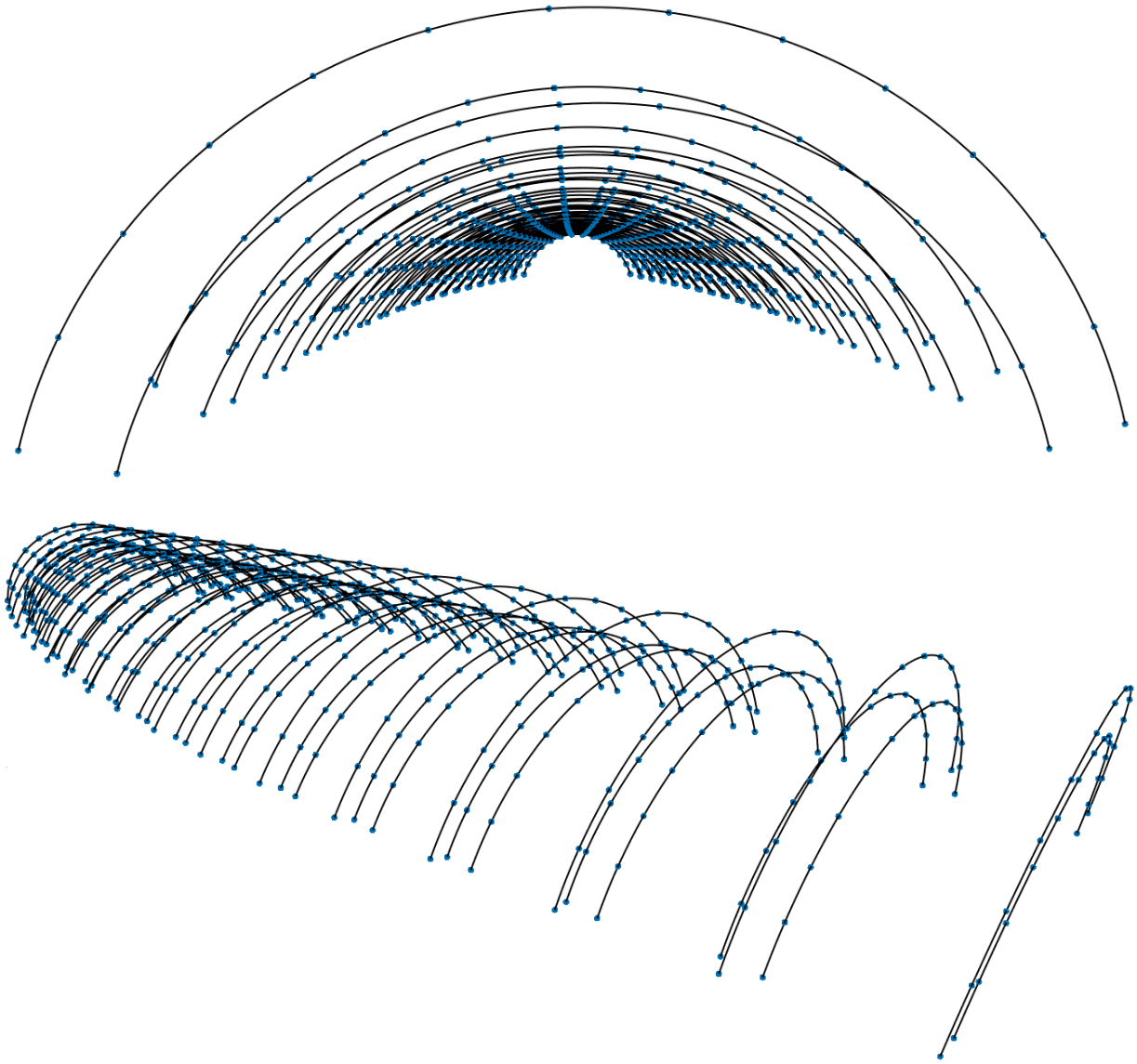
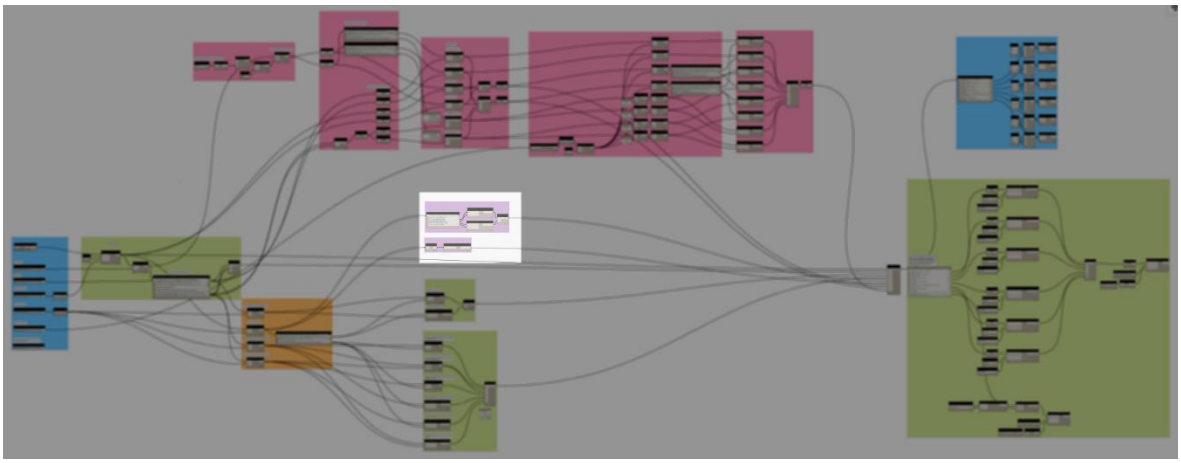
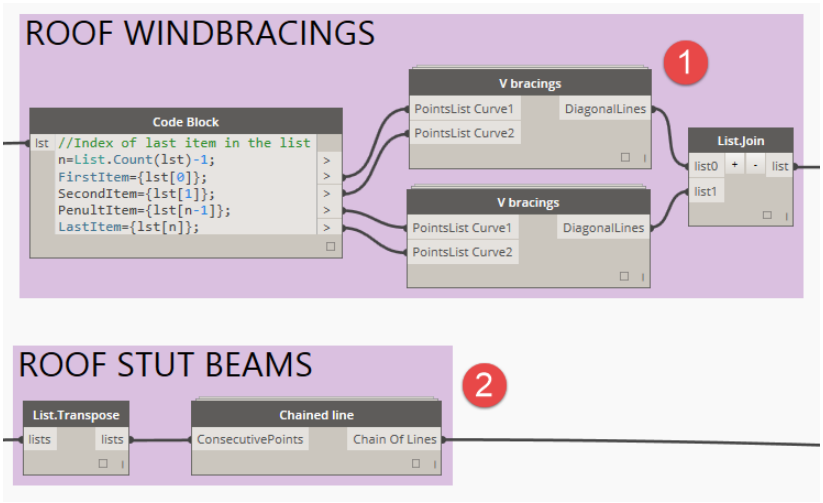


FIG. 9 - GENERATION OF POINTS ON THE MAIN CHORDS

Step 4 - Roof beams & bracings



Once the points are generated on the curves representing the main chords of the trusses, they can be used for generation of the roof lateral stability members.



(1) The *Curve.PointAtParameter* node for the division of the Top Chords (explained in Step 3), results in a number (equal to the number of trusses) of lists of points. As we only want to have wind bracings in the first and last field of the roof, we just need to select the sub lists from this node, with indices 0, 1, n-1 and n (with n = the last index). These resulting points are connected with the Custom node "V bracings". This custom node is taking 2 lists of points and generates diagonal lines between the curves, the points are generated on.

(2) Besides wind bracings, we also need strut beams in the roof. Therefore the top chord point lists are transposed with the *List.Transpose* node and brought back in the Custom node "Chained line", which generates an "exploded polycurve" from the points, in order to have separate lines for the output to Revit in later phase.

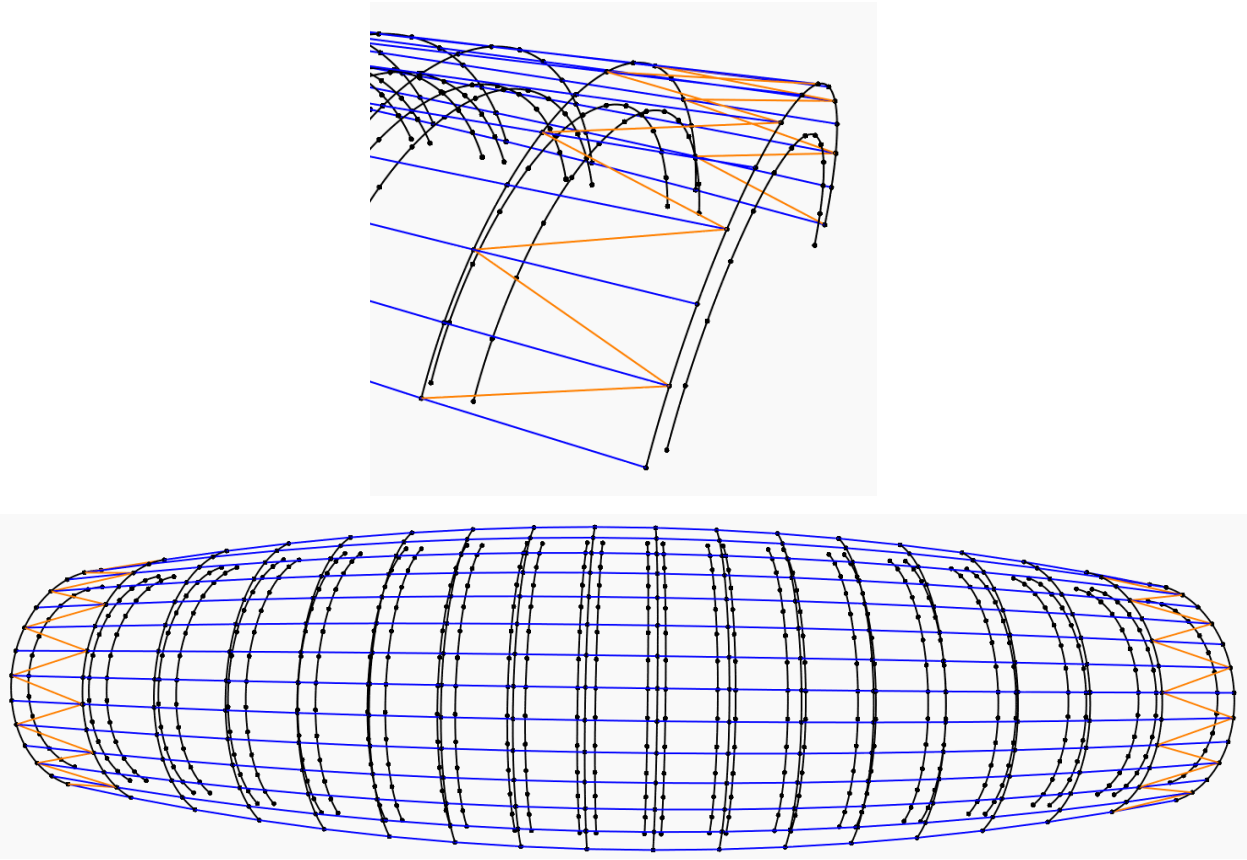
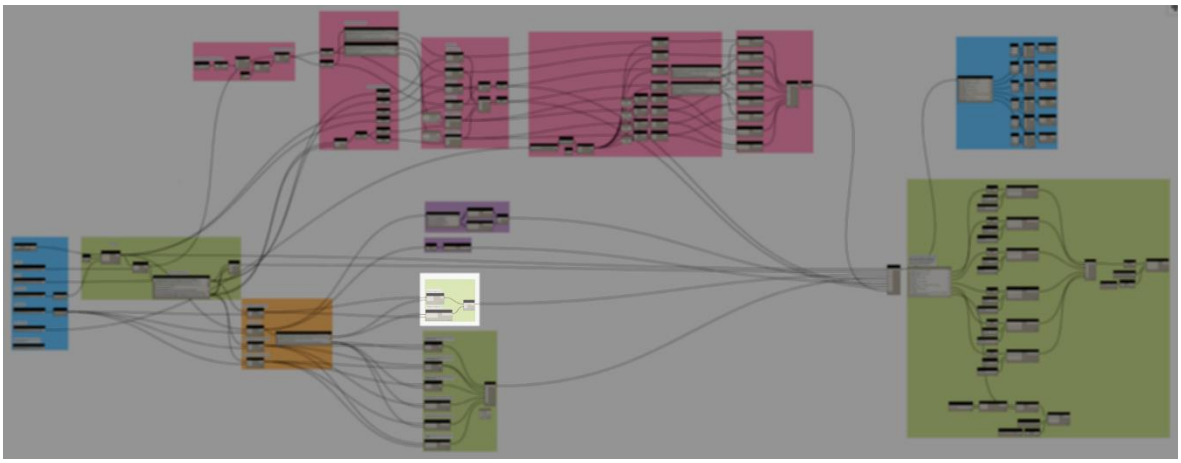
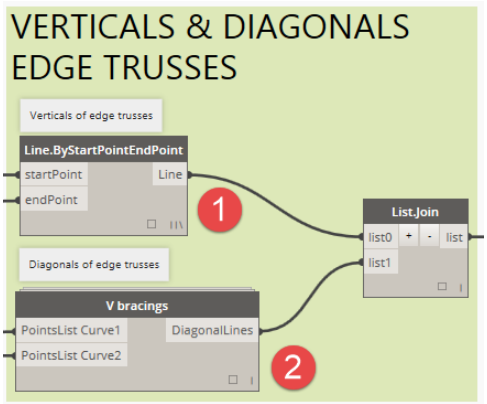


FIG. 10 - WIND BRACINGS (ORANGE) AND STRUT BEAMS (BLUE) IN THE ROOF

Step 5 - Edge trusses





In the 5th step of the script, the diagonals and verticals (posts) of the trusses at the outer edges of the roof are generated. These are 2D trusses in the XZ plane of the model.

(1) The verticals are generated by using the *Line.ByStartPointEndPoint* which uses the points generated at the top and bottom chords from step 3, for the point lists with index 0 and n (where n = the last index of the point sub lists). The input for this node is coming from the *EdgeChordPoints* variable in the code block (step 3).

(2) The diagonals use the same source points but they are connected with the *V bracings* custom node, which is explained a bit further below.

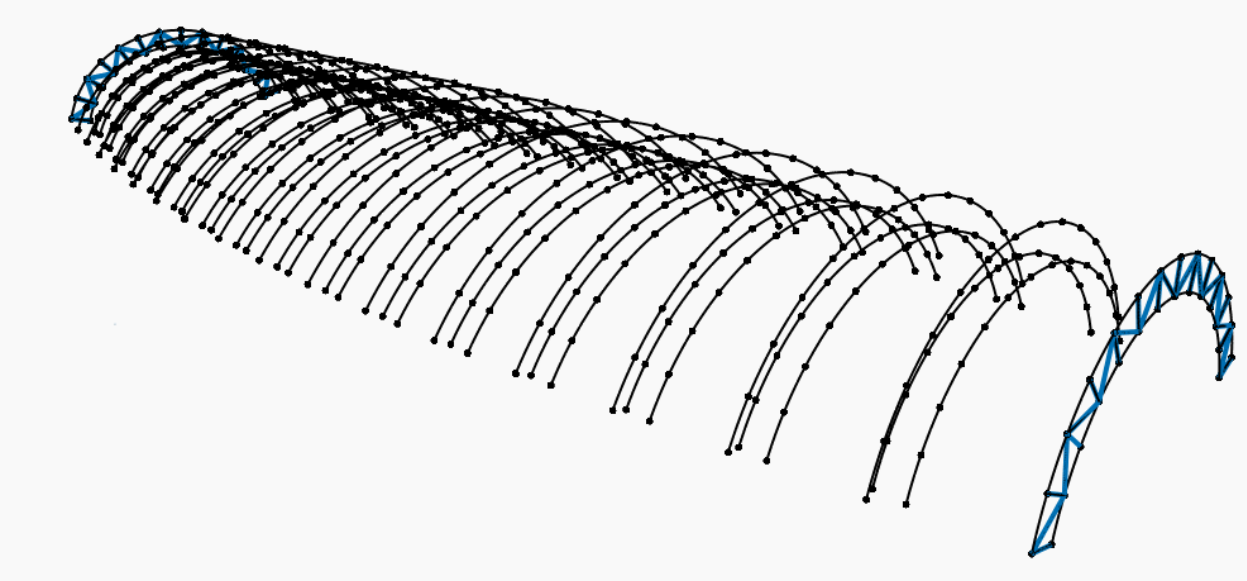
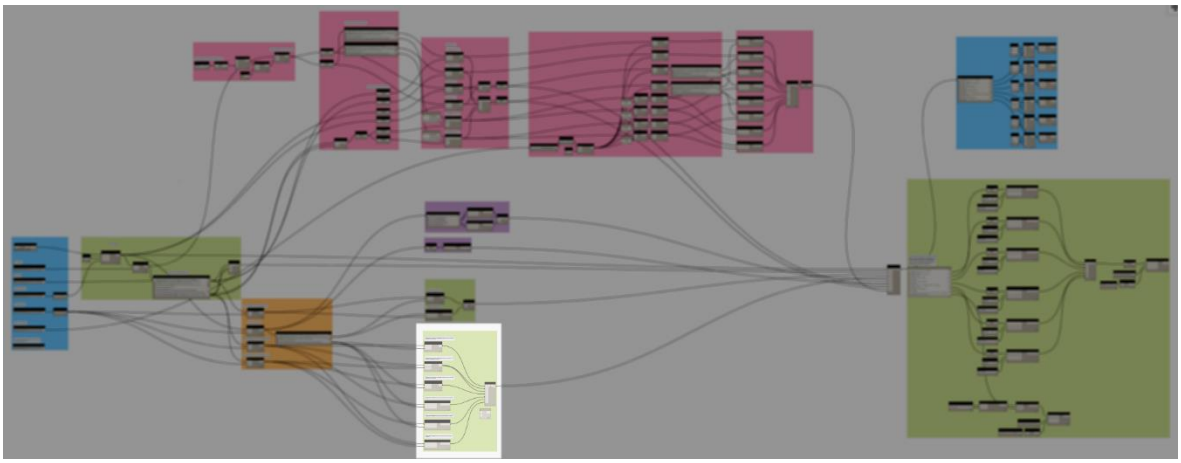
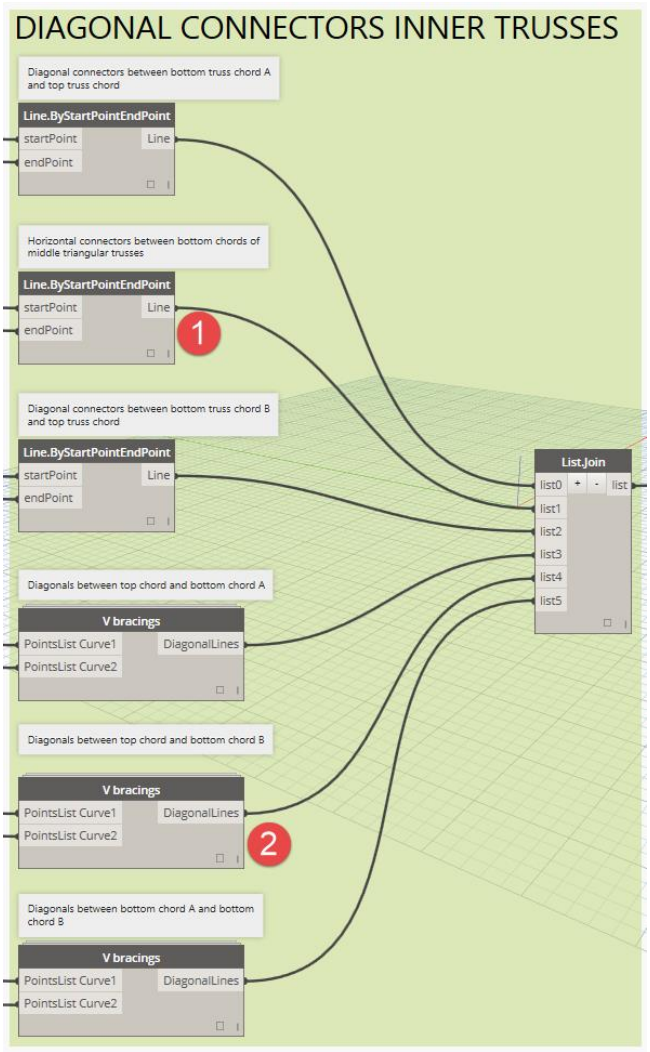


FIG. 11 - DIAGONALS AND VERTICALS OF THE EDGE TRUSSES

Step 6 - Inner trusses





The inner trusses, which are spatial 3D, are generated in an analogue way as the edge trusses, by connecting the nodes on the top and bottom chords with lines using the *Line.ByStartPointEndPoint* and *V bracing* custom nodes.

The input for these nodes is coming from the *InnerChordPoints* variable in the code block (step 3).

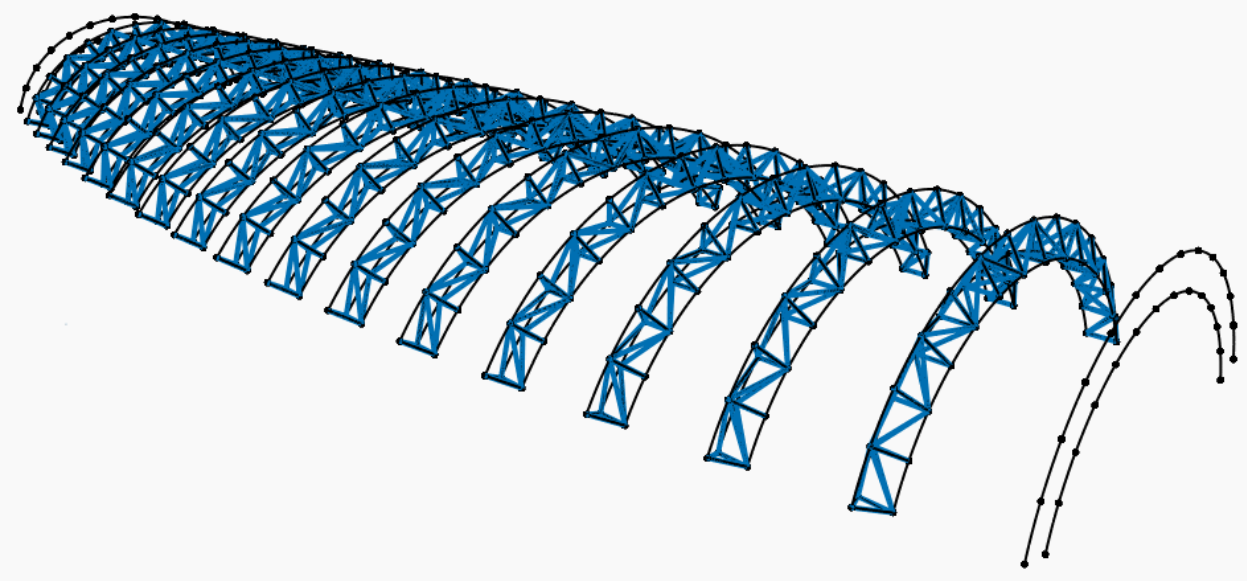
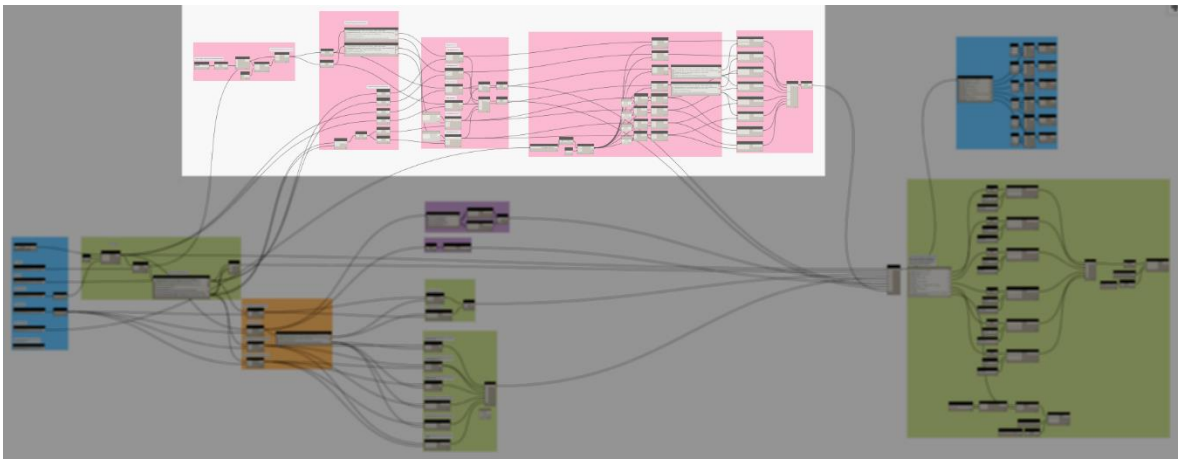
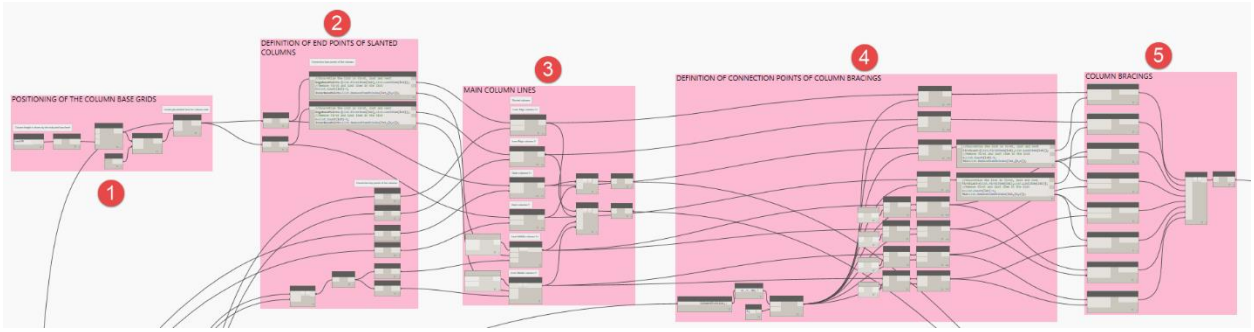


FIG. 12 - DIAGONALS OF THE SPATIAL 3D TRUSSES

Step 7 - Truss columns



The columns of the structure are also lattice structures and are treated in the same way as the truss girders. The generation of the columns exists of 5 parts to get to the final result:



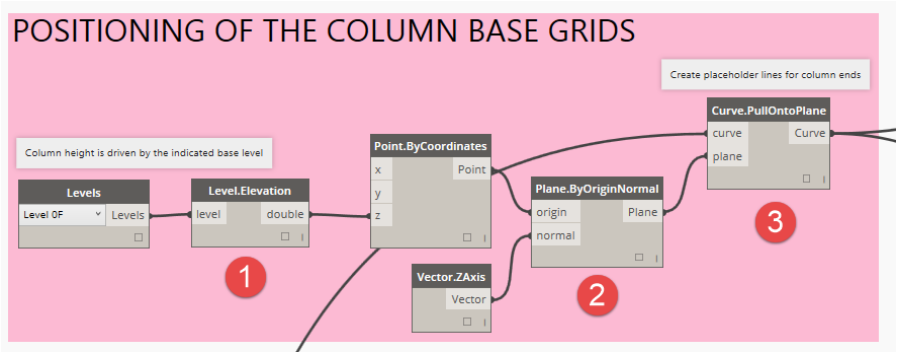
Step 7-1 Positioning of the column base grids

The base points of the columns are defined in two ways:

(1) The base level of the column is linked to the elevation of a chosen Revit level in the *Levels* node.

(2) The elevation is used to define a plane through the point with this Z coordinate.

(3) On this plane the top chord lines are projected using the *Curve.PullOntoPlane* node.



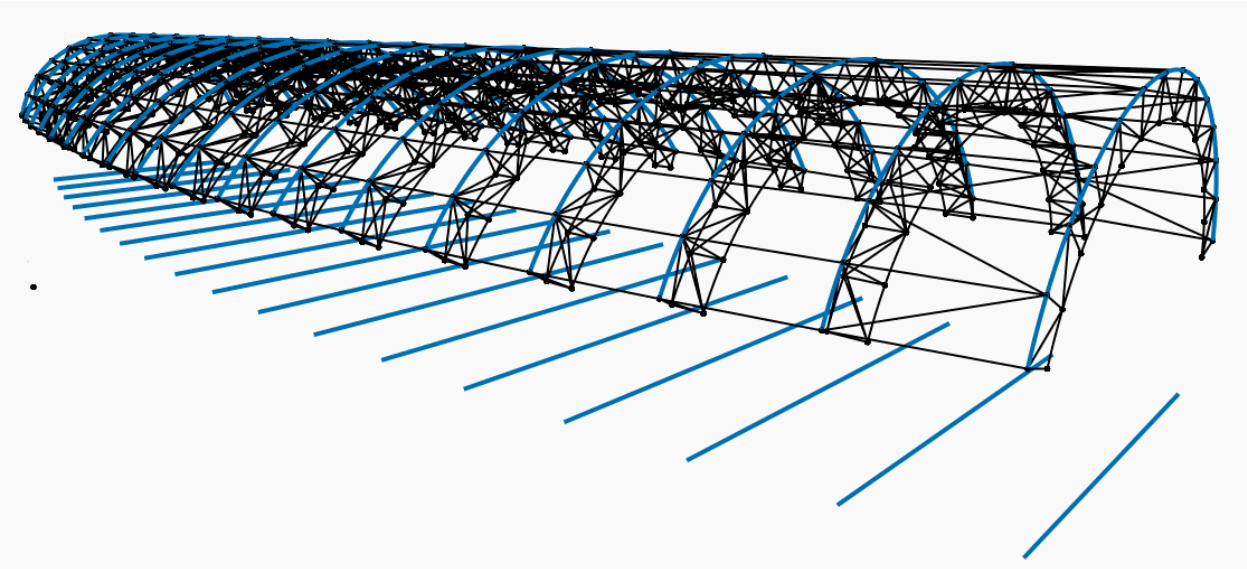
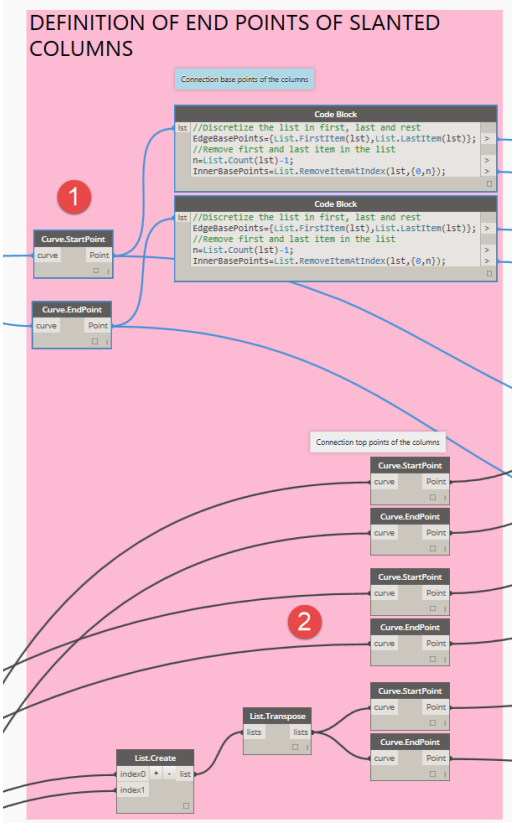


FIG. 13 - PROJECTED BASE GRIDS OF THE LATTICE COLUMNS

Step 7-2 Starting points of the columns

Once the base level of the columns is known, the result of the project lines can be used to detect the center points of the columns.



(1) By taking the *Curve.StartPoint* and *Curve.Endpoint* of the projected lines, we know the insertion point of the lattice column.

(2) The top of the columns are detected by taking the start- and endpoints of all the chords.

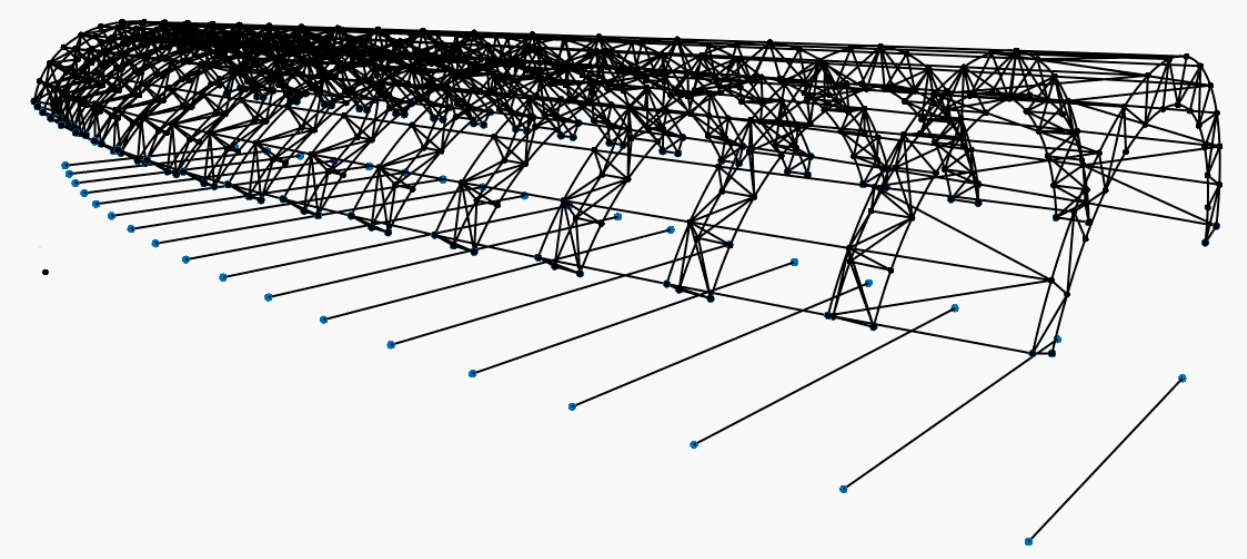


FIG. 14 - START AND ENDPOINTS OF THE LATTICE COLUMNS

Step 7-3 Main column lines

The generated start- and endpoints are now connected with each other to form the main lines of the lattice columns.

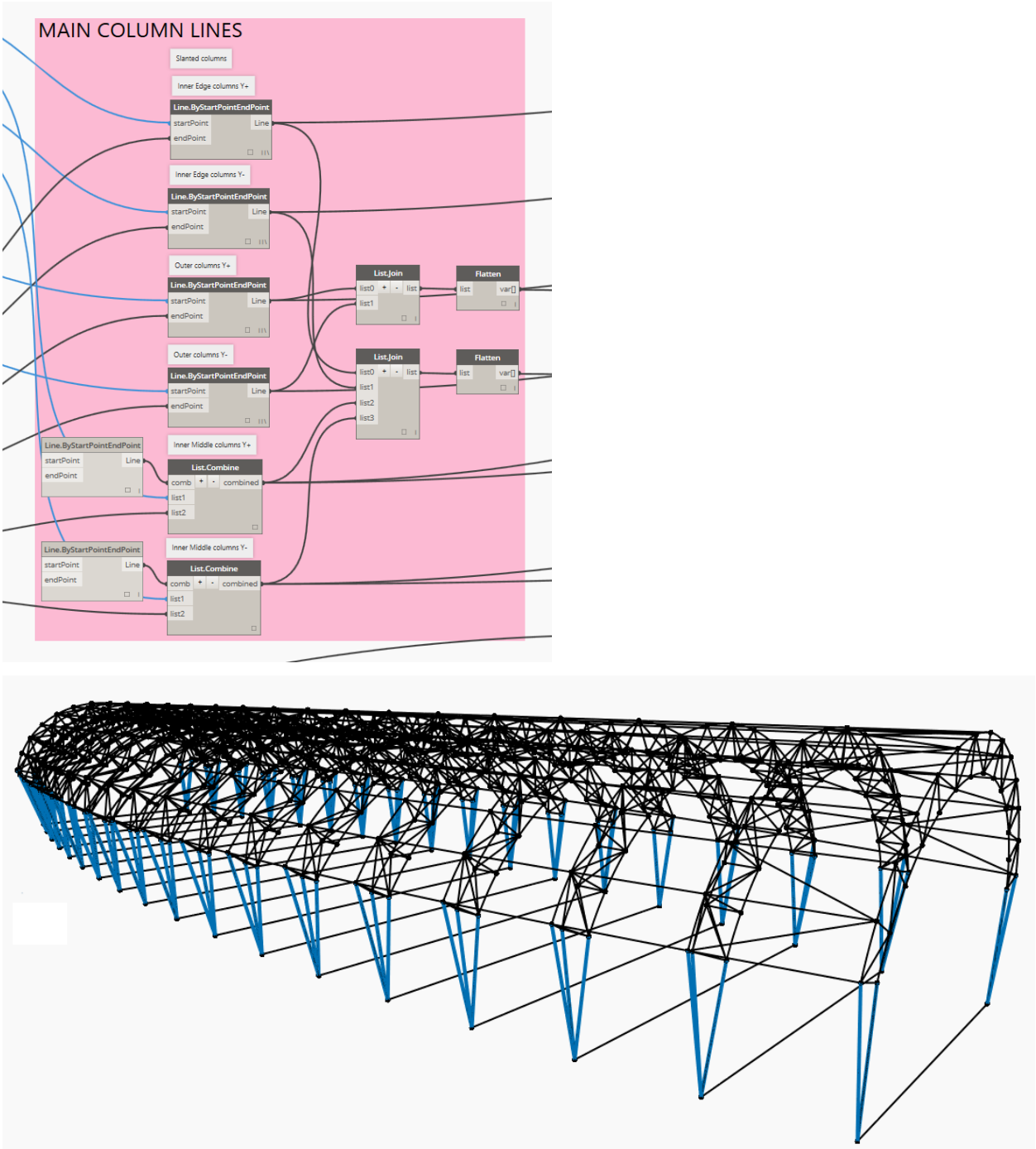
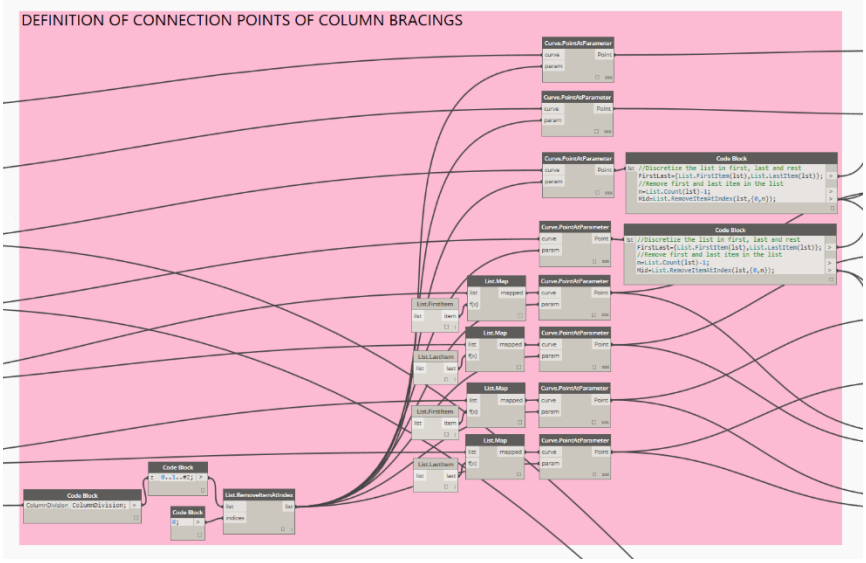


FIG. 15 - GENERATION OF THE MAIN LATTICE COLUMN LINES

Step 7-4 Definition of the connection points for the bracings

The main lines are divided with the *Curve.PointAtParameter* node. The *param* input port is a number sequence between 0 and 1, with a division equal to the value defined for the column truss division in Step 1 - Input parameters.



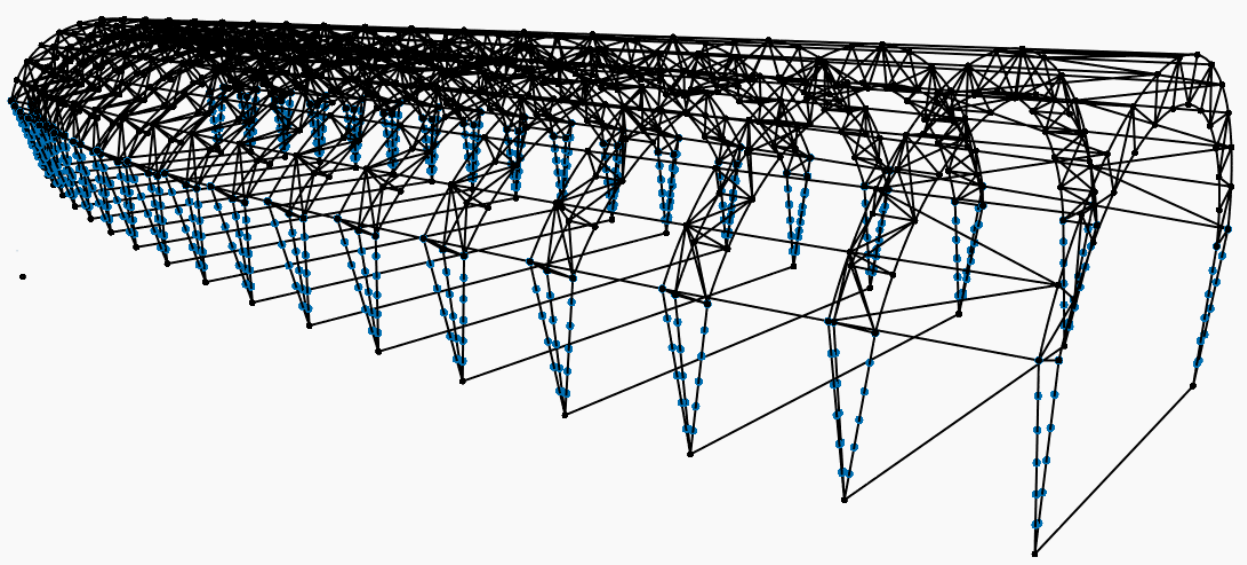


FIG. 16 - DIVISION OF THE MAIN LINES OF THE LATTICE COLUMN

Step 7-5 Lattice column bracings

The points, generated on the column lines are now used to make the diagonals by connecting all the lists with the Custom node “V bracings”.

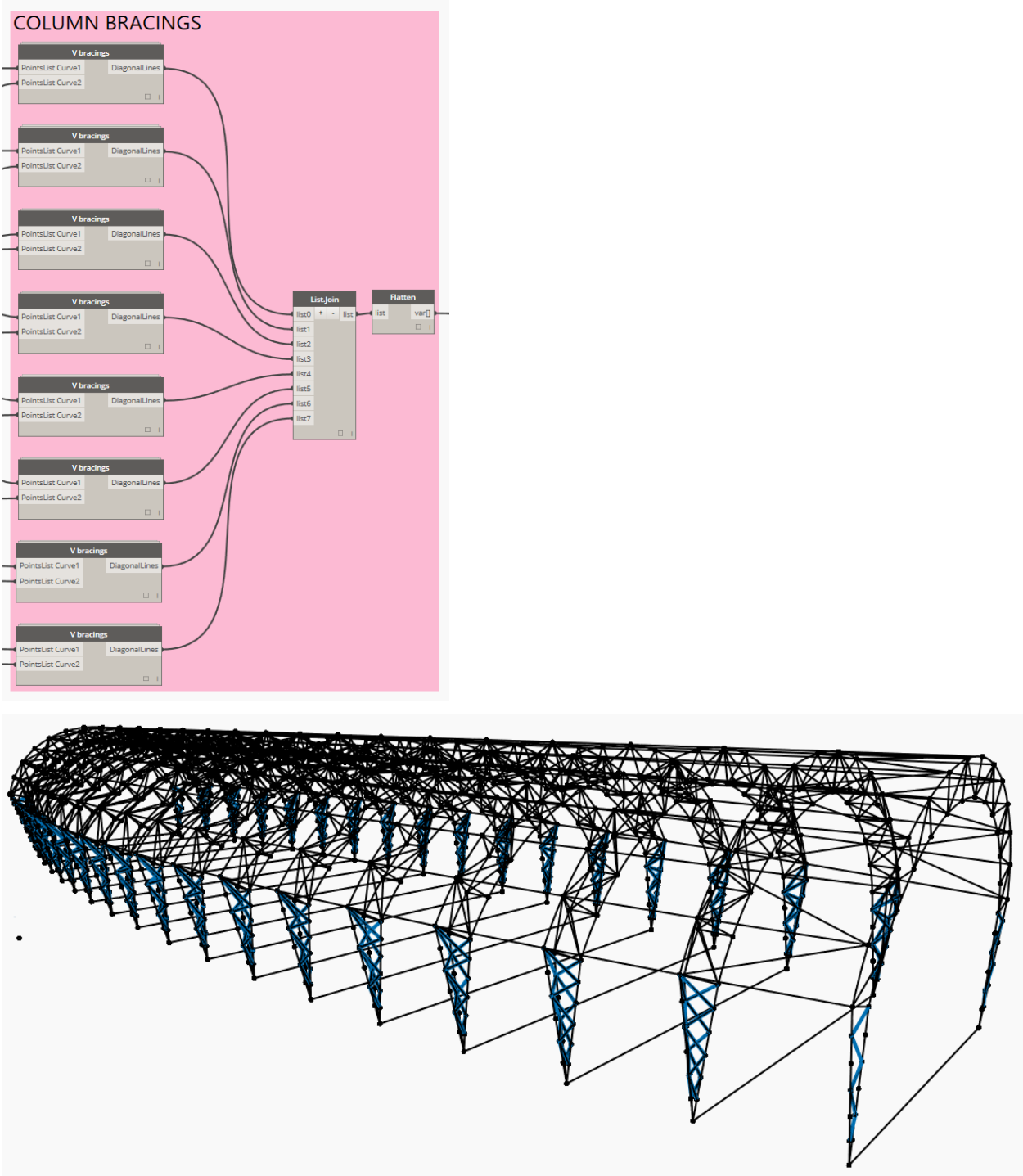
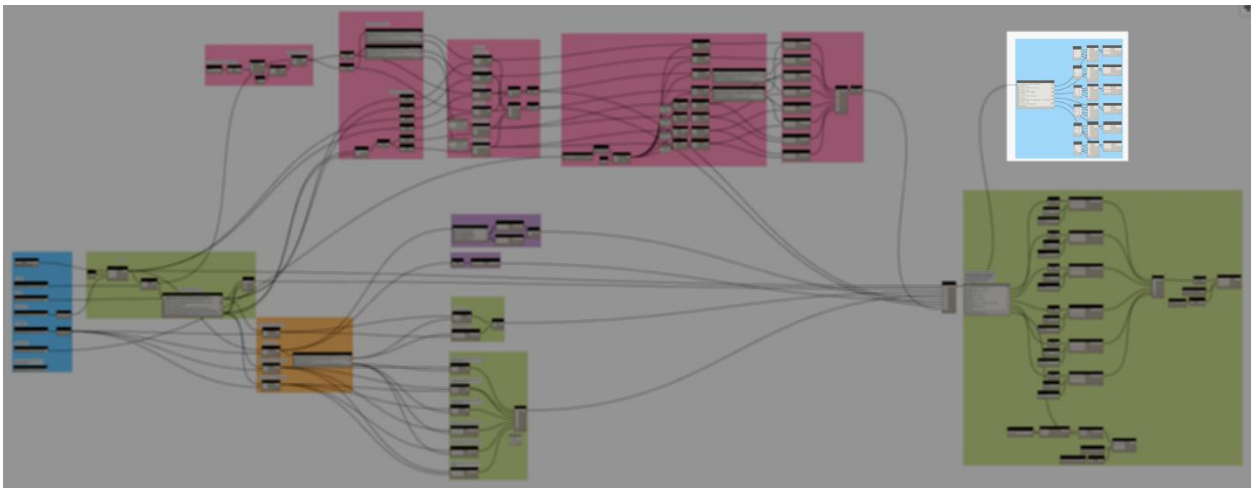
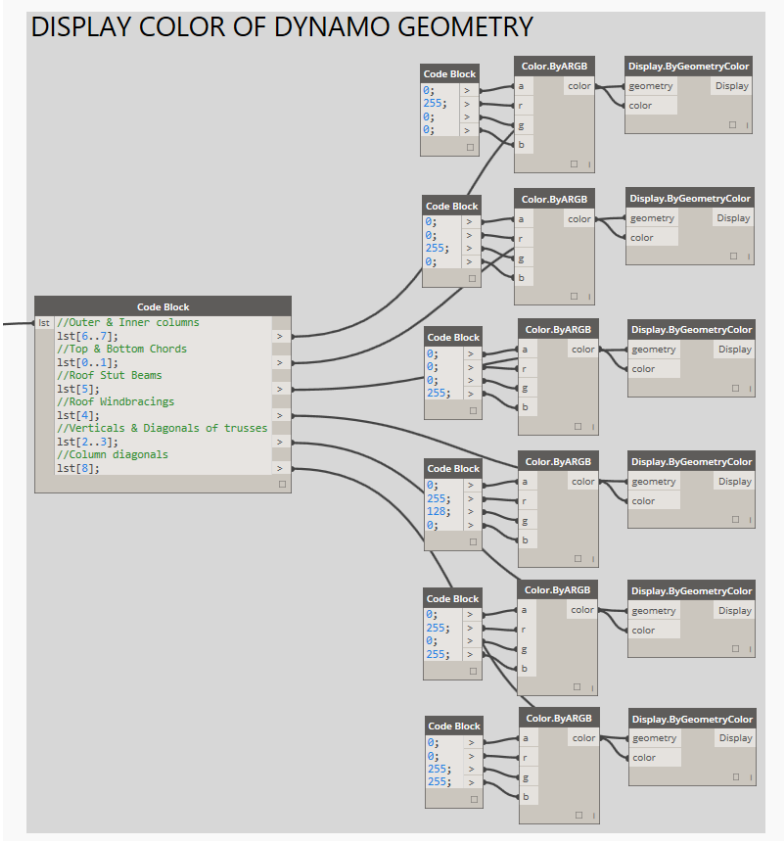


FIG. 17 - BRACINGS IN THE LATTICE COLUMNS

Step 8 - Colorized display in Dynamo



To get a more distinctive view of the structural outline in Dynamo you can apply colours the geometry in the Dynamo canvas. Use the *Color.ByARGB* node to define a colour. The input port “a” represents the alpha channel, which is needed for setting transparency of the geometry. With a = 0 you set the geometry to opaque, with 255 the geometry is fully transparent.



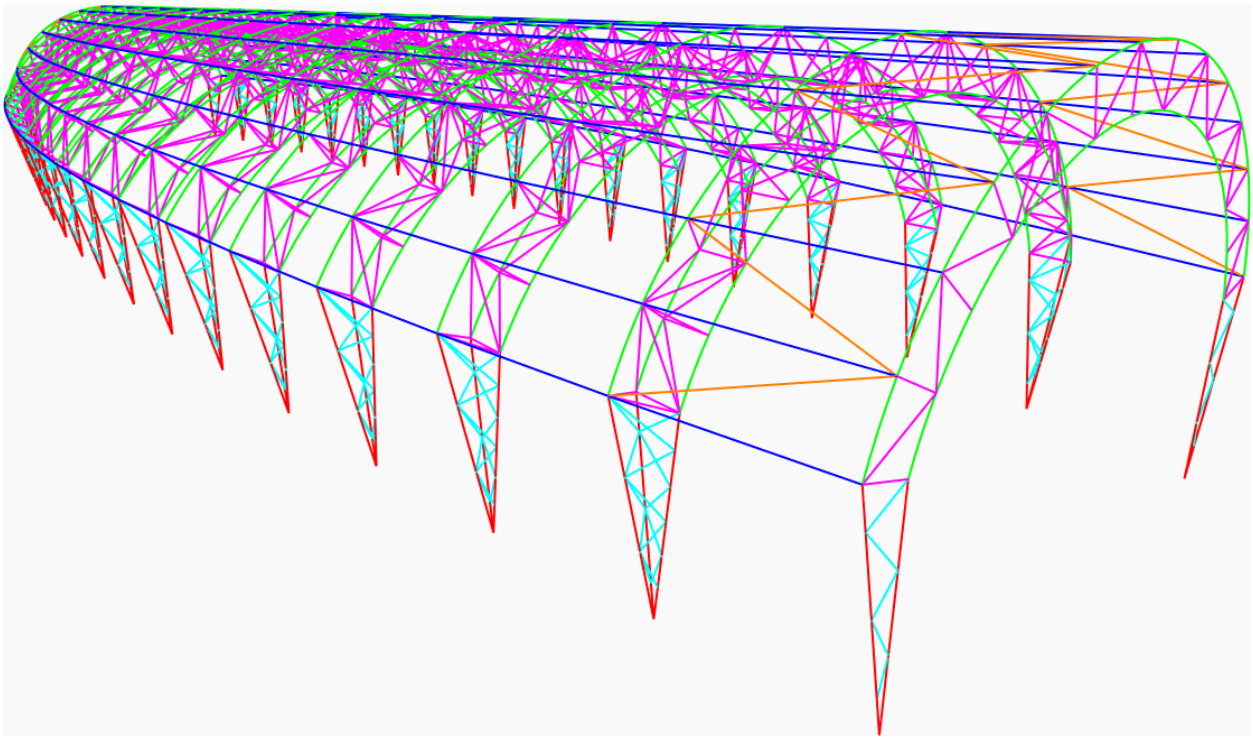
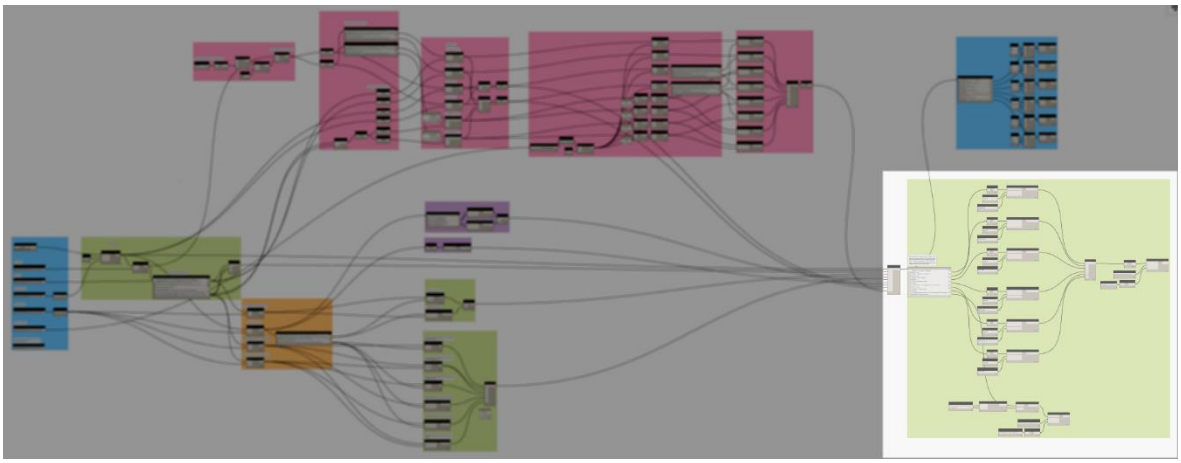
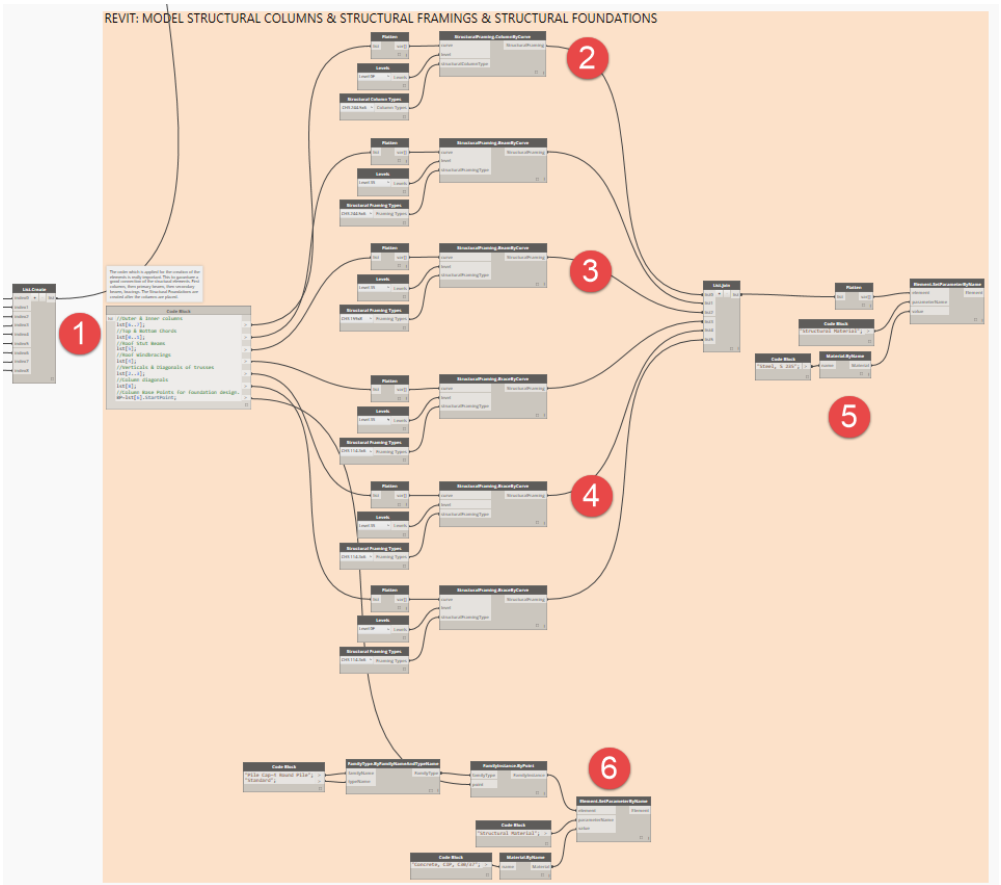


FIG. 18 - COLORIZED DISPLAY OF THE OUTLINES IN DYNAMO

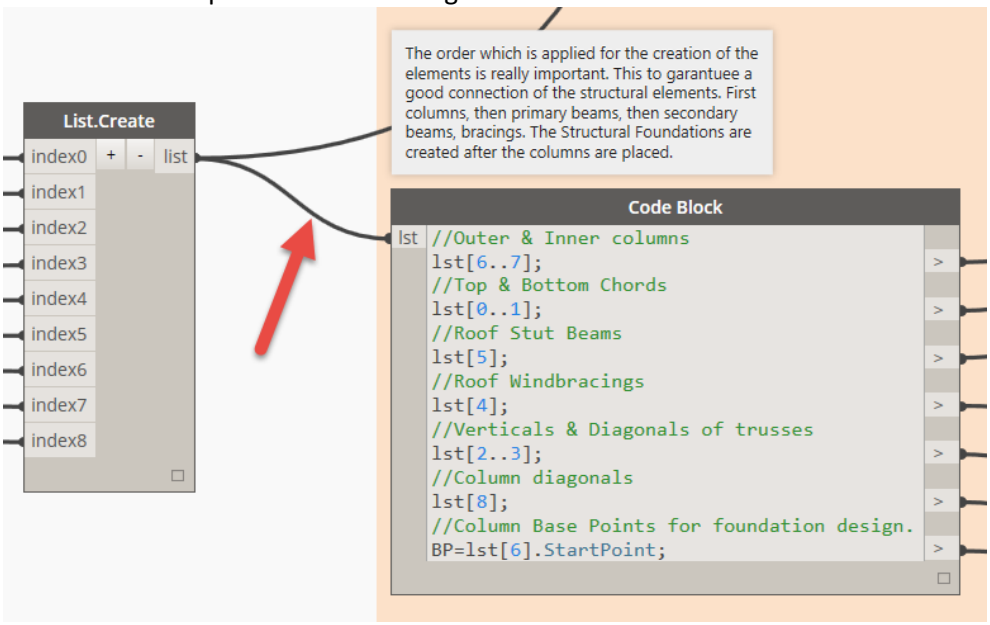
Step 9 – Generation of structural model in Revit



Now the Dynamo geometry is set out, it's time to connect the list of lines with the Revit UI nodes. These nodes can be found in the node library in the *Revit* category. In this last step, all the lists of lines are nicely grouped in the right order with a *List.Create* node, in order to find the right groups.

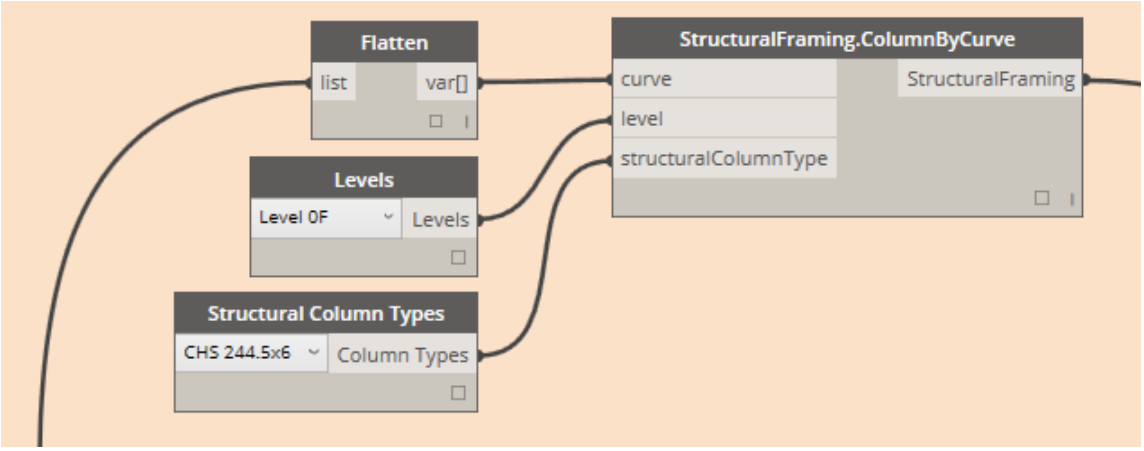


(1) Before you start running the script, it is important to connect *List.Create* node with the *Code Block* node that splits the list in the right order.

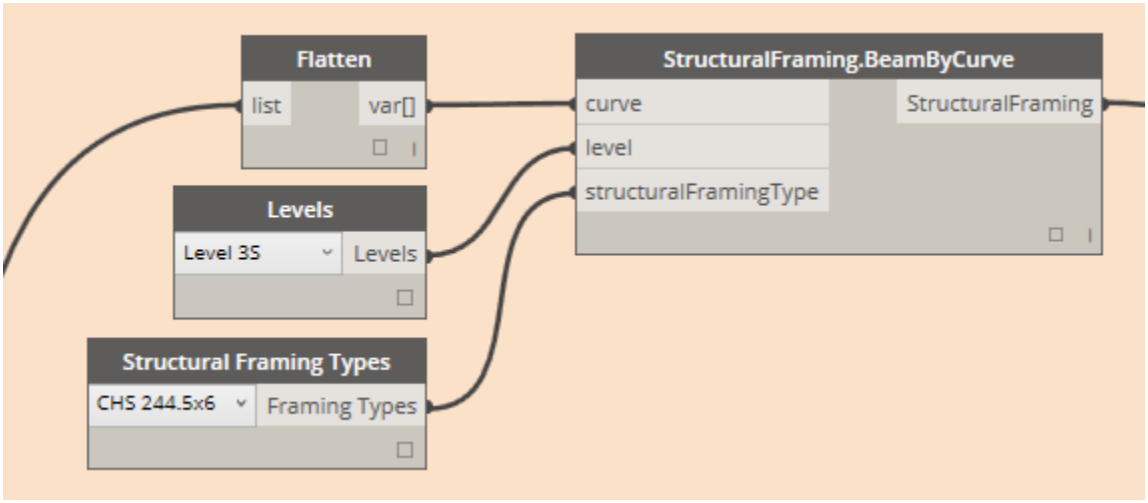


The goal of this code block is to output the Dynamo geometry in the right order. The order which is applied for the creation of the elements is really important. This to guarantee a good connection of the structural elements in Revit. First columns, then primary beams, then secondary beams, bracings. The Structural Foundations are created after the columns are placed.

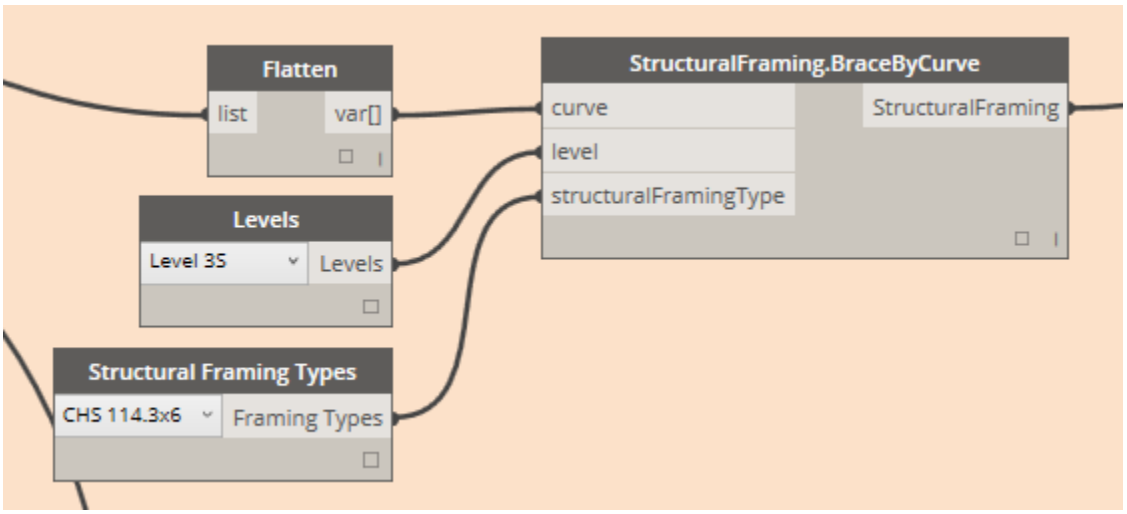
(2) First, the structural columns are created with the *StructuralFraming.ColumnByCurve* node. This node can create vertical and slanted columns. It is important the *Structural Column* family is preloaded in the Revit model, for having it appearing in the *Structural Column Types* node.



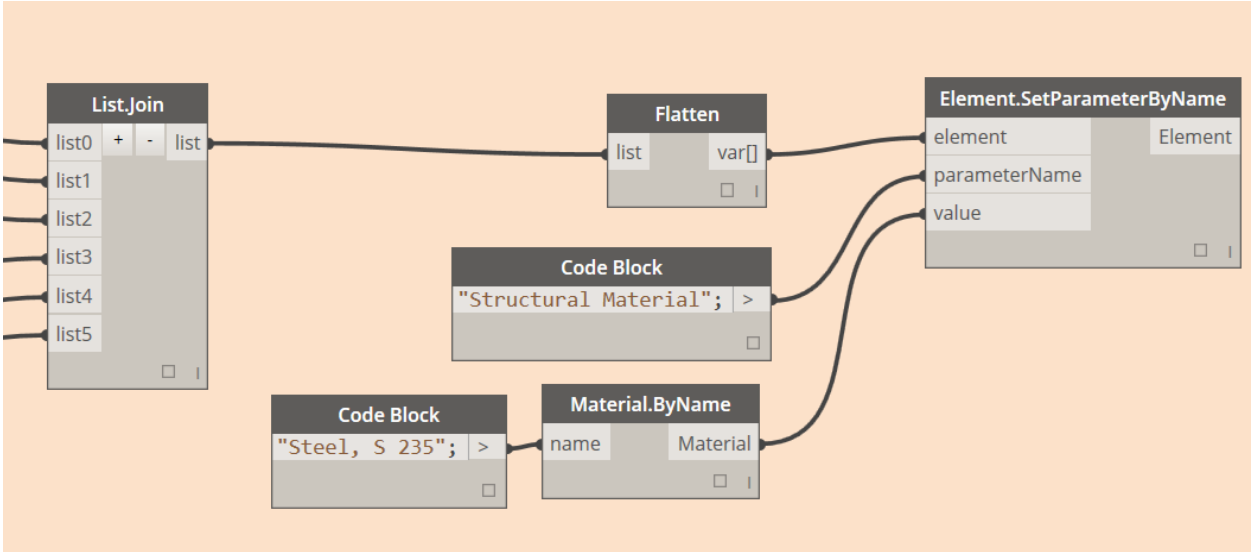
(3) Second, the truss chords, roof strut beams and wind bracings are created with the *StructuralFraming.BeamByCurve* node.



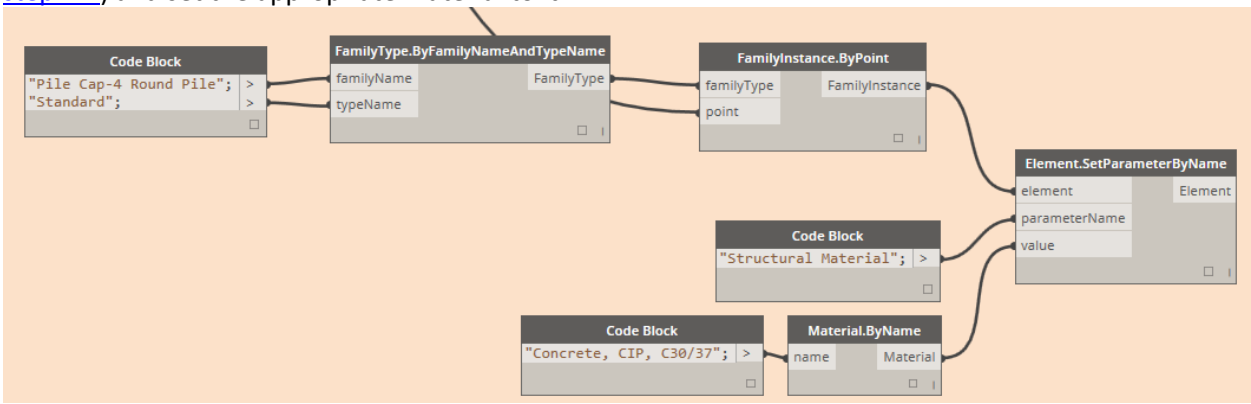
(4) Last, the secondary elements, which are the diagonals in the trusses and lattice columns are created with the *StructuralFraming.BraceByCurve* node.



(5) All the structural columns and framings are grouped in a new list and the material in Revit for these elements is set to i.e. “Steel, S 235”. Again, the material definition should exist in the Revit model and it can be called in Dynamo with the *Material.ByName* node. The *Element.SetParameterByName* node is then used to call the *Structural Material* parameter of all these elements in Revit and set the right value to it.



(6) Optionally, you can add structural foundations to the base of the columns in Revit using the *FamilyInstance.ByPoint* node, which needs the base point of the columns (previously defined in step 7-2) and set the appropriate material to it.



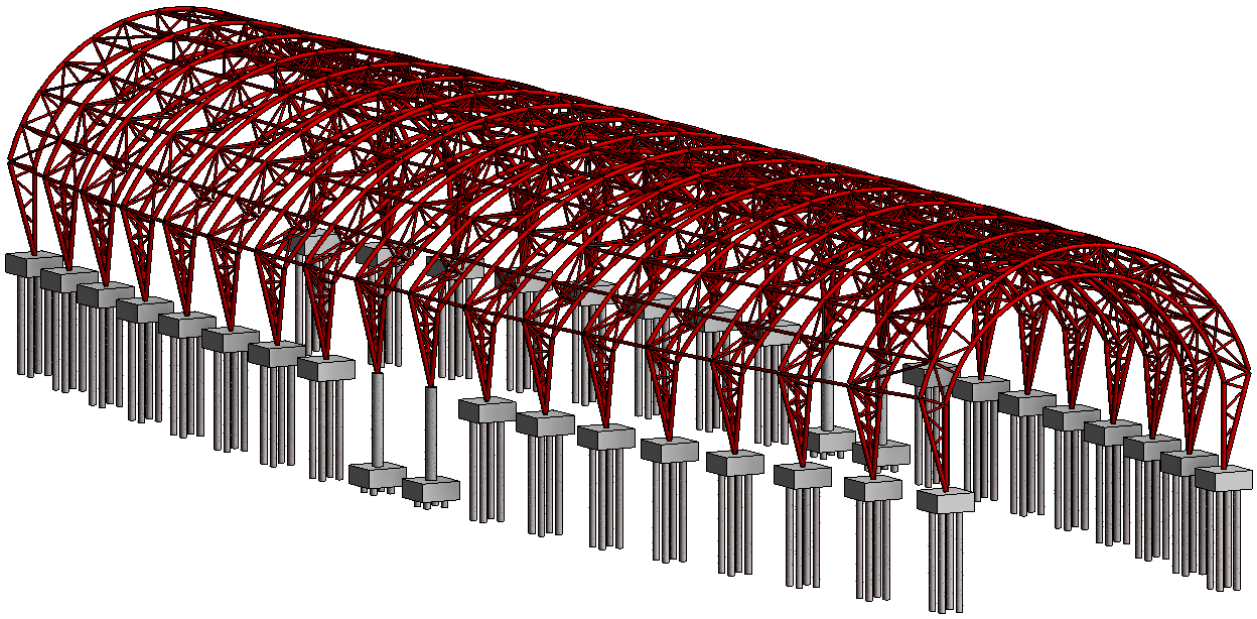
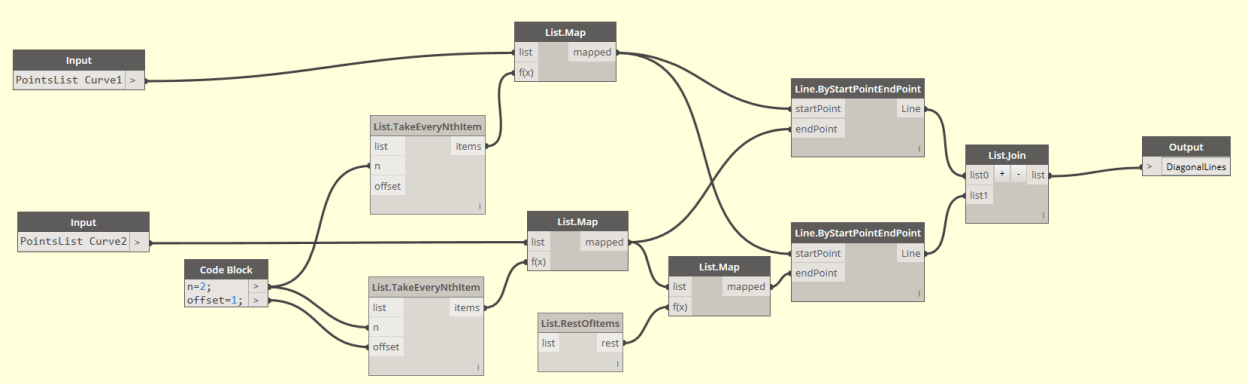


FIG. 19 - FINALIZED DESIGN OF THE ROOF STRUCTURE IN REVIT

Custom node “V bracings”

For the generation of the diagonals in the trusses and the lattice columns, a custom node *V bracings* is made to keep a clean workspace. This custom node combines two lists of points into diagonal lines.

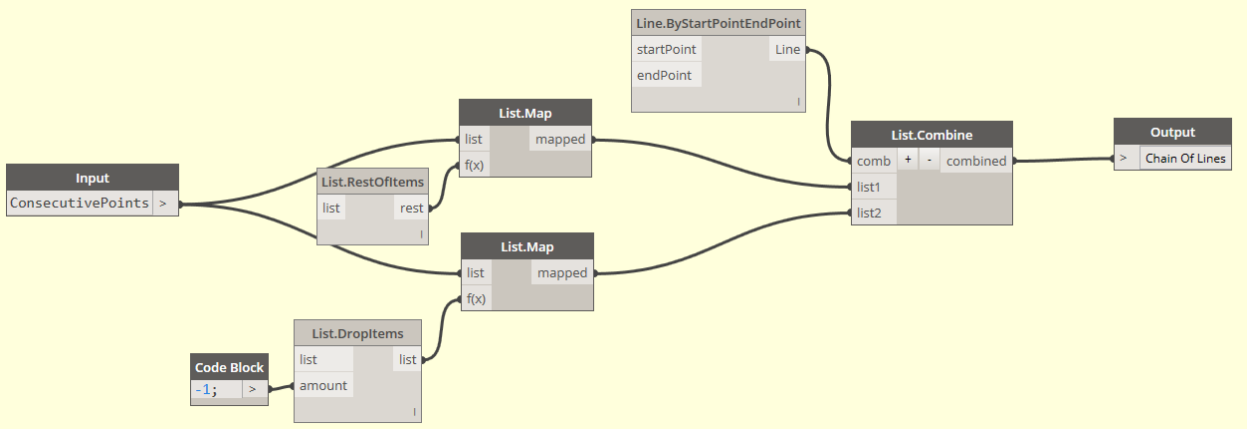
An alternative of this could be “X bracings”. This custom node is added to the dataset is an extra.



Custom node “Chained line”

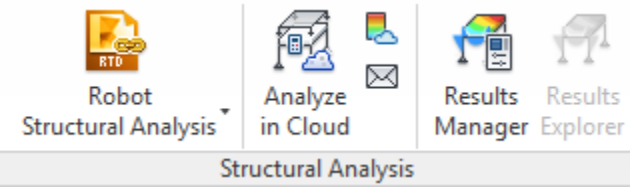
As the roof strut beams cannot be composed of continuous polycurves, and need to be segmented at the points where they meet the top chords, a custom node for making a chained line based on a list of consecutive points is made.

This custom node will be reused later for the discretization or arcs for analysis.

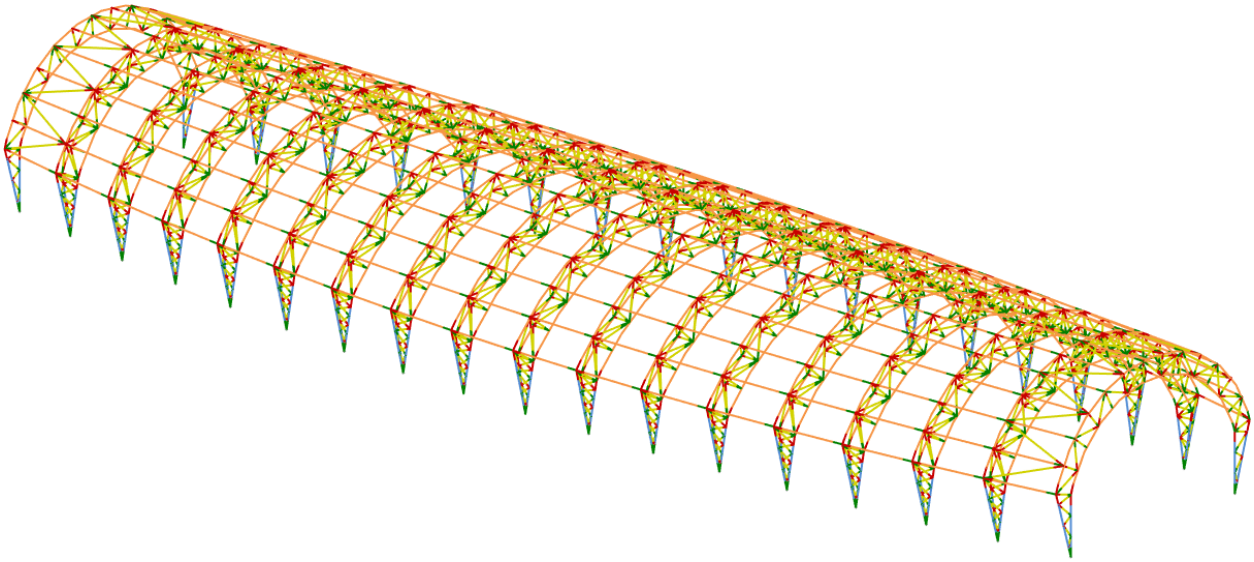


Extended workflow – Link with Robot Structural Analysis

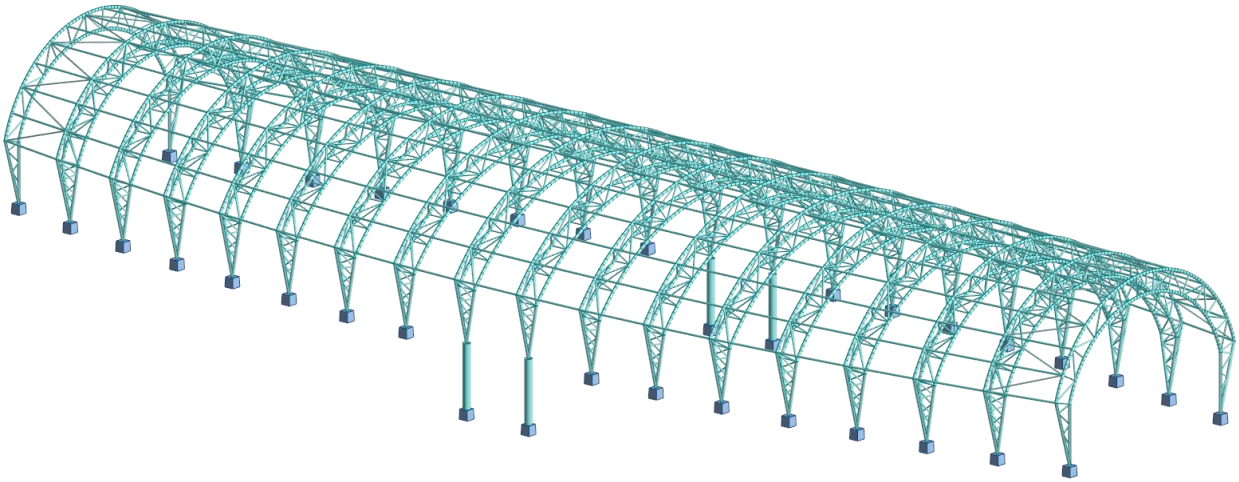
The analytical model of this structure, which is simultaneously created in Revit, can be reused for further analysis by sending it to Robot Structural Analysis with the Structural Analysis tools for Revit.



This toolbar can be found in the Analyze tab in Revit, after installing the Structural Analysis Toolkit 2016 from Autodesk Exchange.



As we are going to generate the analysis model from within Dynamo for specific computational optimization purposes, we won't go into detail on this workflow.



Case 2 – Complex analysis model in Robot Structural Analysis

In this second case we will build the complex analysis model of the *Armadillo Roof* in Robot Structural Analysis, using a parametrized approach with Dynamo, based on the initial design of the architect done in Revit.

The base geometry of the roof is a SAT export of the Mass object initially created in Revit by the architectural designer.

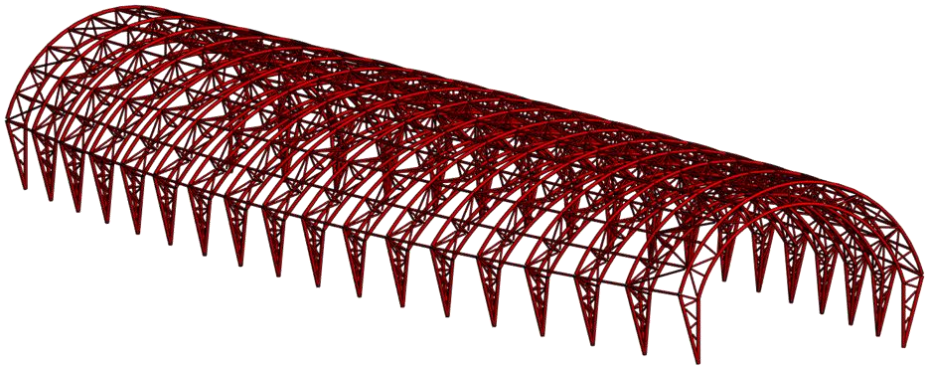


FIG. 20 - MILLENNIUM RAILWAY STATION - ARMADILLO ROOF - ANALYSIS MODEL

DATASETS

RSA



Start model:
Armadillo Platform - Analysis Model_start.rtd
(Armadillo_start.rstructure in case you use React Structures TP 2)
Finalized model:
Armadillo Platform - Analysis Model.rtd

Dynamo

Workspace:
Armadillo Platform - Analysis Model.dyn
Custom nodes:
V bracings.dyf
Chained line.dyf
External input
Platform Roof.sat

To open the dataset files:

1. Launch the sandbox version of Dynamo 0.9 (C:\Program Files\Dynamo 0.9\DynamoSandbox.exe) or use Dynamo Studio 2016 (v. 0.9.3072)
2. Open the Dynamo script (.dyn)
3. Launch Robot Structural Analysis 2016
4. Open the RSA start model file. Without this step the script won't work!
5. Connect the wires as shown in the class and play with the parameters.

General overview of the script

The Dynamo script to generate the analysis model in RSA is basically the same as the one used in our Case 1. In fact it reuses step 1-7 from the script used for the generation of the structural model in Revit.

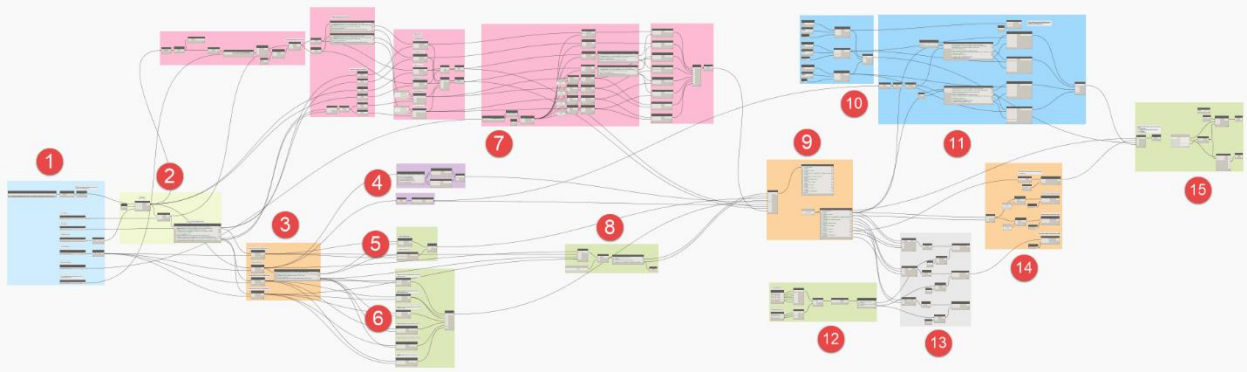


FIG. 21 - DYNAMO SCRIPT ANALYTICAL MODEL - OVERVIEW

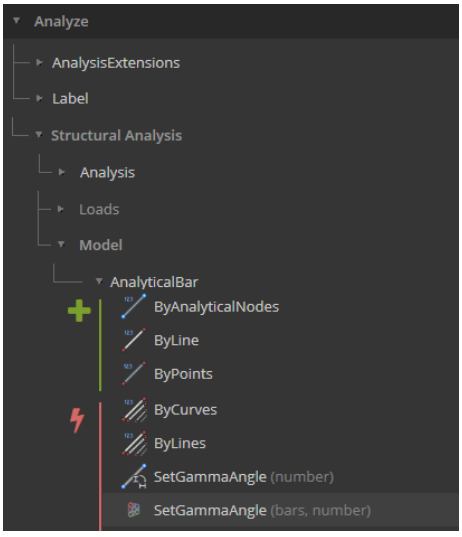
Step 1 = input parameters, such as the basis roof surface, truss geometry …

Steps 2-7 = generation of the model outline in Dynamo

Step 8 = discretization of the arcs for analytical model generation

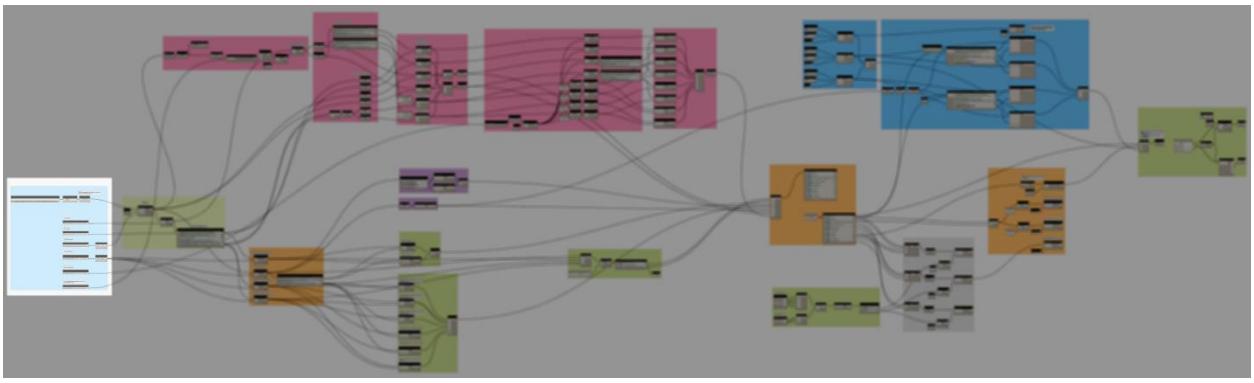
Steps 9-15 = generation of the simulation model in RSA and analysis

Each of the steps is described in the chapters below.



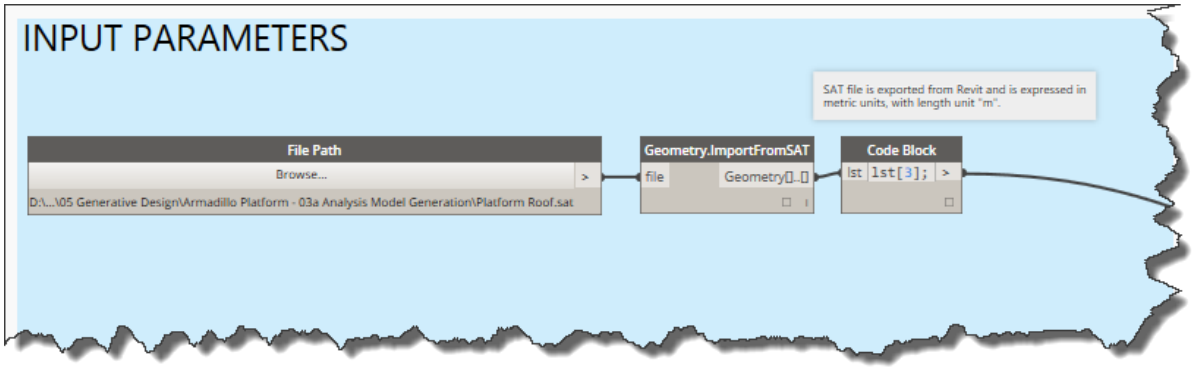
The nodes for Structural Analysis connection can be found in the Analyze > Structural Analysis category in the Dynamo nodes library.

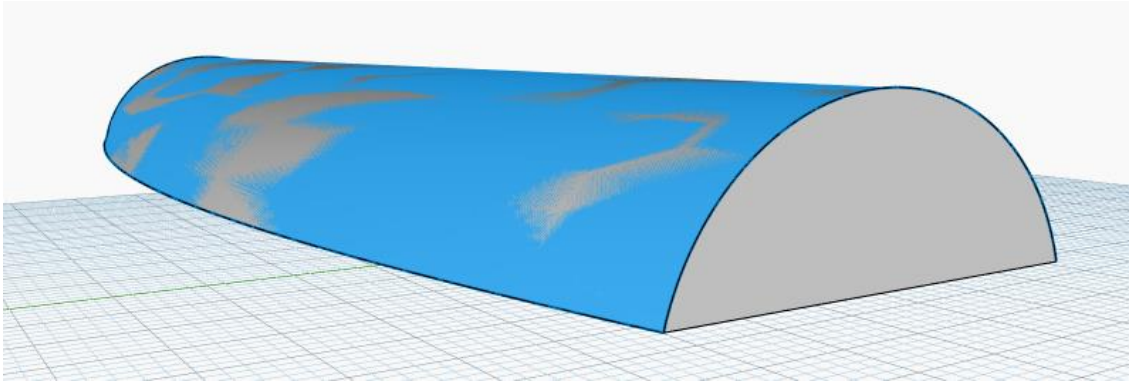
Step 1 – Input parameters



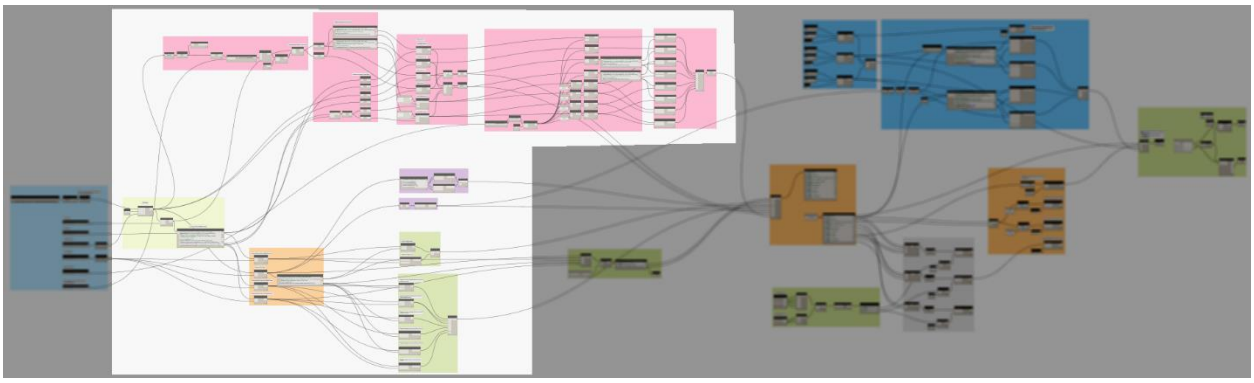
As in this case we are not working with a live link with a Revit model (so called element binding) we need an external reference for the roof surface geometry. This is done by making an SAT export from the mass model of the architectural roof. This SAT export is already done and is available through the "Platform Roof.sat" file. The other input parameters (for truss geometry) are described already in Step 1 – Input parameters of Case 1.
The import of the SAT file in Dynamo is done with the *Geometry.ImportFromSAT* node. Make sure you point to the right folder and file for source file, by clicking the *Browse* button on the *File Path* node. As we only need the top surface of the geometry, a *Code Block* is used to point to the 3rd item (lst[3]) in the list of surfaces from the SAT import.

The SAT file is defined in metric units "m". So make sure the values of the parameters you use are also in "m" and not in "mm as in case 1. The "m" unit is also the default one to use in Robot Structural Analysis.

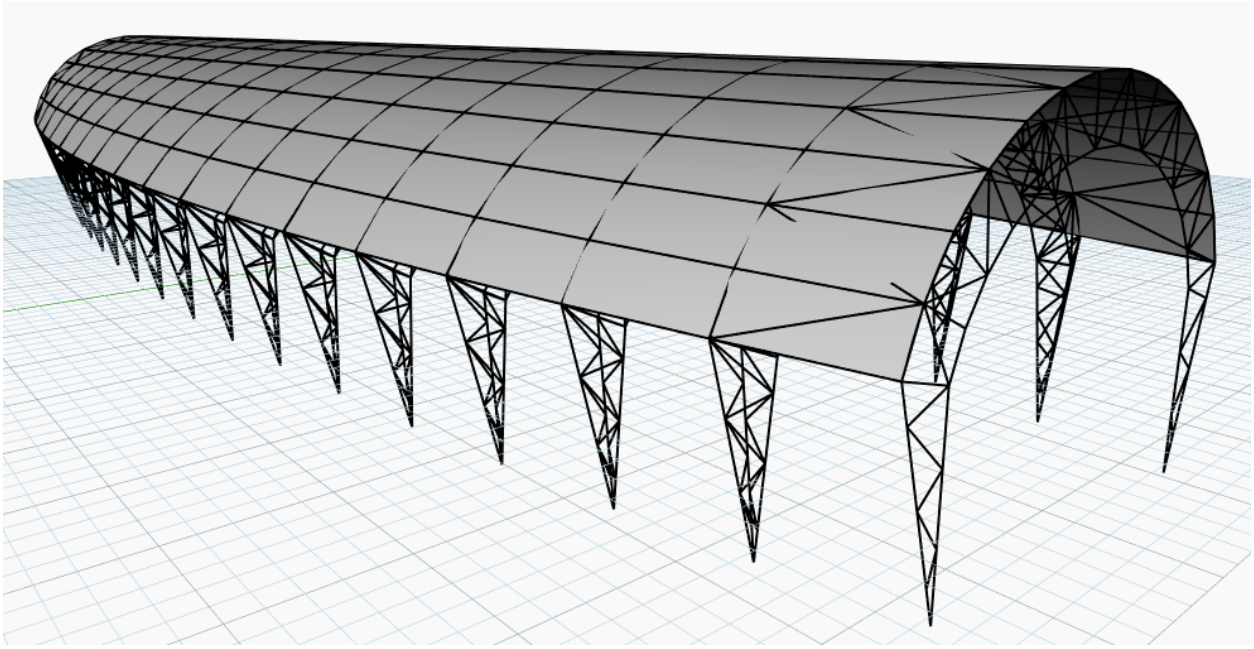




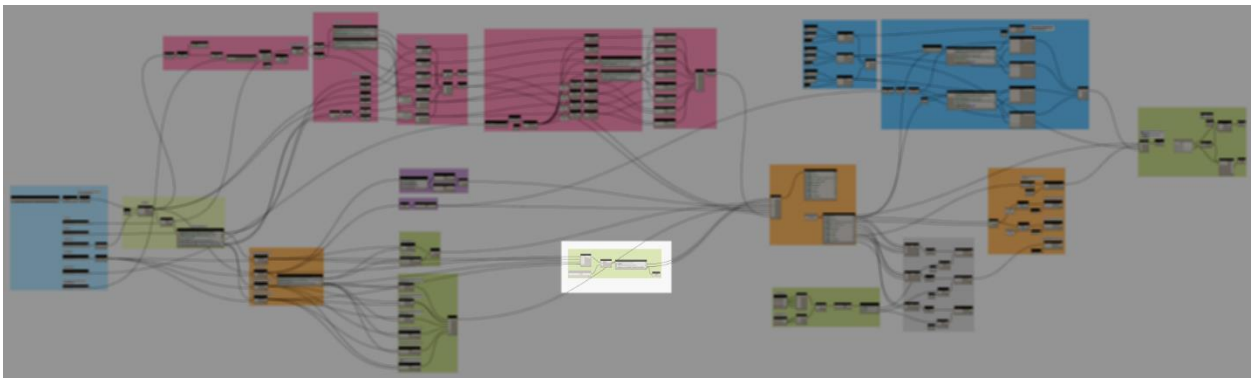
Step 2 to 7 – Creation of Dynamo geometry

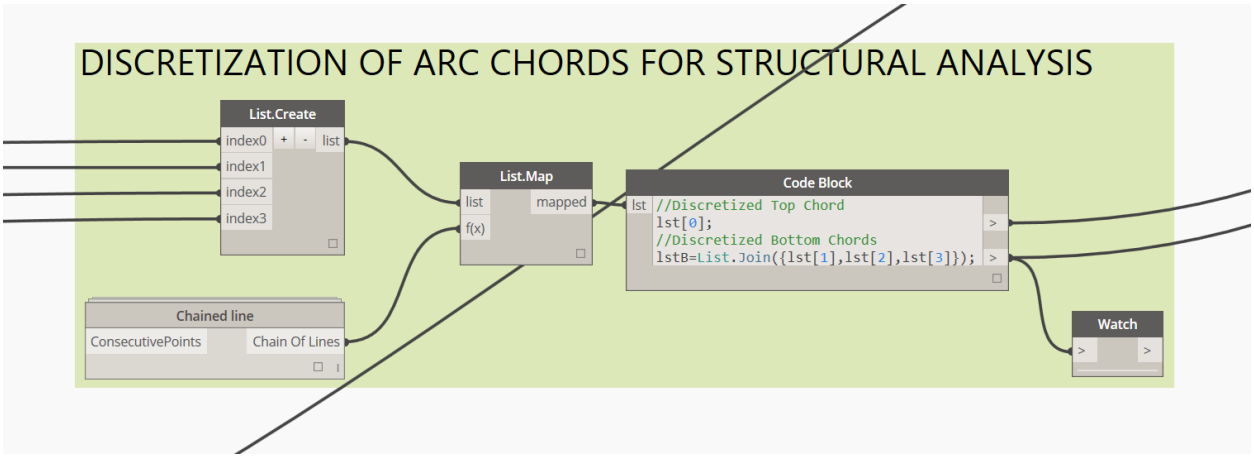


These steps are discussed in detail in Case 1.



Step 8 – Discretization of arcs for structural analysis model



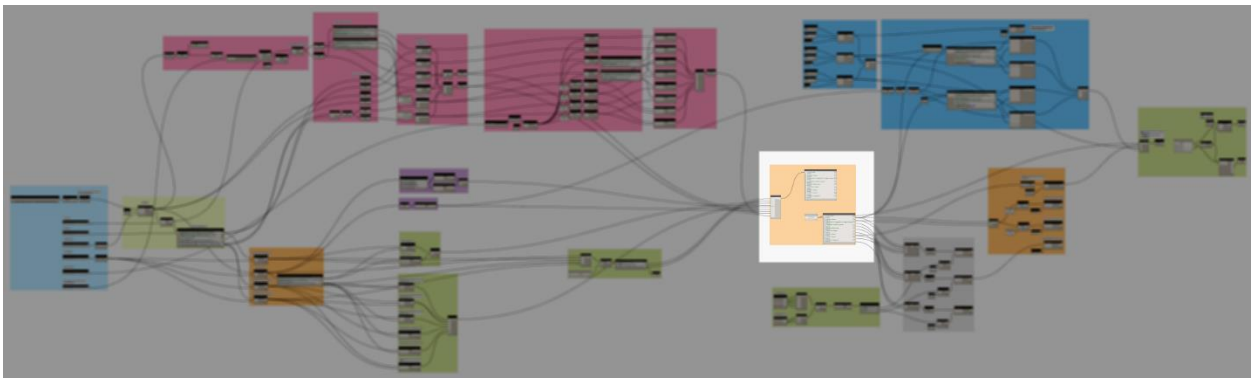


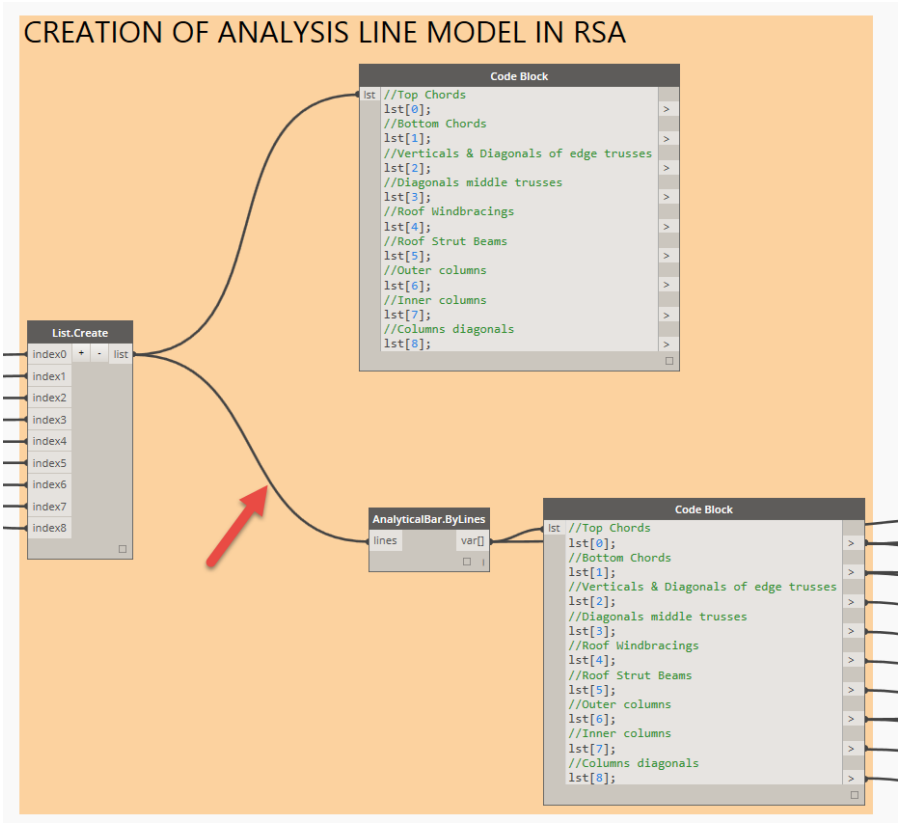
In Robot Structural Analysis it is not possible to model a pure arc. The arc is approached by means of discretization. This means we need to "replace" the arc lines for the trusses in the Dynamo model with segmented lines approaching the arc. There are several ways to do this. The best one is to reuse the points that were generated on the chords and to combine them with the Custom node "Chained line", which was previously used in Case 1.

Important note:

In case you might consider the *Curve.Division* node to achieve this, I would advise not to do this, as it will cause the usage of 2 different methods to calculate coordinates of the same Dynamo point (one with the *Curve.PointAtParameter* and one with the *Curve.Division* node) and based on the calculated coordinates they are not on the exact same location for RSA. This will turn into conflicts when making changes to the geometry.

Step 9 – Creation of analysis line model in RSA





Now the Dynamo geometry is set out, it's time to connect the list of lines with the **Structural Analysis for Dynamo** nodes. These nodes can be found in the node library in the *Analyze > Structural Analysis* category.

In this step, all the lists of lines are nicely grouped in the right order with a *List.Create* node, in order to find the right groups. Don't forget to connect the output from the *List.Create* to the *AnalyticalBar.ByLines* node. This node is specifically design for the faster creation of multiple lines to analytical bars in RSA.

The *AnalyticalBar.ByLines* node is on its way connected with a new Code Block that makes distinction of the elements, in order to apply the right loads, boundary conditions, and so on…

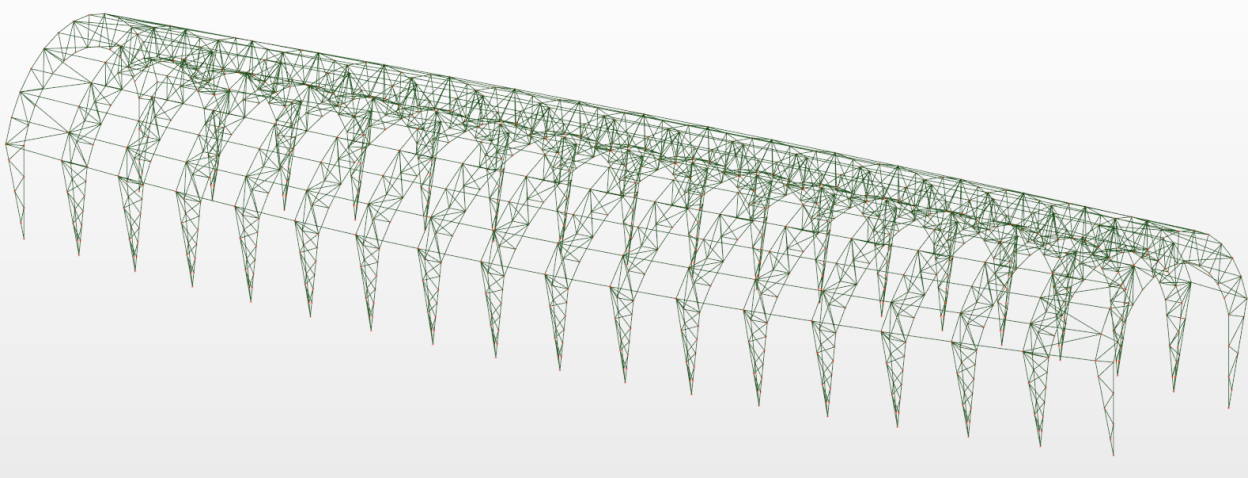
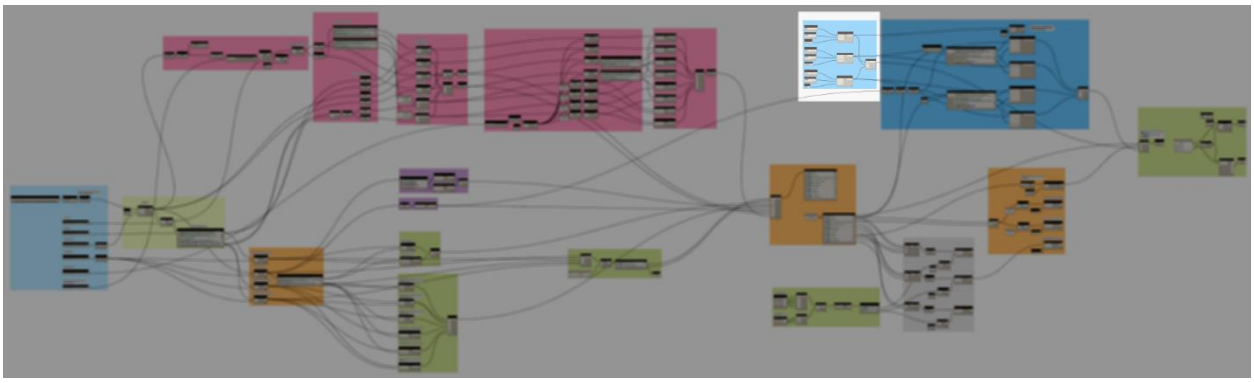
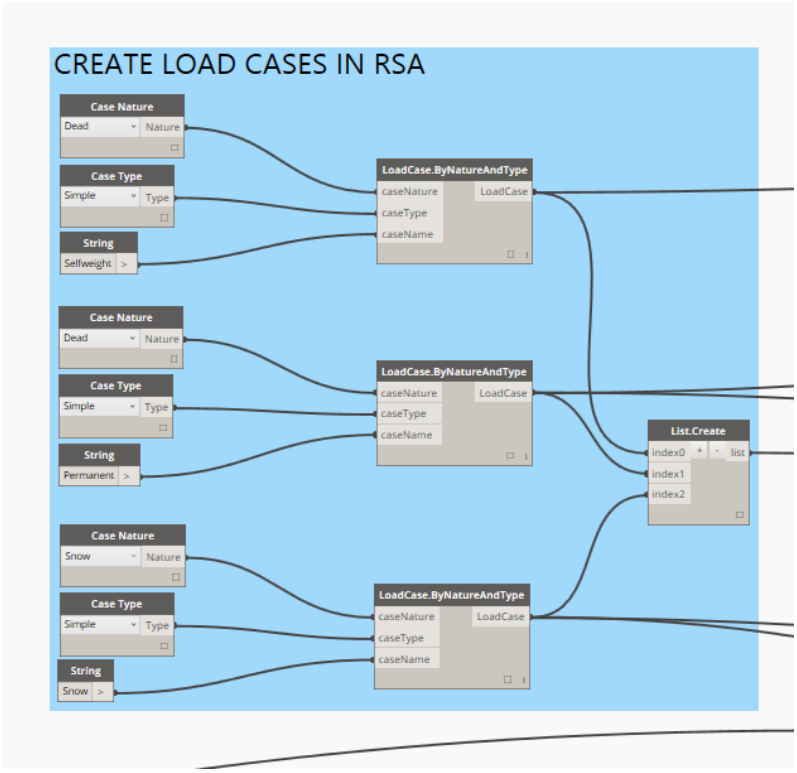


FIG. 22 - ANALYTICAL MODEL CREATED IN RSA

Step 10 – Creation of load cases in RSA





Next is the definition of the load cases in RSA. These load cases will hold the loads defined in step 11.

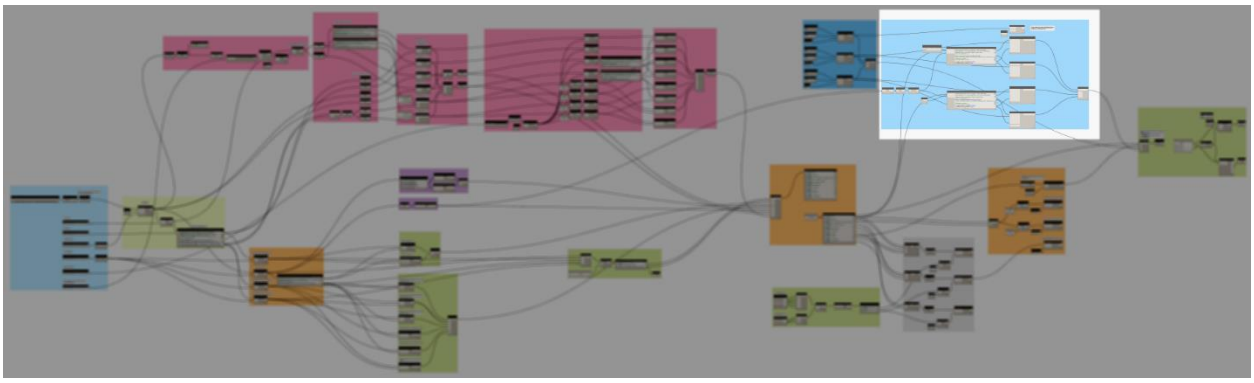
A load case can be created with the *LoadCase.ByNatureAndType* node. This node requires three inputs:

Case Nature:
dead load, live load, wind, snow …

Case Type:
At this moment there is only “simple”

Case Name:
This is a string representing the name of the load case in RSA.

Step 11 – Apply loads to the roof beams

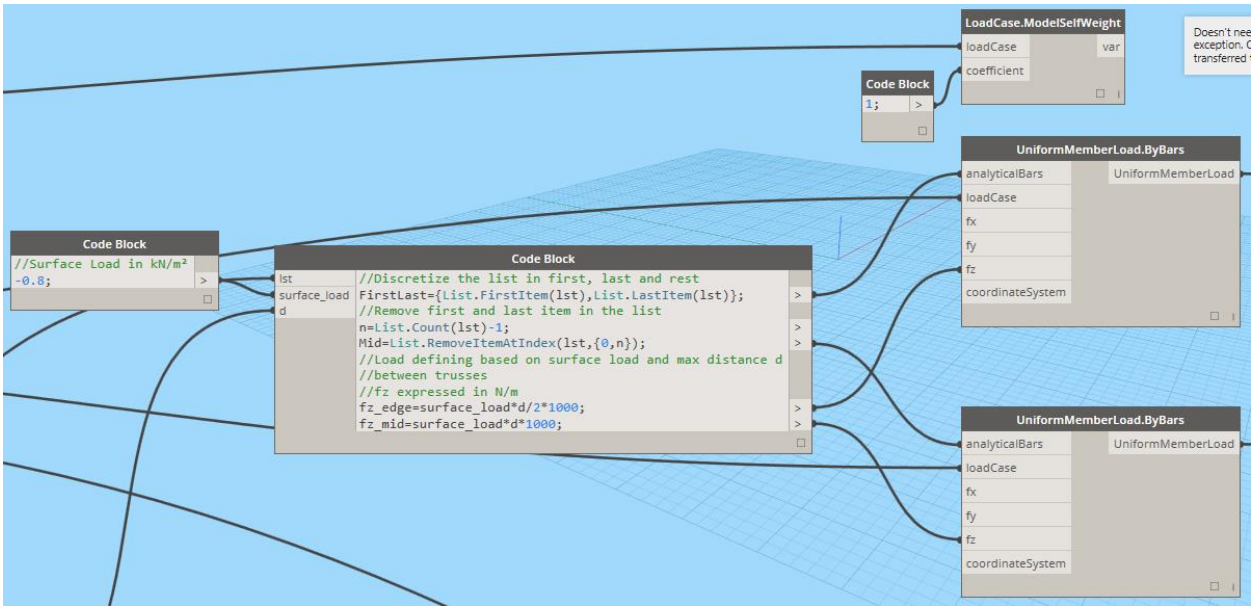


There are two type of loads used in this script:

- **Selfweight**: which can be applied simply with *LoadCase.ModelSelfWeight* node. In the *loadCase* input port, you connect the appropriate load case for selfweight.

- **Imposed loads on the structure**: The only parts that need to be loaded in this structure are the top chords. They carry the roof panels. In this example we will calculate the resulting line loads on these beams as a result of a surface load on the panels (taking into account the distance between the trusses). This is done with *Code Block* node as illustrated below. The line load is created in RSA by the *UniformMemberLoad.ByBars* node.

  The input of this node for fx, fy and fz is **expressed in N/m**, so make sure you use the right values and units. With the *coordinateSystem* input it is possible to set the orientation of the load, which is by default the WCS.



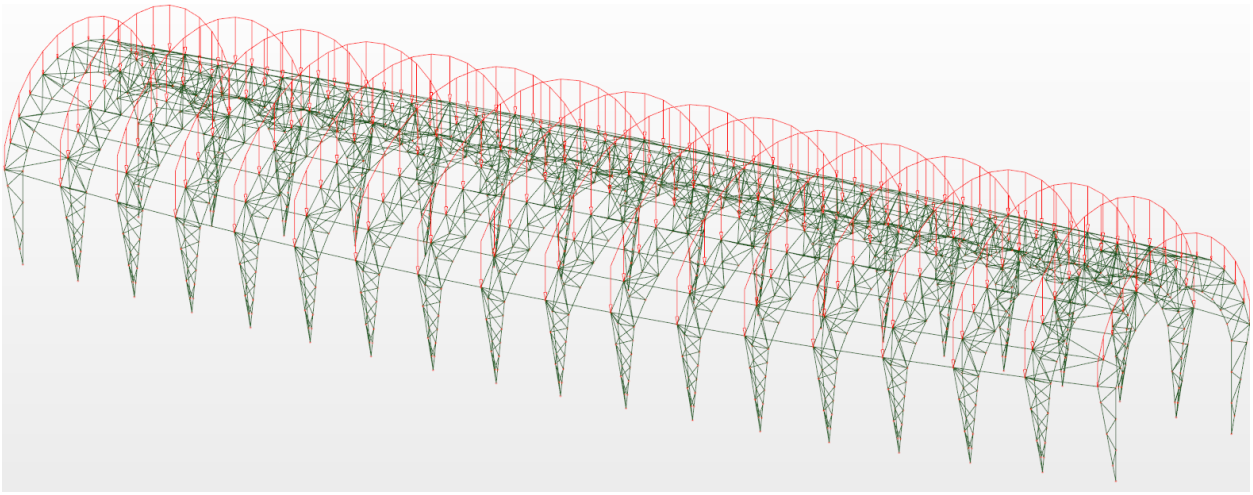
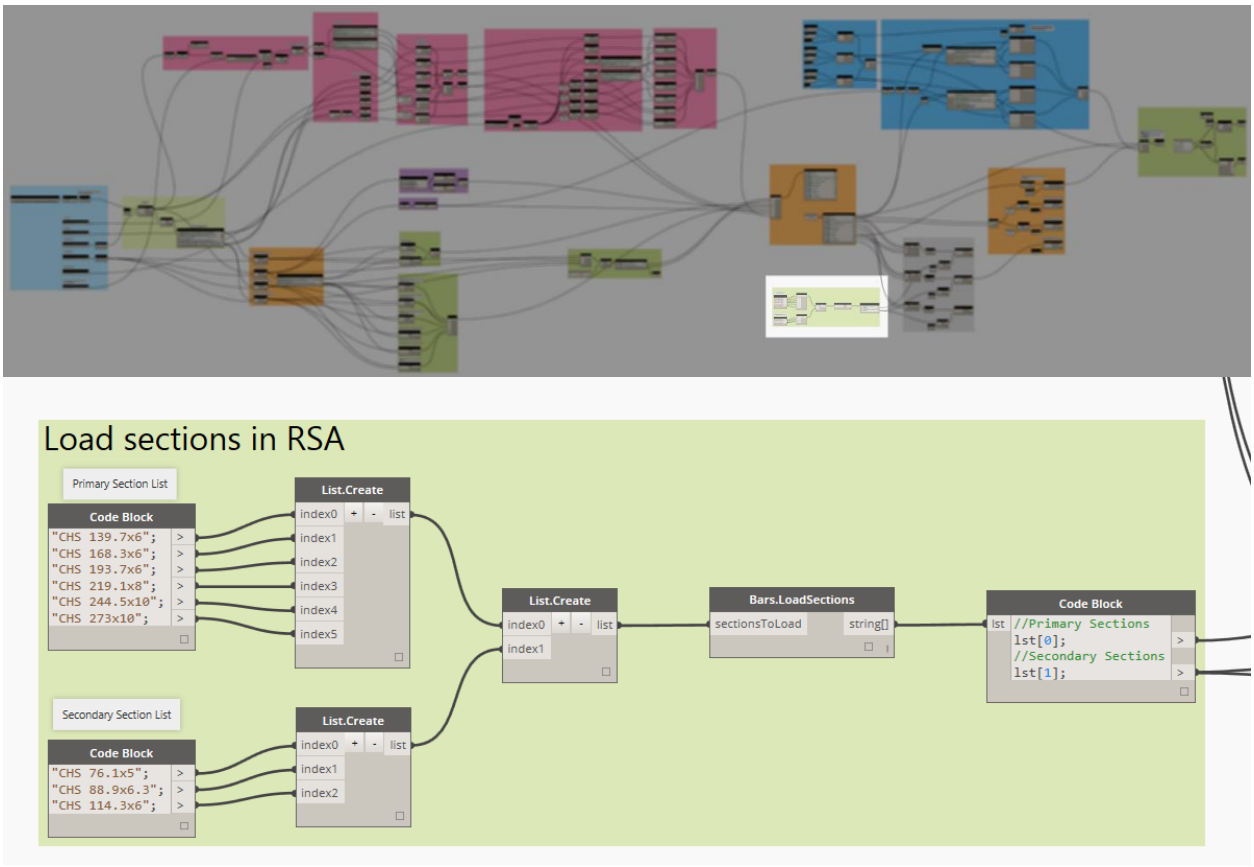


FIG. 23 - LOAD DEFINITIONS IN RSA

Step 12 – Load sections in RSA



Before you assign cross-section properties to the analytical members, you need to make sure the sections are loaded in RSA. This can be done already in the RSA start file you use, but this can also be performed in Dynamo with the *Bars.LoadSections* node.

Make a list of section names (strings) and connect them to the *sectionsToLoad* input port. When you want to reuse the list of the sections, then better use the output of the *Bars.LoadSections* node, as you are sure that they are loaded in the actual document then.

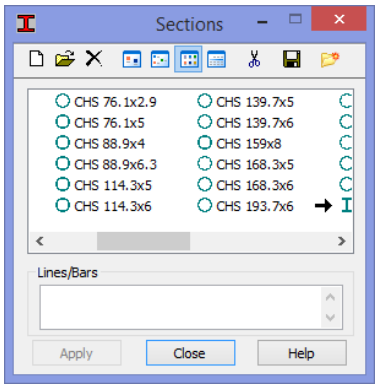
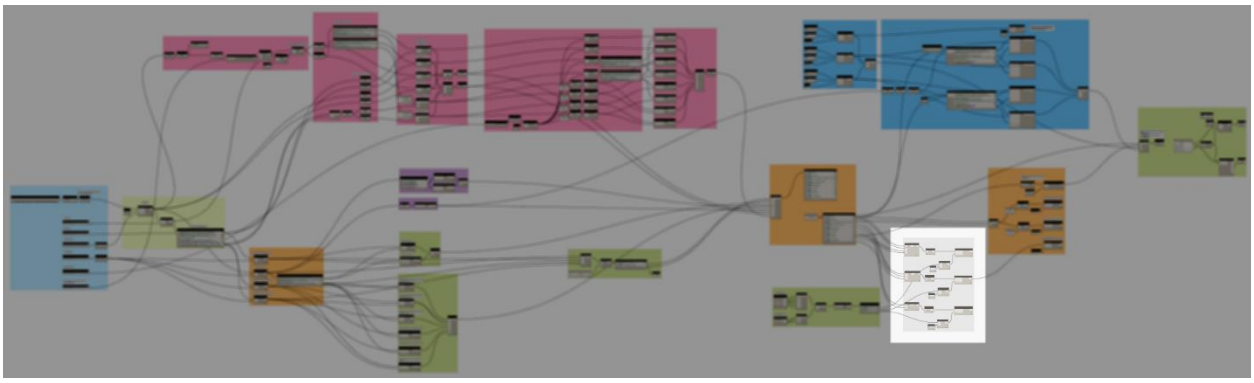
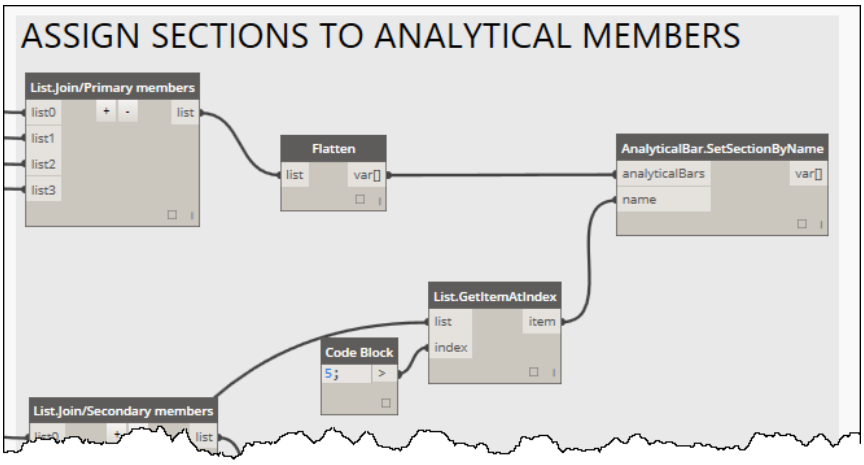


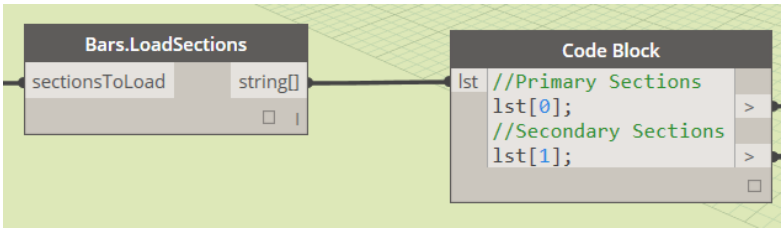
FIG. 24 - LOADED SECTIONS IN RSA

Step 13 – Assign sections to analytical members



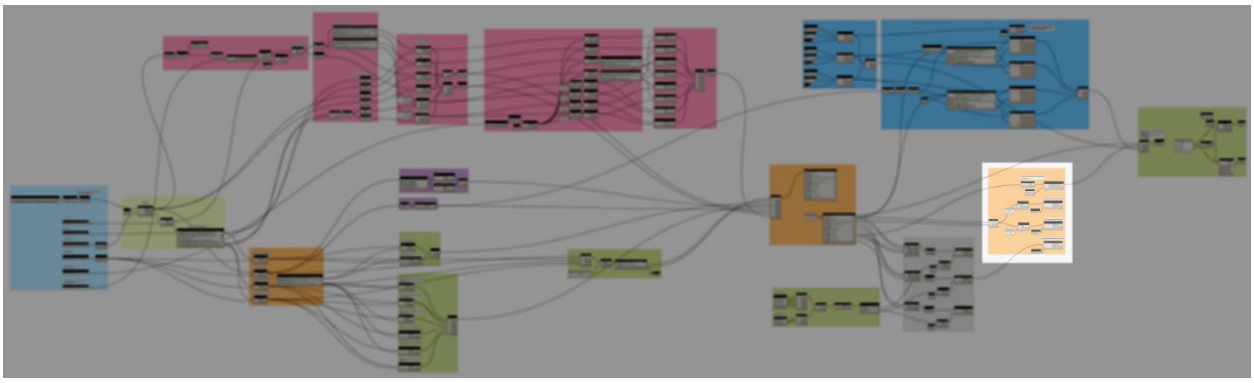


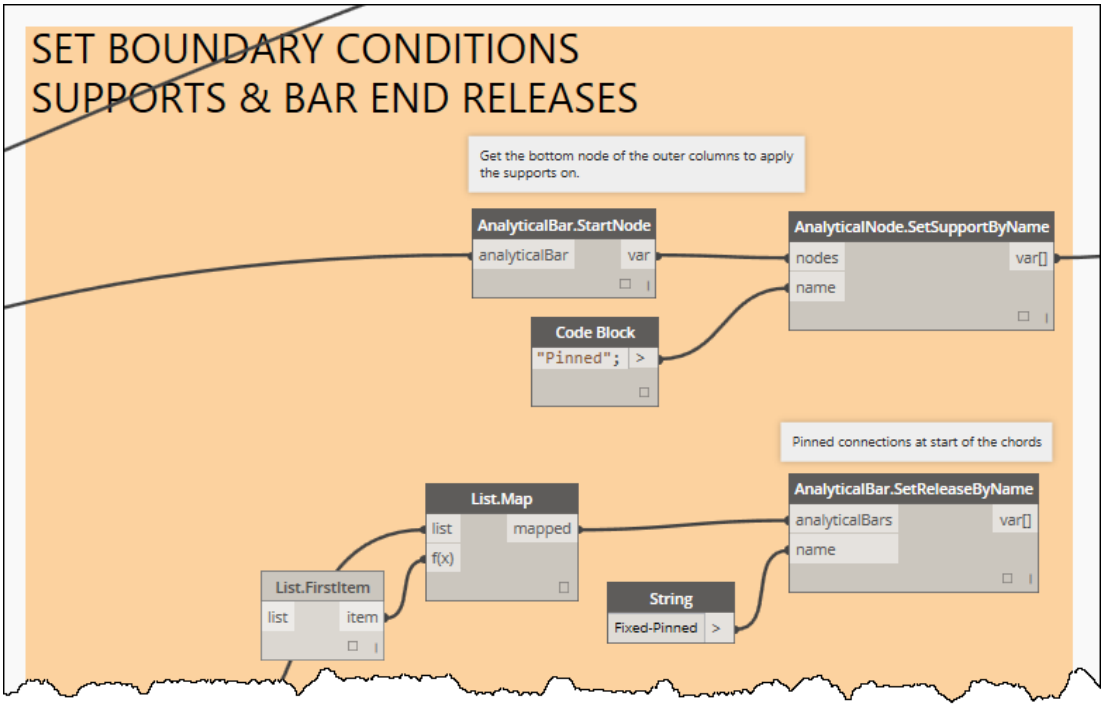
If the sections are available in the active RSA project, then you can start assigning the right sections to the right analytical bars with the *AnalyticalBar.SetSectionByName* node. There exist two nodes for assigning the sections. One with *AnalyticalBar* as input port (which will process the bars one by one) and one with *AnalyticalBars* as input port (which will select all bars and assign the chosen section in a single operation). This last one is advised for this type of structure.
The *name* input port accepts strings for the section names. As we work with a list of sections in this case, the choice of section can be done by connecting the section list from Step 12 to the *List.GetItemAtIndex* node and choose the appropriate section by pointing to the right index.



There is distinction made between Primary Sections (truss chords and roof beams) and Secondary Sections (truss diagonals and wind bracings).

Step 14 – Set boundary conditions : Supports and bar end releases





The boundary conditions are the most important ones for the simulation of a structure. They will define the behaviour of a structure. In case no boundary conditions are defined, RSA assumes the structure as perfectly fixed. Though supports are always necessary to define.

Supports

In this step the supports are defined by the *AnalyticalNode.SetSupportByName* (with input port for multiple *nodes*, for faster processing).

The supports are set to analytical nodes. These last ones can be detected using the *AnalyticalBar.StartNode* or *AnalyticalBar.EndNode* nodes, depending on which side of the element needs support.

The support name is a string referring to the Support label available in RSA. Make sure the support label is available in RSA. This can be checked in RSA via the Supports dialog (Geometry menu > Supports). In case the support is not available, you can define one yourself using the button in the dialog. There you need to set the conditions of the support.

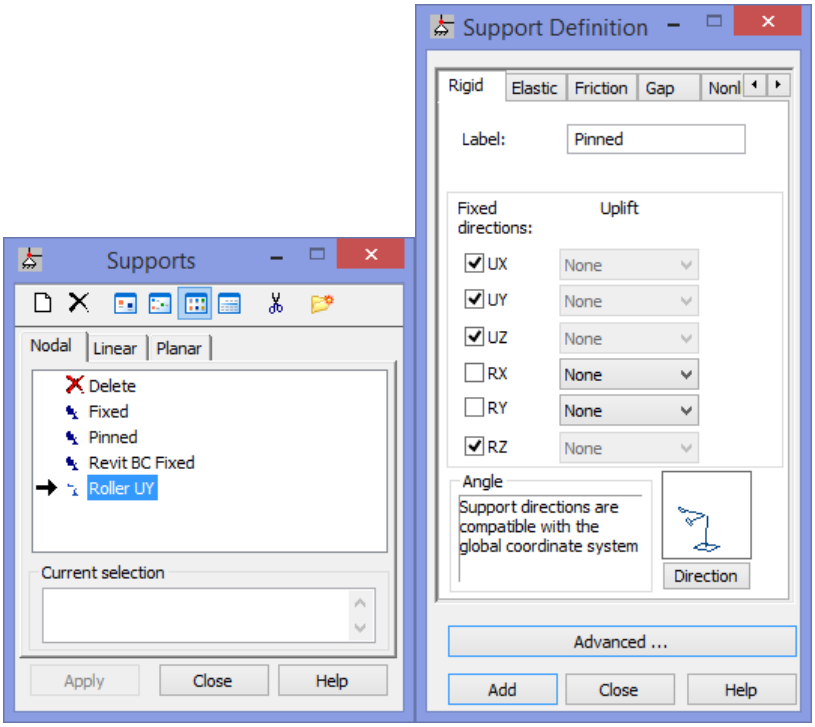


FIG. 25 - SUPPORT DEFINITION IN RSA

Releases

The bar end releases are optional, but in case of trusses they are indispensable. The same logic is used as for the supports, but you use the *AnalyticalBar.SetReleaseByName* instead.

The input for this node are *AnalyticalBars*. The direction of an analytical bar is very important for the definition of releases. In case of an asymmetric release (i.e. fixed at the start, pinned at the end), you need to make sure you have used the right order in Dynamo to create the geometry lines. The way how they are created will define the local x-direction of the analytical bar (i.e. the choice of start and endpoints with the *LineByStartPointEndPoint* node)

The release name is a string referring to the Release label available in RSA. Make sure the label is available in RSA. This can be checked in RSA via the Releases dialog (Geometry menu > Releases). In case the label is not available, you can define one yourself using the 🗋 button in the dialog. There you need to set the conditions of the release.

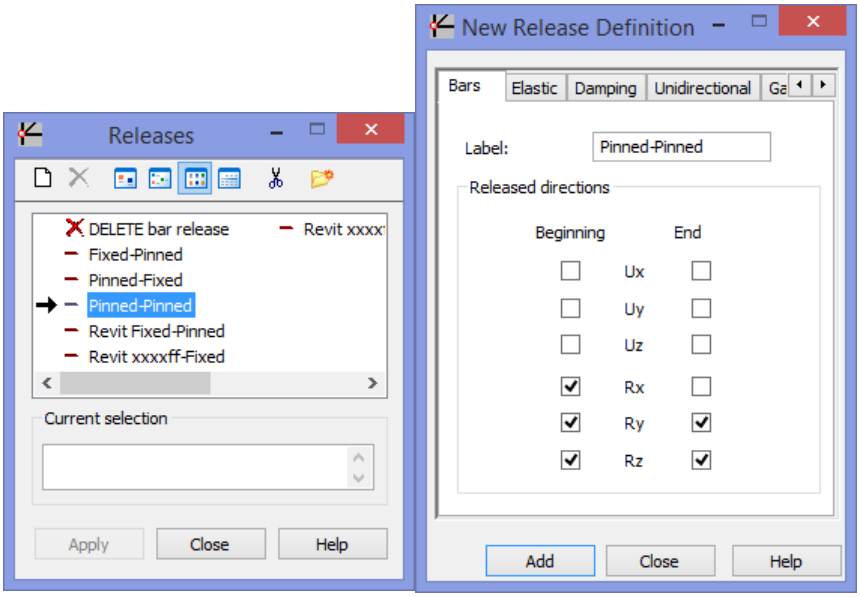


FIG. 26 - RELEASE DEFINITION IN RSA

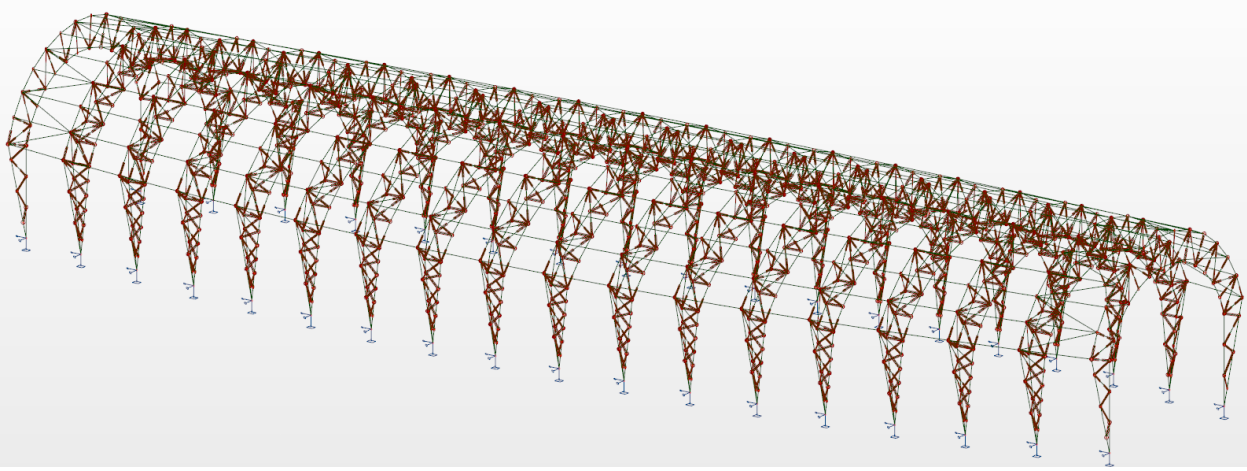
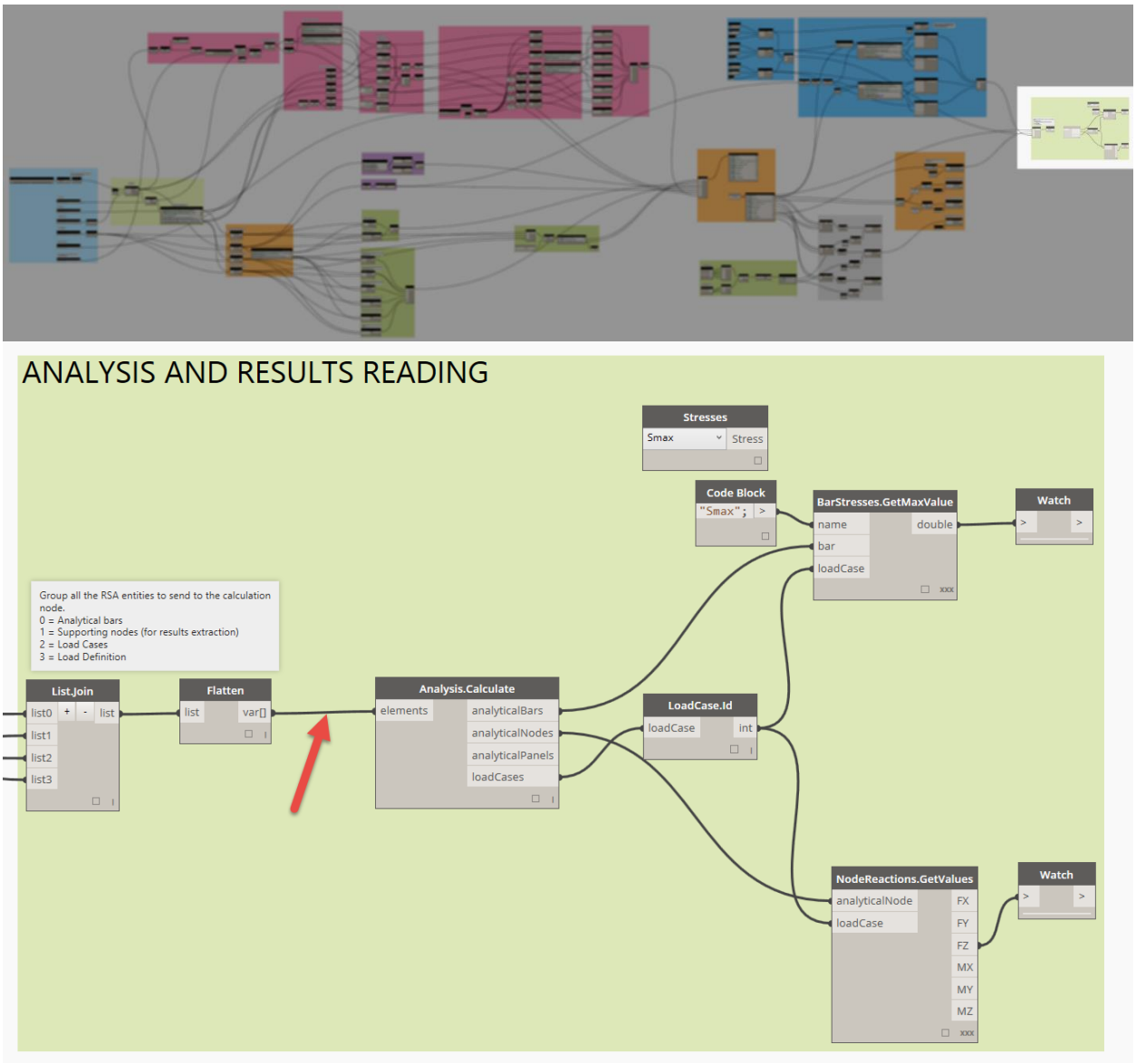


FIG. 27 - ANALYTICAL MODEL AFTER APPLYING SUPPORTS AND RELEASES

Step 15 – Analysis and results reading



This last step is the most important one, and also the one that needs a bit of your attention before you go to work with it.

The *Analysis.Calculate* node is the one that starts the simulation in RSA. It's advised only to connect the node, when you simulation model is set up completely (geometry, sections, loads and boundary conditions).

Analysis

Before you connect, it is important to first create a flattened list which contains the next elements:

- All **analytical bars (and panels)** that are generated
- The **analytical nodes with supports** applied (take the output from the *AnalyticalNode.SetSupportByName* node)
- All defined **load cases**
- All **load definitions** (except for the *LoadCase.ModelSelfWeight* node, only superimposed loads)

Analytical objects that are not part of the input list will be removed from the calculation model.

Now run the script and see how RSA calculates the structure.

The output of the *Analysis.Calculate* node are 4 separated lists with elements classified by category (bars, nodes, panels and load cases).

Results

At the outputs of this node you can connect the Result nodes.
Depending on which type of results you want to request, different inputs are needed.
But all the result nodes have one thing in common, which is the *loadCase* input. This needs to be an integer representing the *LoadCase.ID* of a defined load case.

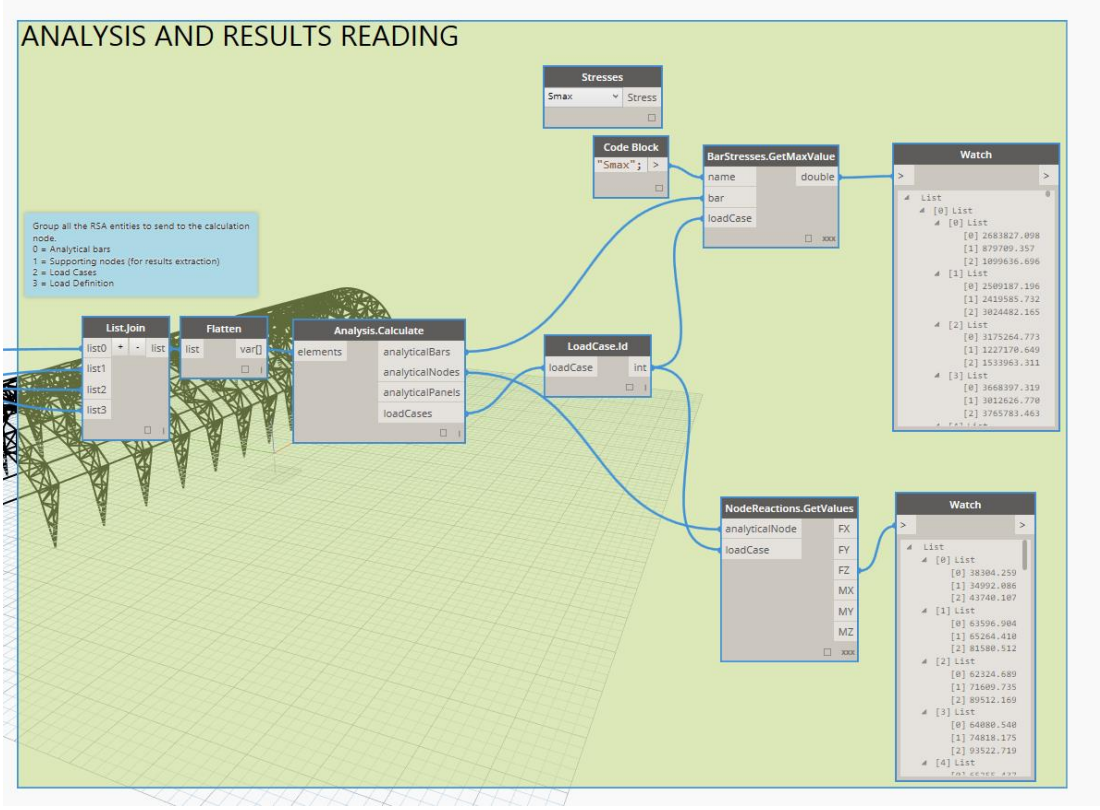
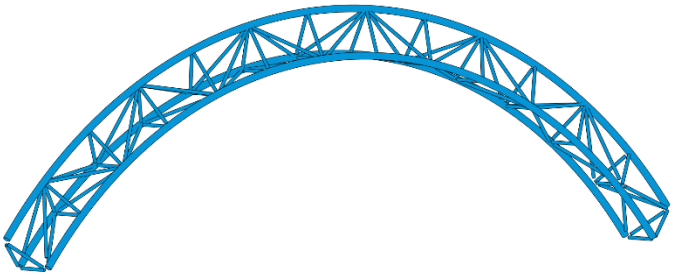


FIG. 28 - RESULTS IN DYNAMO AFTER RSA ANALYSIS

Case 3 – Manual design exploration of a single complex truss

In this third case we will take a single spatial truss from the complex analysis model of the *Armadillo Roof* in Robot Structural Analysis, using a parametrized approach with Dynamo, based on the initial design of the architect done in Revit. The goal is to find a better result for the truss using a manual method by manually changing the parameters and saving each analysis model. The conditions for having a "better result" is a lightweight structure that has stress ratios within the allowable range.

The base geometry of the roof is again the SAT export of the Mass object initially created in Revit by the architectural designer.



DATASETS

RSA



Start model:
Armadillo Platform - Single Truss Analysis_start
Finalized model:
TH 2.600000_TW 1.500000_CHS 219.1x8_CHS 88.9x6.3_12420.rtd

Dynamo

Workspace:
Armadillo Spatial Truss - 01 Analysis.dyn
Armadillo Spatial Truss - 02 Manual Parametrizing.dyn

Custom nodes:
V bracings.dyf
Chained line.dyf
Armadillo Spatial Truss Analysis.dyf

External input
Platform Roof.sat

To open the dataset files:

1. Launch the sandbox version of Dynamo 0.9 (C:\Program Files\Dynamo 0.9\DynamoSandbox.exe) or use Dynamo Studio 2016 (v. 0.9.3072)
2. Open the Dynamo script (.dyn). In the next chapters it will be specified which file to open.
3. Launch Robot Structural Analysis 2016
4. Open the RSA start model file. Without this step the script won't work!
5. Connect the wires as shown in the class and play with the parameters.

General overview of the script

This Dynamo script will generate a single spatial truss, without the lattice columns. The script that has been used in Case 2 can be reused, by deleting some of the steps and adding a new step for results evaluation. The final result of the script is shown below.

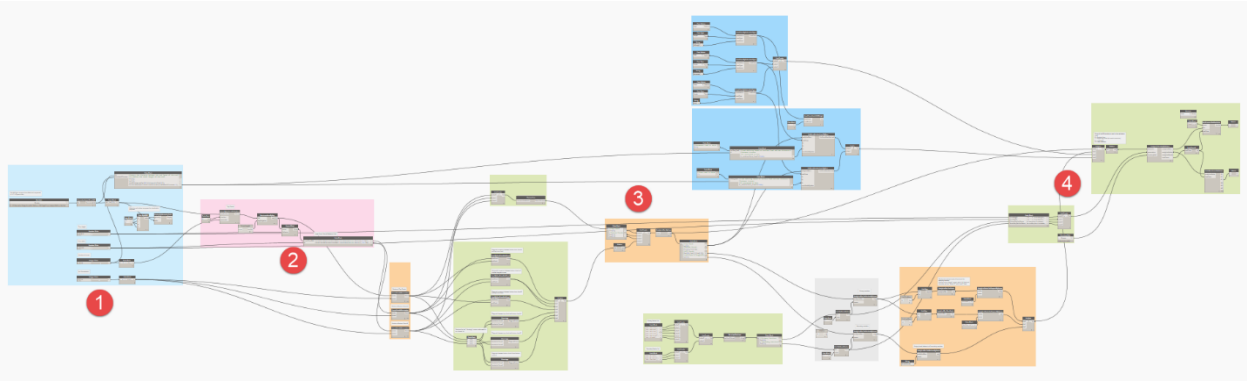


FIG. 29 - DYNAMO SCRIPT SINGLE TRUSS ANALYTICAL MODEL – OVERVIEW

For this case, two scripts are used:

Dynamo script: Armadillo Spatial Truss - 01 Analysis.dyn

Step 1 = input parameters, such as the basis roof surface, truss geometry …

Step 2 = definition of the main truss chords

Step 3 = setting up the simulation model in Dynamo and RSA

Step 4 = analysis of the structure

Dynamo script: Armadillo Spatial Truss - 02 Manual Parametrizing.dyn

In a later phase we are going to take this script and group it in custom node to allow a list input.

Step 5 = preparing the script for list input handling

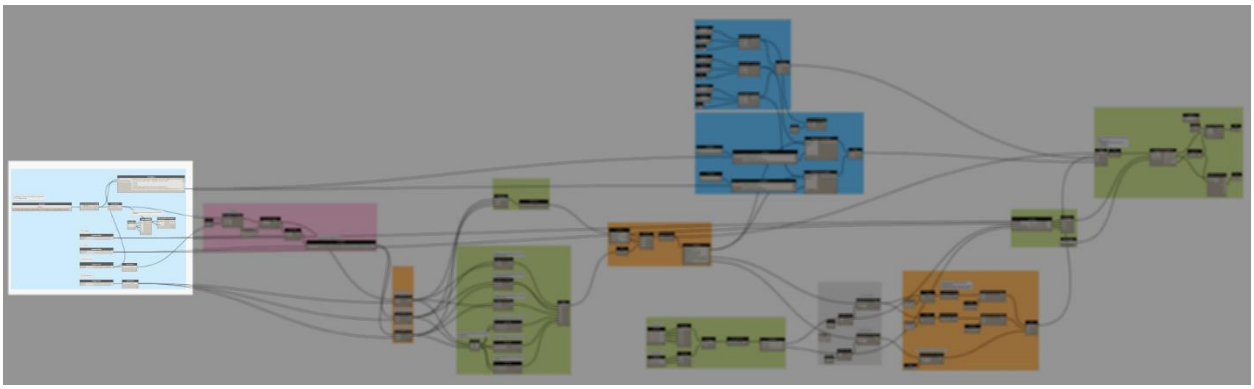
Step 6 = evaluation of results

Each of the steps is described in the chapters below.

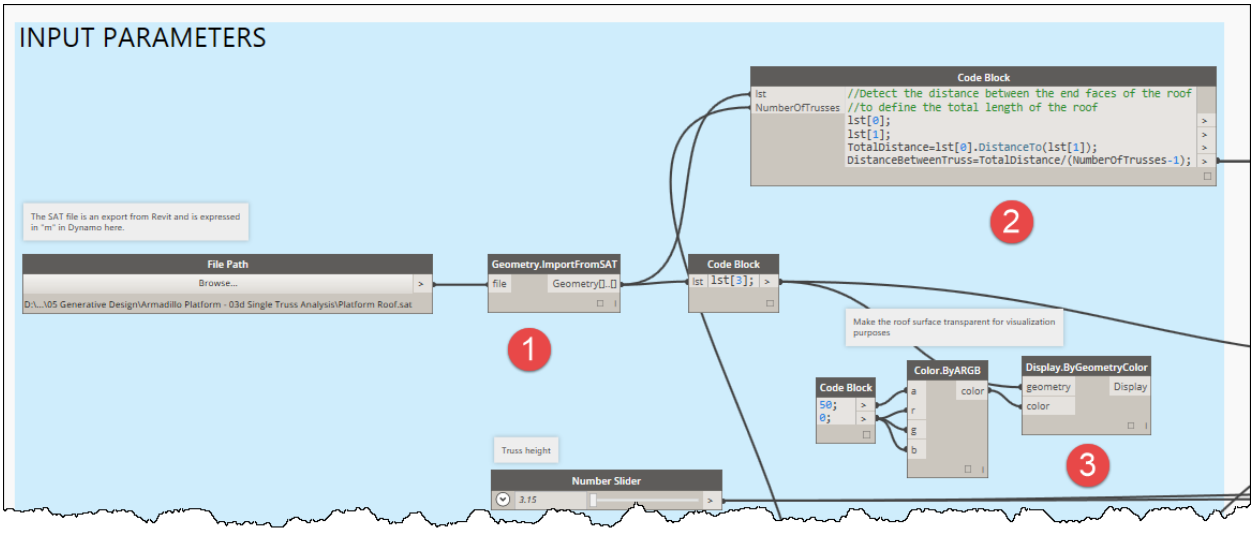
The nodes for Structural Analysis connection can be found in the Analyze > Structural Analysis category in the Dynamo nodes library.

Before you run the script, please read Important notice – Setting in Robot Structural Analysis

Step 1 – Input parameters



Most of the input parameters are already described in Case 1 and Case 2. The only difference here is that we only want one spatial truss for this design. As the roof has a varying width along its length axis, we have to detect the truss with the largest span, which will be done in Step 2 – Definition main truss chords. For that we need to know the total length of the roof.

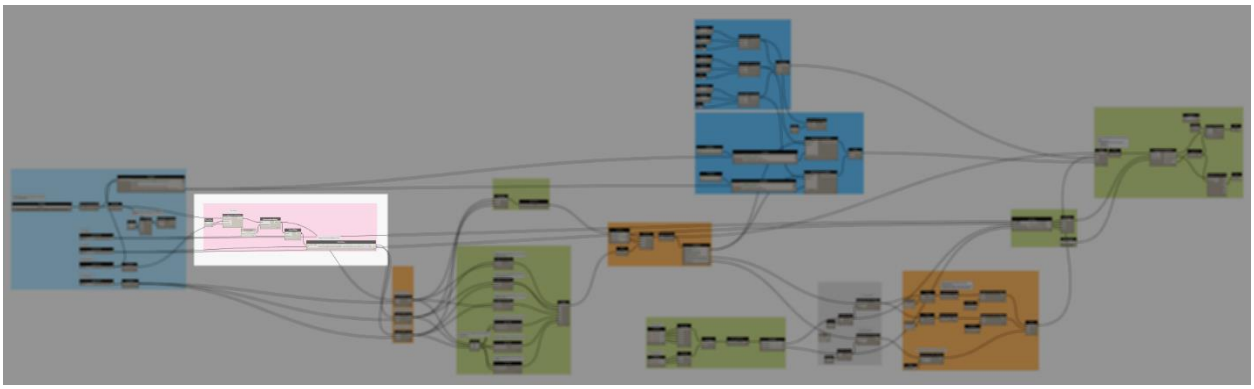


(1) Read the SAT geometry with the *Geometry.ImportFromSAT* node. Make sure you point to the right path for the "Platform Roof.sat" source file, by clicking the *Browse* button on the *File Path* node.

(2) The top roof surface is still indexed as [3]. The vertical front and back faces of the SAT geometry are indexed as [0] and [1]. With the *Code Block* node we detect the shortest distance between those two surfaces and divide it by the "NumberOfTrusses – 1" to get the center distance between each truss.

(3) Optionally you can add some transparency to the roof surface, to see through it when you generate the Dynamo geometry.

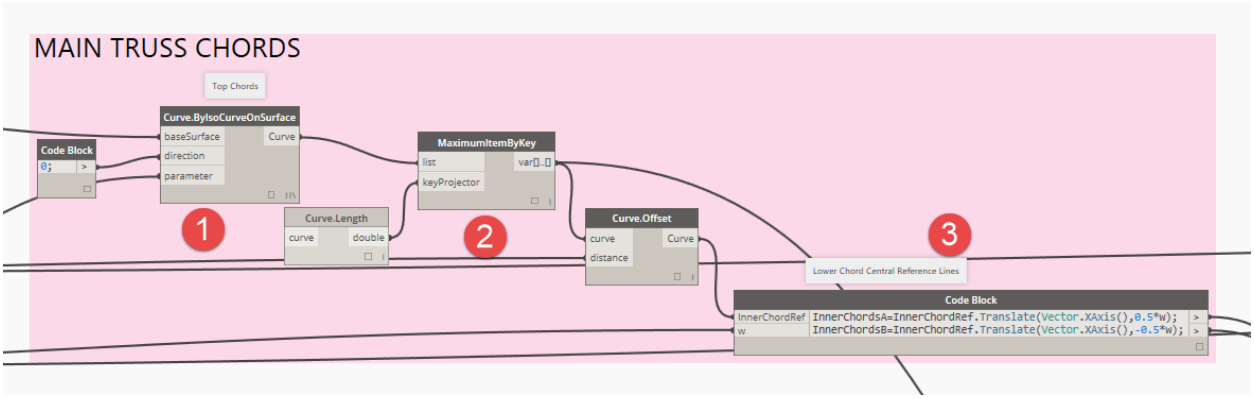
Step 2 – Definition main truss chords



(1) Once the roof surface has been identified, it's time to set up the main truss chords again using the *Curve.ByIsoCurveOnSurface* node (see also Step 2 – Outline main truss chords)

(2) As we only want to have one single spatial truss we first need to detect which one of the number of trusses (18 in this case) is the determining one (the one with the largest span). Therefore the *MaximumItemByKey* can help us to find the element which is maximum in its *keyProjector*. The key for this is the *Curve.Length*. So this step will check the length of each top chord curve and will return the curve with the maximal length.

(3) On this result we apply an offset and a translation to get the bottom chords.



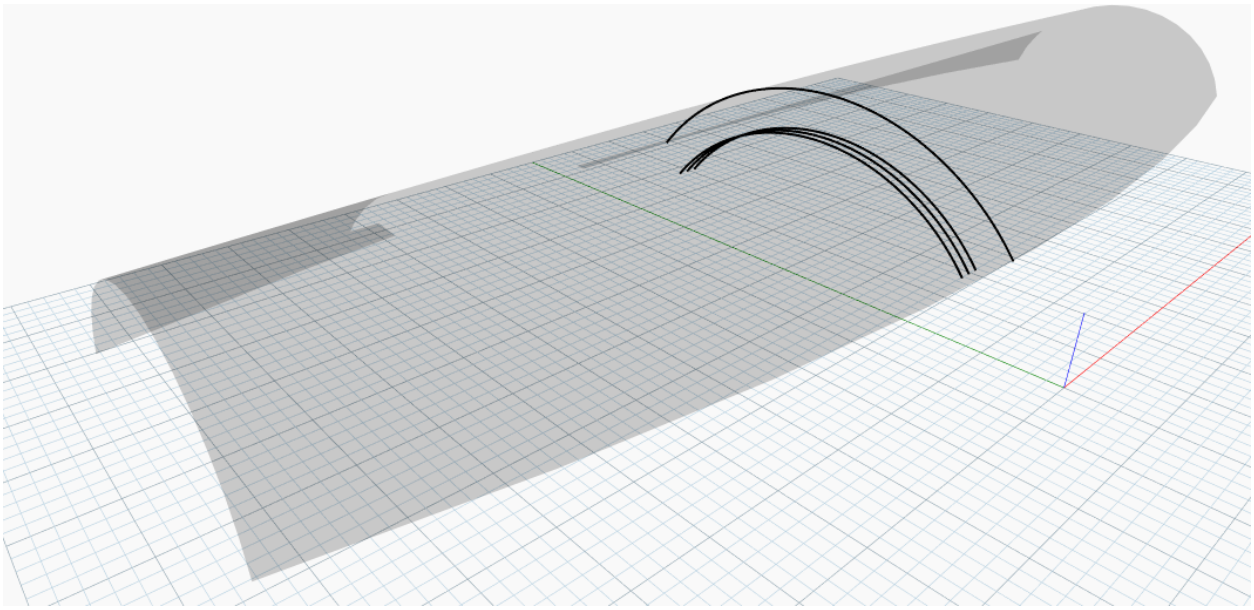
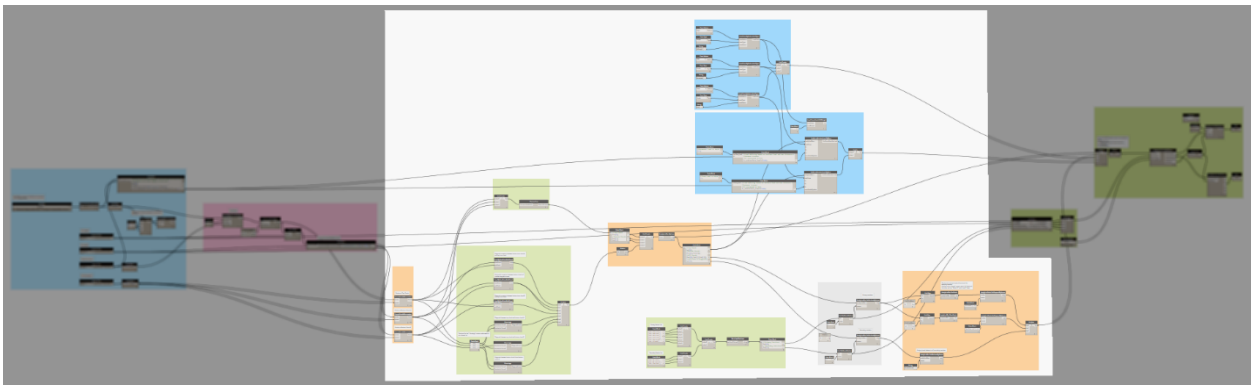


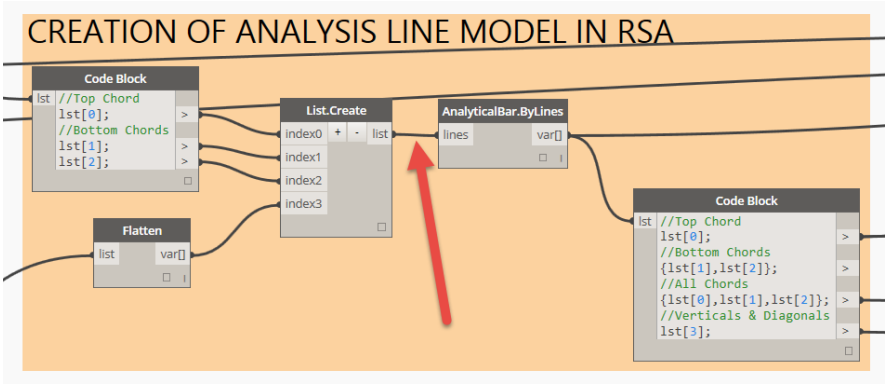
FIG. 30 – MAIN CHORD OUTLINES FOR A SINGLE TRUSS IN DYNAMO

Step 3 – Generation of the full analysis model



In this step the nodes from Case 2, Step 2 till 14 are retaken (except for the ones for the column lattices). This way a single truss outline is generated in Dynamo and at the same time also the analysis model in RSA.

To get the model in RSA, don't forget to connect the *List.Create* node with the Dynamo geometry to the *AnalyticalBar.ByLines* node.
To improve the performance of the script, it is sometimes advisable to disconnect these two first, to see the result in Dynamo first before you send it to RSA.



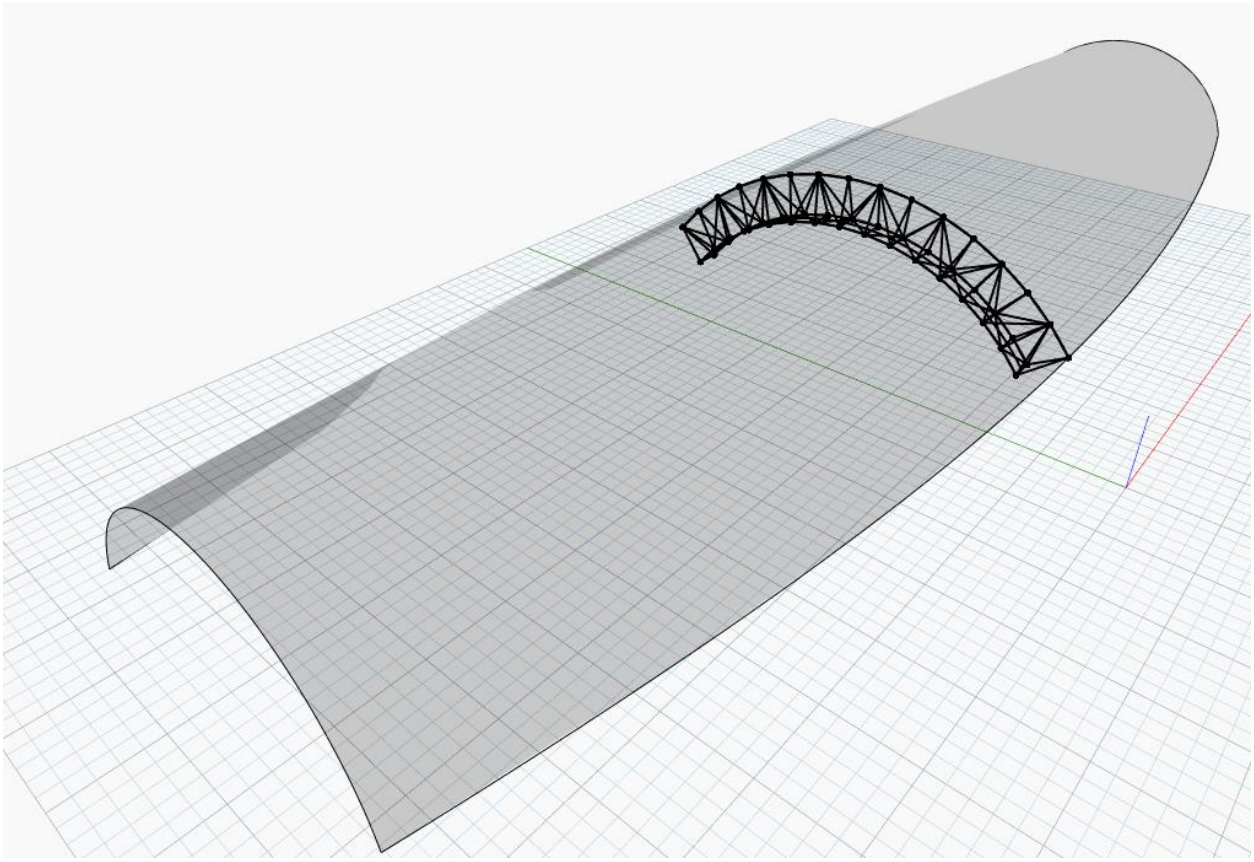
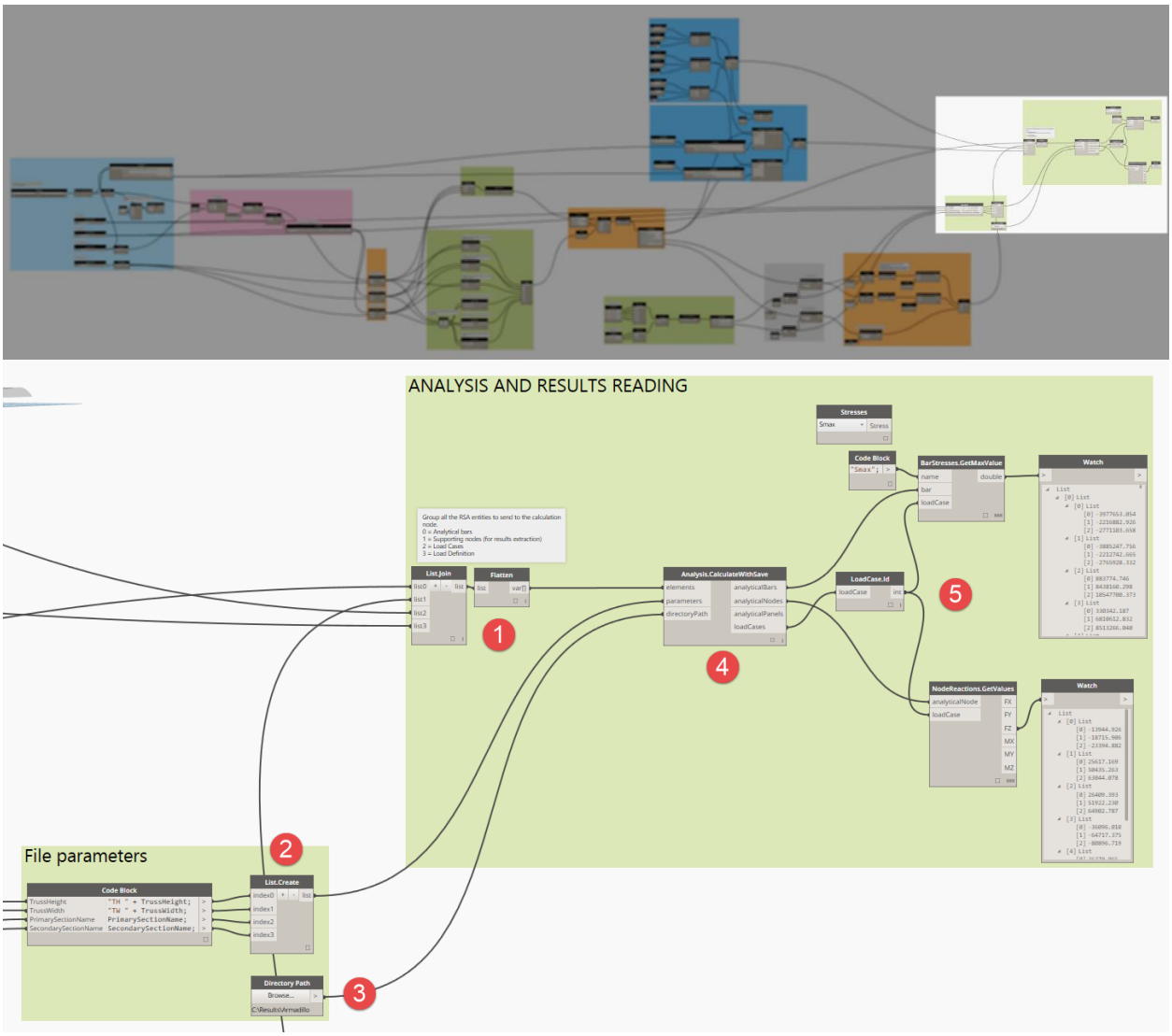


FIG. 31 – FULL ANALYSIS MODEL OF A SINGLE TRUSS IN DYNAMO

Step 4 – Analysis of the model



In this case we want to explore multiple design options by changing the input parameters manually. Additionally we want to review each model afterwards in RSA.

(1) As mentioned before it is important to connect all RSA entities in a flattened list with the analysis node.

(2) To create separate files for each design option, a list of parameter values is required. This identifies the saved models easier. In this case we make a list of the input parameters (truss height, truss width, primary section index, and secondary section index). This will result in a file name with a "_" between each list item and a time stamp at the end.
Example: "TH 2.6_TW 1.5_CHS 219.1x8_CHS 88.9x6.3_12420.rtd" where "12420" is the timestamp.

(3) Besides a file name, we also need to set a folder where to store the models in with the *Directory Path* node.

(4) Next the flattened list of RSA entities, the parameter list and the folder to save in, are connected with the *Analysis.CalculateAndSave* node. This node gives the ability to calculate and save each different model with the indicated parameters to the indicated folder.

(5) Finally, the results can be read in a similar way as explained before.

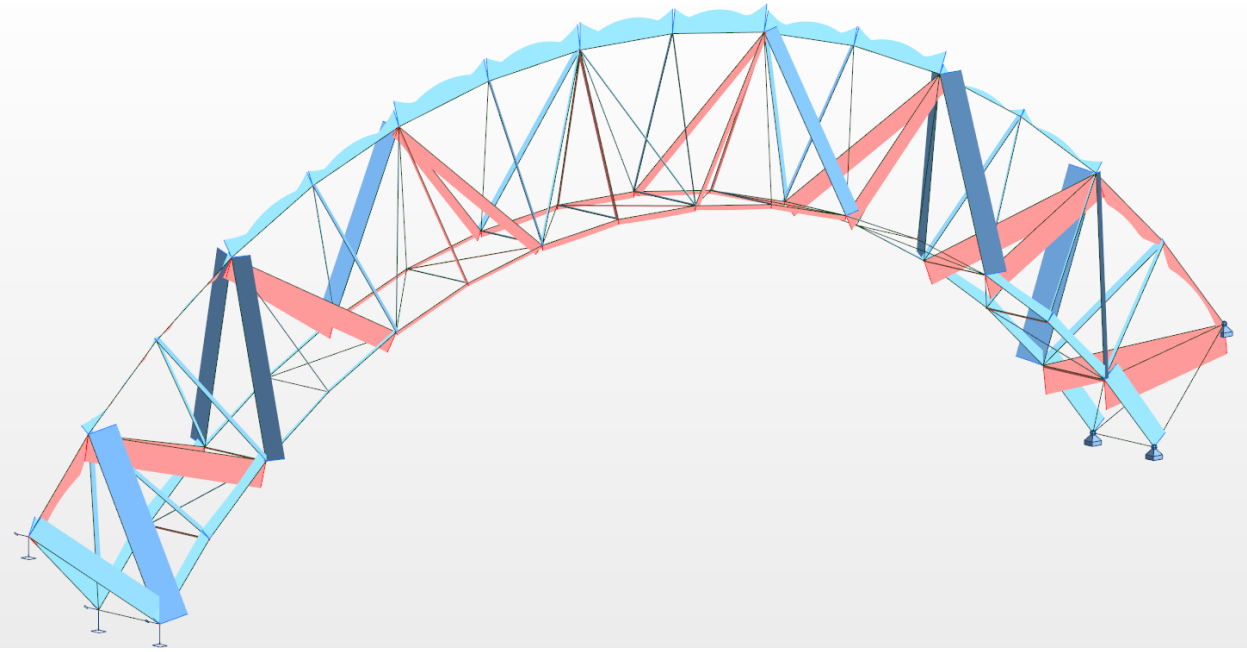
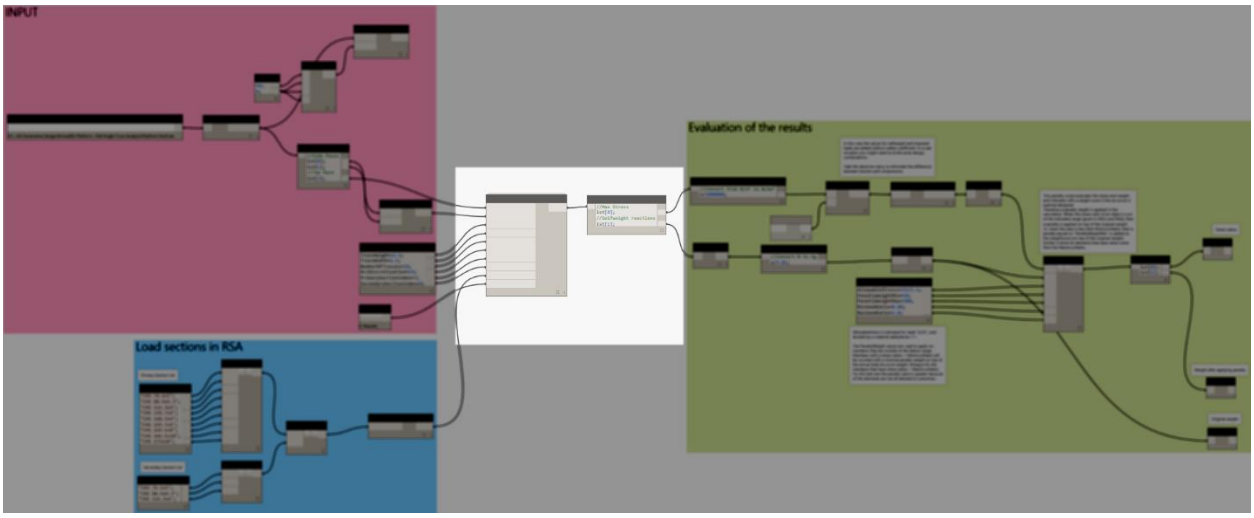


FIG. 32 – RESULTS DISPLAY IN RSA OF A SINGLE SPATIAL TRUSS

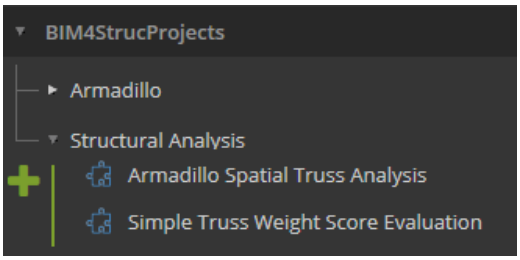
Step 5 – Preparing the script for list input

For this step the next Dynamo script needs to be opened:

Armadillo Spatial Truss - 02 Manual Parametrizing.dyn



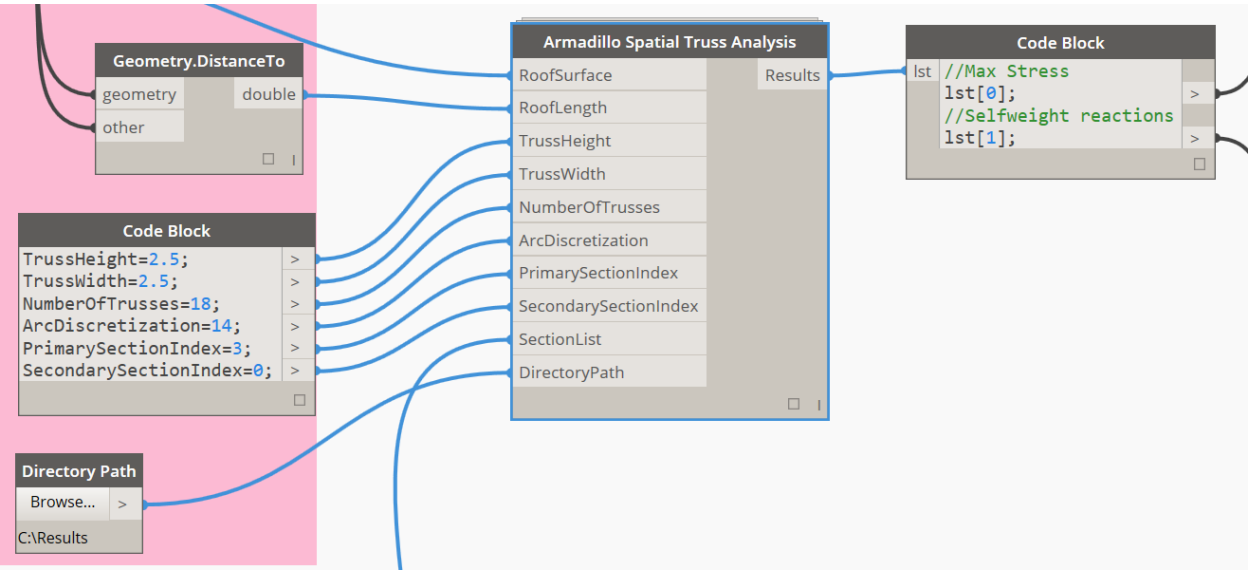
In this script, all the nodes used for generating the Dynamo geometry and the RSA analysis model and calculation are grouped in a custom node (.dyf), called **Armadillo Spatial Truss Analysis.** This custom node will be visible in the node library, in the BIM4StrucProjects > Structural Analysis category



The *Input* and *Load sections in RSA* group are isolated from this custom node for better control on the script. For more info about the nodes in these groups, please review Step 1 – Input parameters and Step 12 – Load sections in RSA from Case 2.

The custom node will accept input for a list of values in order to automate the design options exploration further on in Case 4. The input ports are explained below.

Spatial Truss Analysis Input



The input ports consist of:

- **RoofSurface** : the surface of the roof, as explained in Step 1.
- **RoofLength** : the length of the roof, based on the distance between the front and end face.
- **TrussHeight:** the height of the truss to consider in the design option
- **TrussWidth** : the width of the truss to consider in the design option
- **NumberOfTrusses** : this is important to define the position of the truss (variable length of the truss) and the center distance between the trusses in a full model (important for load defining)
- **ArcDiscretization** : necessary for the discretization of the arcs in RSA
- **PrimarySectionIndex** : the index to the item in the primary section list (for chords)
- **SecondarySectionIndex** : the index to the item in the secondary section list (for diagonals)
- **SectionList** : the full section list that is loaded into RSA at the beginning of the script
- **DirectoryPath** : the folder where to save the separate models to.

Each time you change one of the variables in the Code Block, a new analysis model will be created, analyzed and saved in the folder indicated with the *Directory Path* node.

Spatial Truss Analysis Output

The output consists of a list of results:

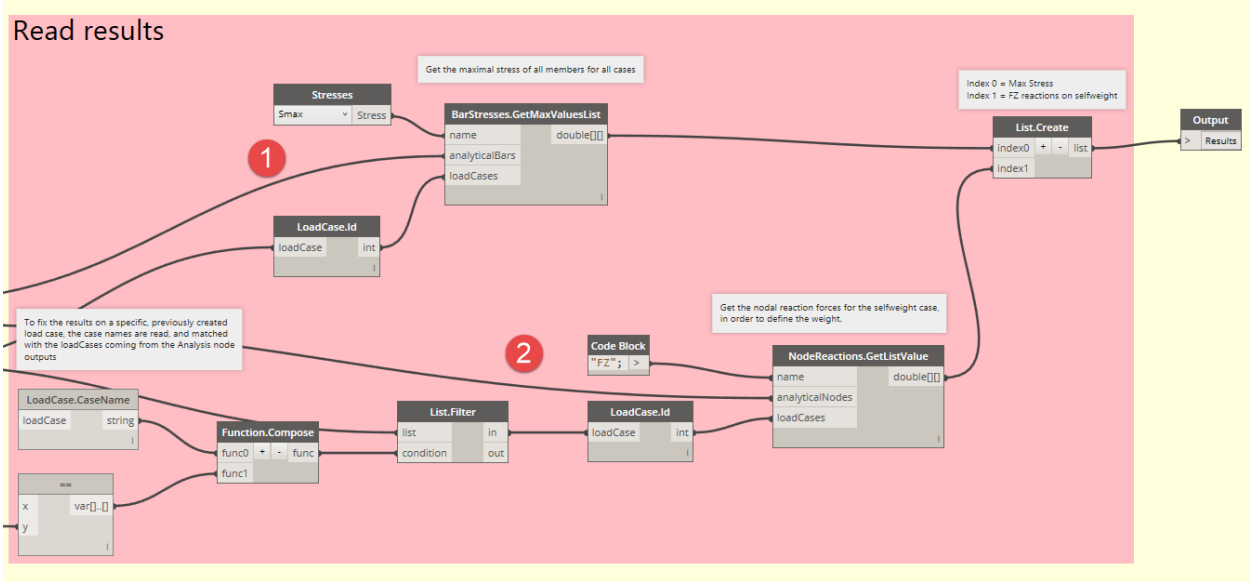


FIG. 33 - RESULTS READING IN THE ARMADILLO SPATIAL TRUSS ANALYSIS CUSTOM NODE

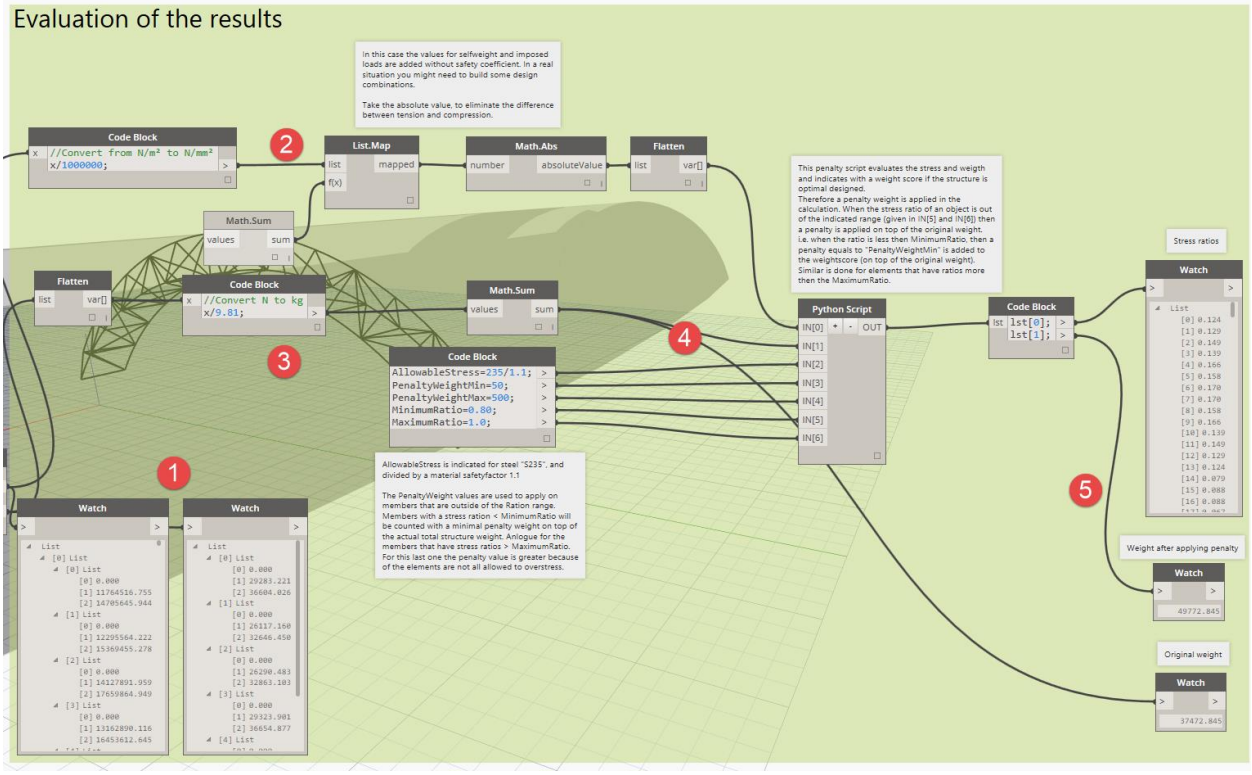
(1) Get the maximal values of Smax (which is the maximal Von Mises stress) in each bar with the *BarStress.GetMaxValuesList* node. The *analyticalBars* input is connected with the *analyticalBars* output from the *Analysis.CalculateAndSave* node. As input for *loadCases* we take the *LoadCase.ID* from each calculated load case. Later, these values will be added to each other to have a total bar result.

(2) A second result we want to have is the selfweight of the structure. This can be found by using the *NodeReactions.GetListValue* node and harvesting the results for *FZ* (which represents the vertical reaction force) for all *analyticalNodes* from the *Analysis.CalculateAndSave* node (these nodes are basically all the ones with a support assigned to). The *loadCases* needed is the one with the appropriate *LoadCase.Id* for the "Selfweight" case. A bit special here is that we need to match the *loadCases* output from the *Analysis.CalculateAndSave* node with the right load case name (string value). This because of the risk of recreation of load cases while running the script multiple times, and thus having new loadCase IDs. With the *List.Filter* node, we filter the *loadCases* output and find the case with a *LoadCase.CaseName* which is equal to the name of the selfweight case ("Selfweight" in this case). Then take the *LoadCase.ID* of this result.

Step 6 – Evaluation of the results

Back to the workspace of *Armadillo Spatial Truss - 02 Manual Parametrizing.dyn* we find a node group for the evaluation of the *Results* list from the custom node.

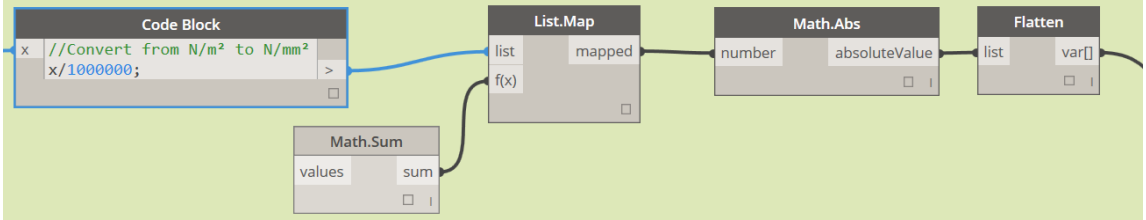
The goal of this part of the script is to evaluate the bar stresses "Smax" and apply "penalty weight values" for the bars that are outside a given stress ratio range. A stress ratio is the max stress / allowable stress ($R = S_{max} / S_{adm}$). The penalty consists of adding extra fictive weight to the actual structure weight. The closer the "Weight with penalty" approaches the "Actual weigh" the more optimal your design option is.



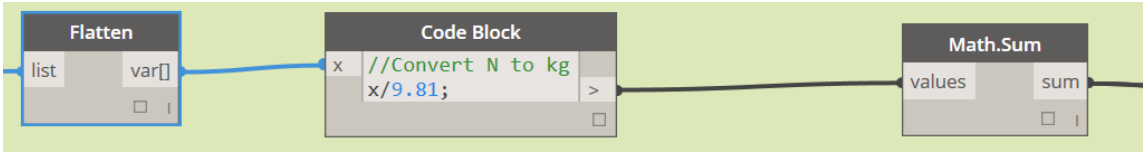
Results review and conversion

(1) Review the results coming from the Results list of the *Armadillo Spatial Truss Analysis* custom node with typical *Watch* nodes.

(2) The results for the stresses need to be
- converted from N/m² to N/mm²
- added together for each sub list ($S_{max} = S_{max,LC1} + S_{max,LC2} + \ldots + S_{max,LCi}$, with LC = Load Case). As the results are grouped per bar and returns results for each load case, we need to make the *Math.Sum* with a *List.Map* function.
- converted to a number without a sign, as we don't want to make a distinction between tensions and compression here.



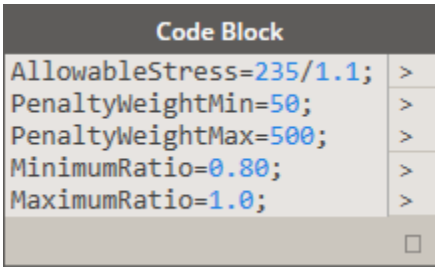
(3) The results for the weight need to be
- converted from N to kg
- added together with the Math.Sum on the flattened results list for all analytical nodes.



Calculation of penalty weight

(4) This part evaluates the resulting stress and weight and indicates with a weight score if the structure is optimal designed.

First we need to set constraints to the evaluation.



AllowableStress is indicated for steel "S235", and divided by a material safety factor γ_M = 1.1.

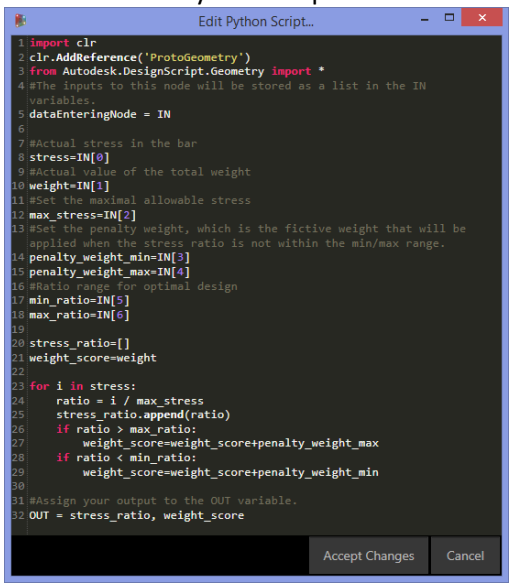
The **PenaltyWeight** values need to be applied on members that are outside of the Ratio range. Members with a stress ratio < **MinimumRatio** will be counted with a minimal penalty weight (PenaltyWeightMin) on top of the actual total structure weight. Analogue for the members that have stress ratios > **MaximumRatio**. For this last one the penalty value is greater because of the elements are not all allowed to overstress.

The formula is this $P = W + a*P_{min} + b*P_{max}$ with W = actual weight, a = number of members below the MinimumRatio, P_{min} is the minimal penalty, b = number of members above the MaximumRatio and P_{max} is the minimal penalty.

Example: In a structure with total weight "W" consists of 10 elements, and 7 of them have ratios smaller than the MinimumRatio. Then the PenaltyWeight will be calculated as $P = W + 7*P_{min}$

The penalty weight is calculated in the Python script.





The Python script needs a few inputs (IN):

- IN[0] = the converted stresses (in form of a list)
- IN[1] = the converted weights (in form of a list)
- IN[3] = minimal penalty weight (for members with ratio < min allowable ratio)
- IN[4] = minimal penalty weight (for members with ratio > max allowable ratio)
- IN[5] = minimal allowable stress ratio
- IN[6] = maximal allowable stress ratio

The output (OUT) of this node returns a list of results:

- Index [0] = the actual stress ratio for each bar
- Index [1] = the weight score (which represents actual weight raised with the penalty weight).

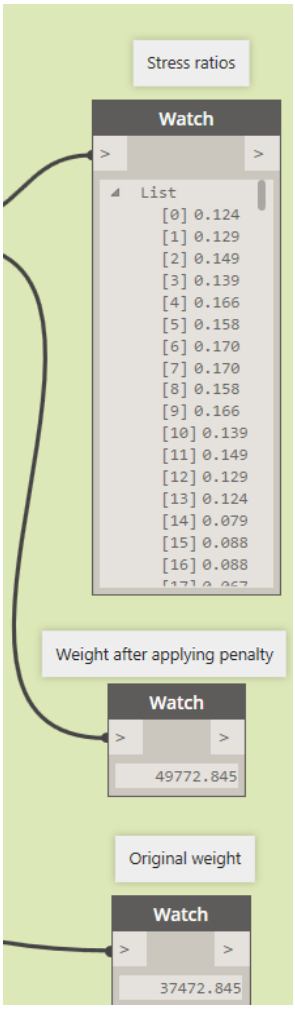


FIG. 34 - RESULTS AFTER PENALTY WEIGHT CALCULATION

Case 4 – Parametric Run Optimization of a single complex truss

Though in the previous case we automatically get one model for each design option, it still needs manual interaction for the input of the variables.

In this forth case we will automate this process by feeding the custom analysis script for a single spatial truss with a combined list of variables for the geometry (height, width…) and feed these values to the *Armadillo Spatial Truss Analysis* custom node that has been made previously. The results of all these calculations are then exported to Excel. This will help us in making a choice of the right truss configuration from an extensive list of design options. To help making that choice, some constraints are built in the script. These constraints will result in a weight score. The solutions with the lowest weight score will be the best ones.

The base geometry of the roof is again the SAT export of the Mass object initially created in Revit by the architectural designer.

DATASETS

RSA



Start model:
Armadillo Platform - Single Truss Analysis_start
Finalized models:
Multiple files stored in the 'Parametric Run' subfolder

Dynamo

Workspace:
Armadillo Spatial Truss - 03 Parametric Run.dyn

Custom nodes:
V bracings.dyf
Chained line.dyf
Armadillo Spatial Truss Analysis.dyf
Simple Truss Weight Score Evaluation.dyf

External input
Platform Roof.sat

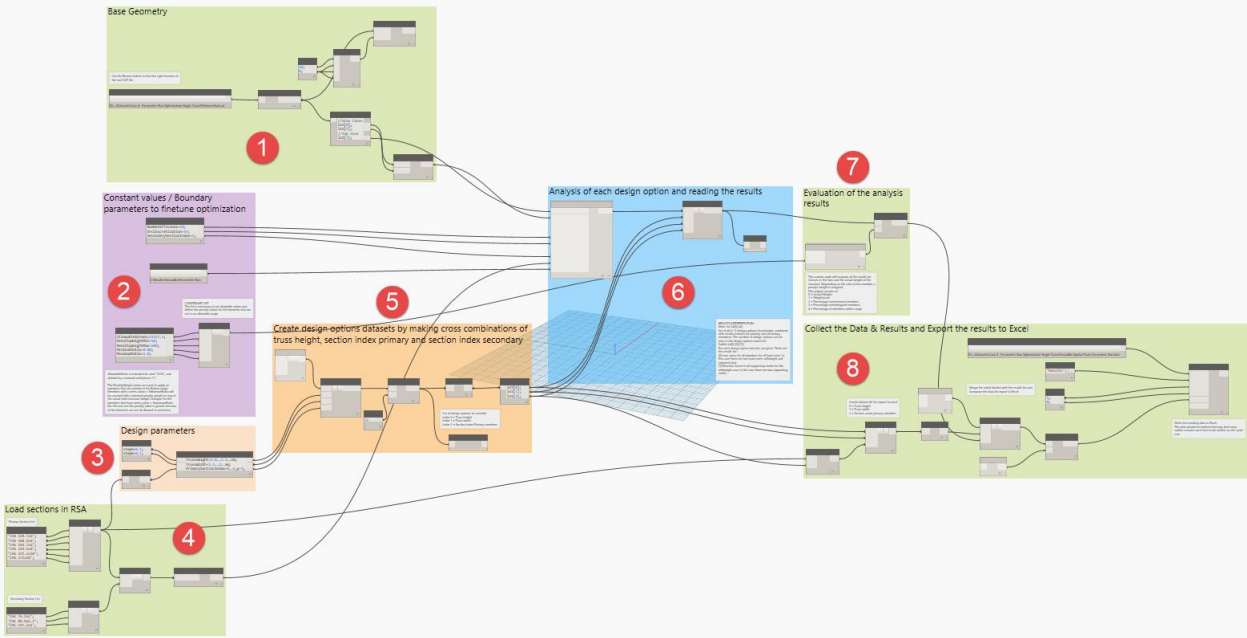
External output
Armadillo Spatial Truss Parametric Run.xlsx

To open the dataset files:

1. Launch the sandbox version of Dynamo 0.9 (C:\Program Files\Dynamo 0.9\DynamoSandbox.exe) or use Dynamo Studio 2016 (v. 0.9.3072)
2. Open the Dynamo script (.dyn). In the next chapters it will be specified which file to open.
3. Launch Robot Structural Analysis 2016
4. Open the RSA start model file. Without this step the script won't work!
5. Connect the wires as shown in the class and play with the parameters.

General overview of the script

The script allows us to analyze multiple designs that are generated within a given range of parameter values. Most of the parts that were used in Case 2 and Case 3 are reused, with some additions to it.



The script consist of 8 main steps:

Step 1 = Input of the base roof geometry

Step 2 = Constant input values and constraints

Step 3 = Design parameters

Step 4 = Load sections to RSA

Step 5 = Generation of design options

Step 6 = Analysis of the design options

Step 7 = Evaluation of the analysis results

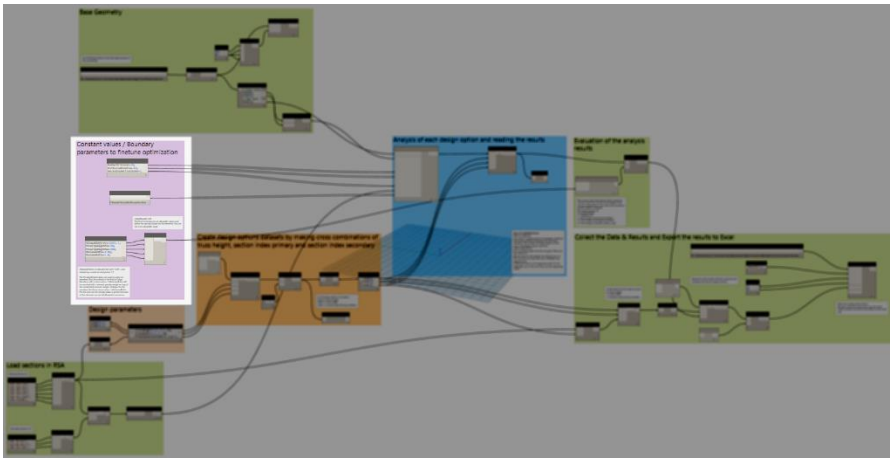
Step 8 = Export of evaluation results to Excel

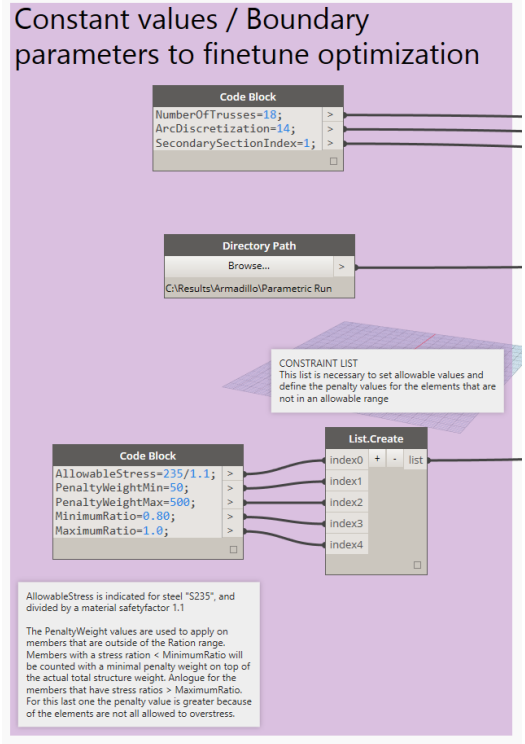
Before you run the script, please read Important notice – Setting in Robot Structural Analysis

Step 1 – Input of the base roof geometry

This is already explained in Case 3 - Step 1 – Input parameters.

Step 2 - Constant input values and constraints

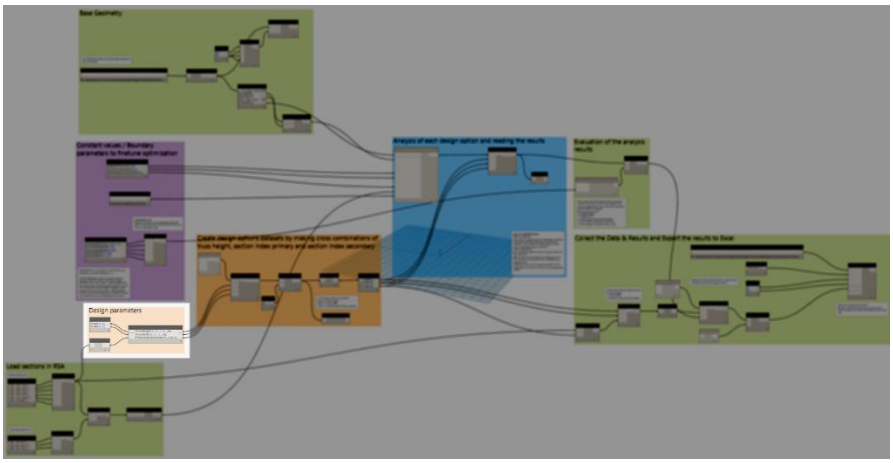


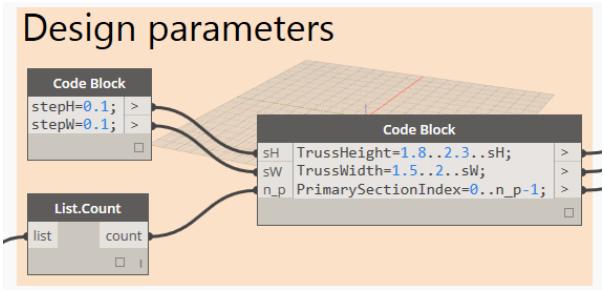


The goal of this part of the script is to define the constant values of the truss design, like the number of trusses, arc discretization and the sections for the secondary truss elements. We won't optimize these values. Though you could change this later and involve them in your design options list as well.

In the bottom part, the constraints for the results evaluation are listed. These constraints are built in to return a weight score for the specific solution. The solution with lowest weight score might be the one you're looking for. More about the constraints can be read on Calculation of penalty weight.

Step 3 - Design parameters





The design parameters represent a list of values that need to be considered for the generation of the several design options.

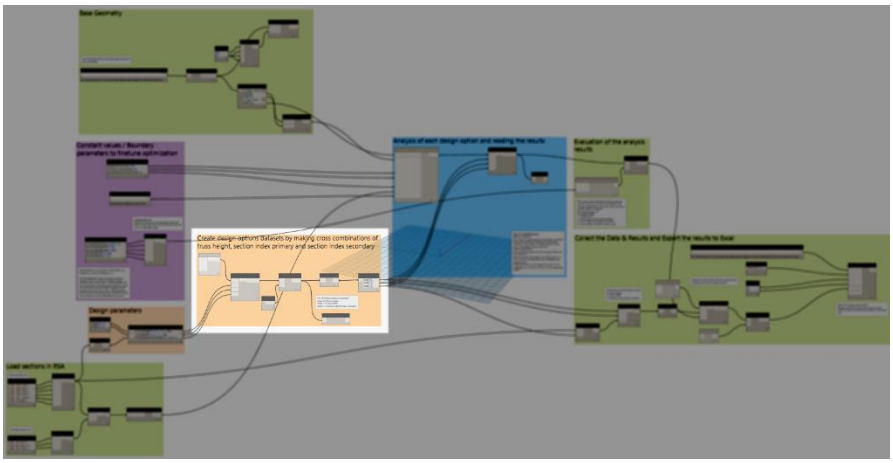
For example: a truss height varying from 1.8 m to 2.3 m with a step of 0.1 m (= 1.8, 1.9, 2.0, …, 2.3)

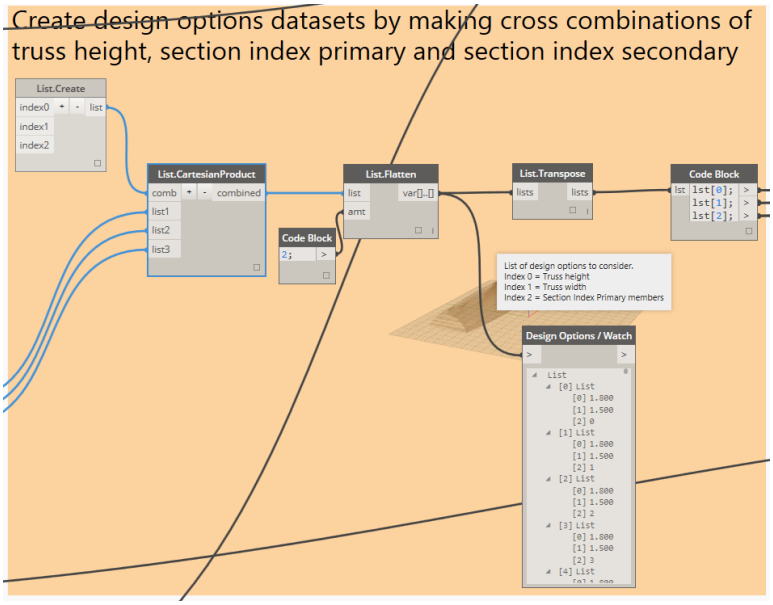
These lists will be combined to design options in Step 5 - Generation of design options

Step 4 - Load sections to RSA

This is already explained in Case 2 - Step 12 – Load sections in RSA

Step 5 - Generation of design options

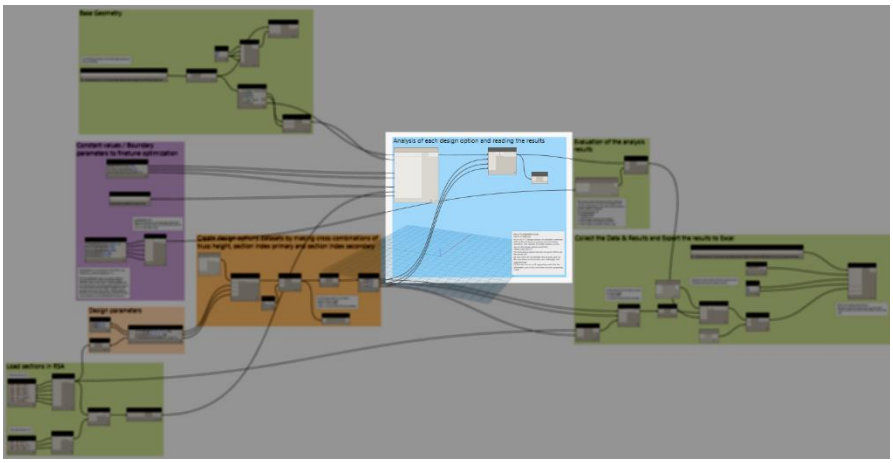


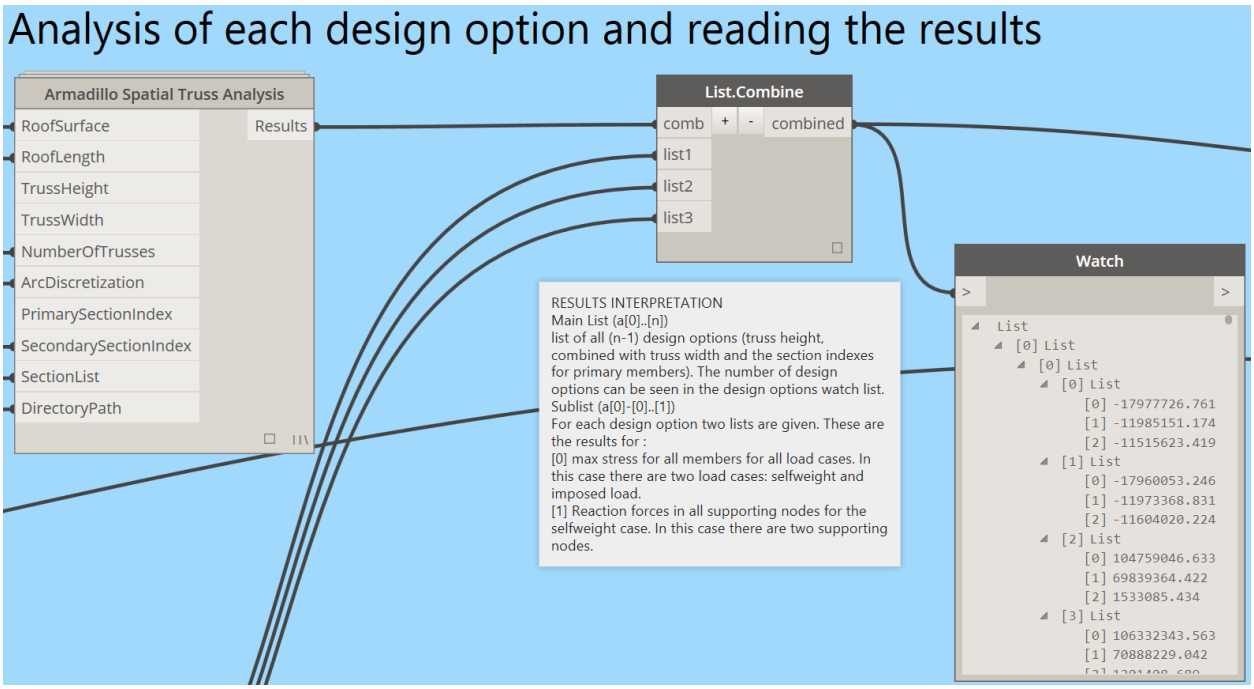


The list of parameter values that are defined in Step 3 - Design parameters, now need to be combined. We need combinations that are crossed along the 3 different design lists.
For example the TrussHeight = 1.8 m needs to be combined with TrussWidth = 1.5, 1.6, 1.7, … and this each time for SectionIndex 0, 1, … This means we need sub lists like {1.8, 1.5, 0}, {1.8, 1.5, 1}, …, {1.8, 1.6, 0}, … {2.0, 1.6, 0}, {2.0, 1.6, 1}, …

This can be achieved with the *List.CartesianProduct* node. The combinatory for this node is a *List.Create* node with 3 indices (we need lists of the design option parameters), and the 3 design parameter lists from Step 3.
In the *Design Options/Watch* node you can see the result of this operation after an array reduction. Finally the values need to be transposed in order to feed them to the *Armadillo Spatial Truss Analysis* custom node.

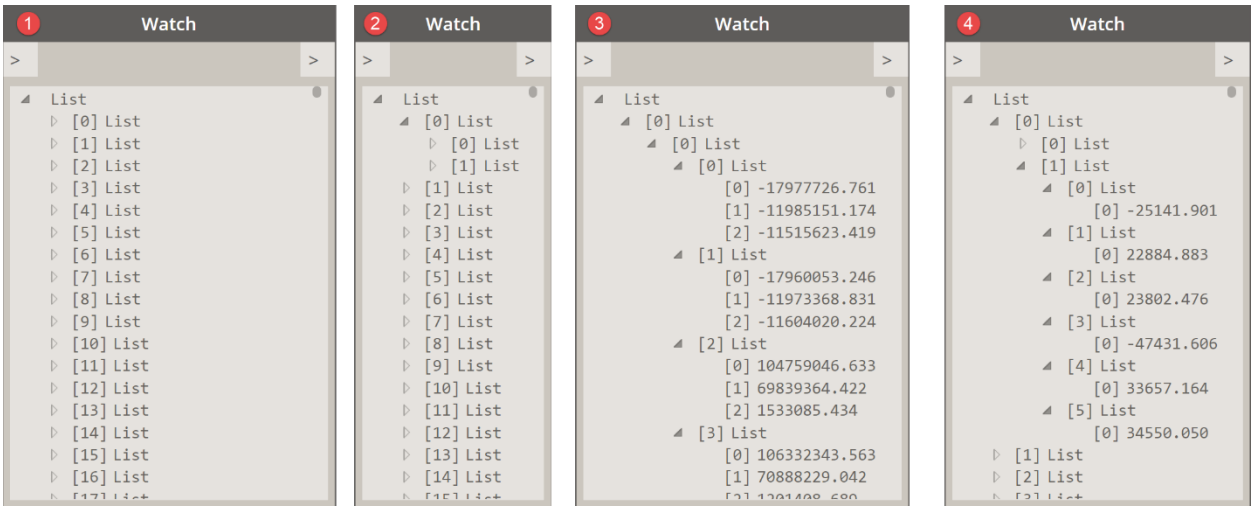
Step 6 - Analysis of the design options





When the constants and variables are defined, they can be connected with the *Armadillo Spatial Truss Analysis* custom node. The constant values are connected directly to the node. The variable lists cannot be sent to this node directly, as they are combined lists, and need to be connected through the *List.Combine* node. This node will make sure that the transposed list outputs from step 5 are combined properly in the custom node. You can compare this with a batch routine of the analysis model generation and analysis.

When the script is finished running, the *Watch* node will display a nested list with all the requested results for each design option.

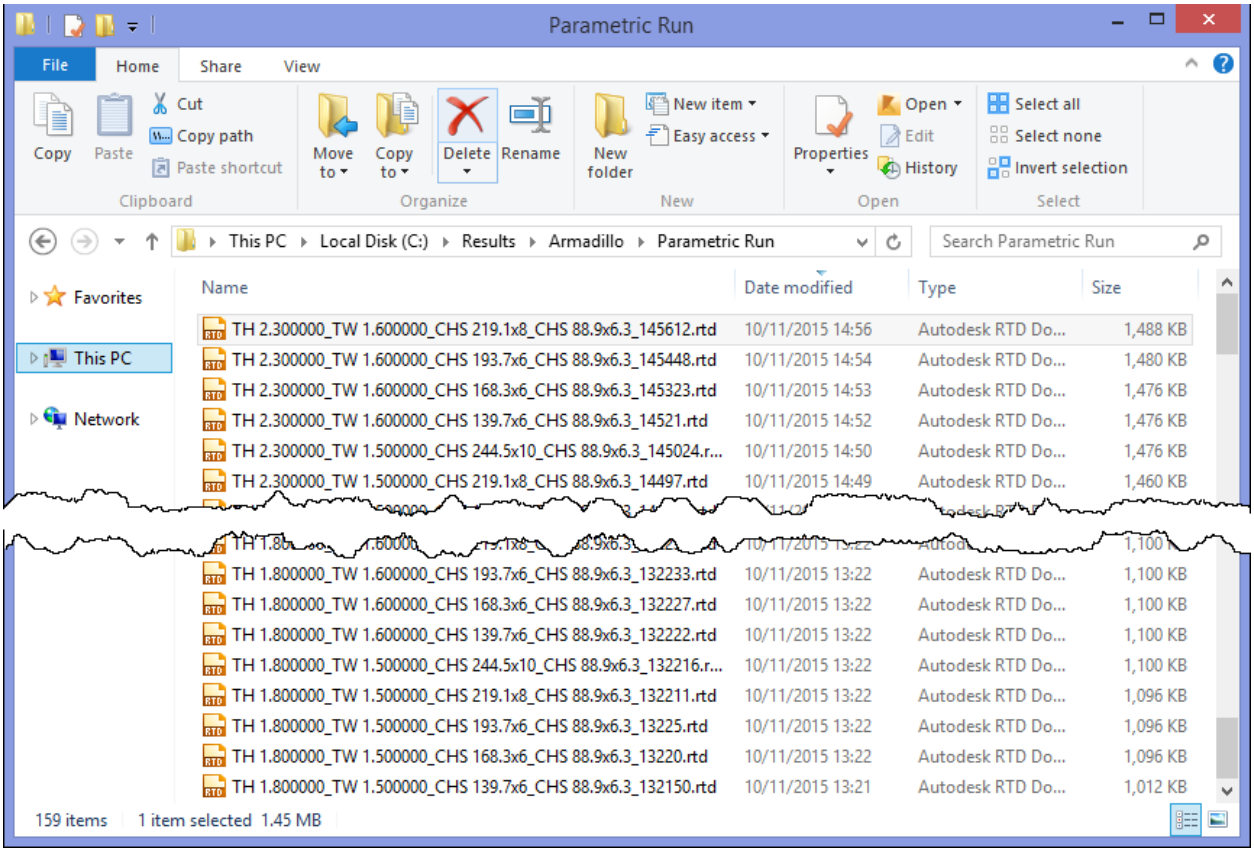


(1) The main branch of the list represents a group of results for each design option that was generated in Step 5 - Generation of design options.
Syntax
a[0]...a[i], with i = number of design options.

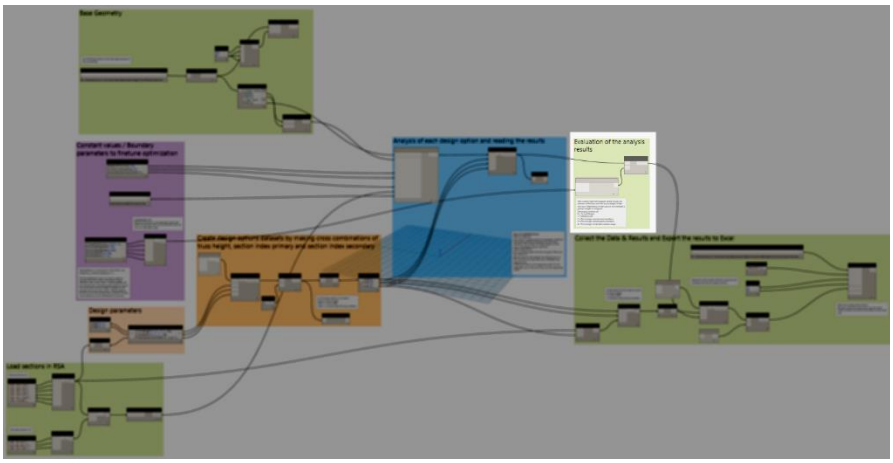
(2) In each element of the main list two items are stored: results for maximum stress and selfweight reaction forces.
Syntax
a[0][0], a[0][1]...a[i-1][0], a[i-1][1], with i=number of design options
a[0][0]…a[i-1][0] = results for maximum stresses
a[0][1]…a[i-1][1] = results for reaction forces for selfweight

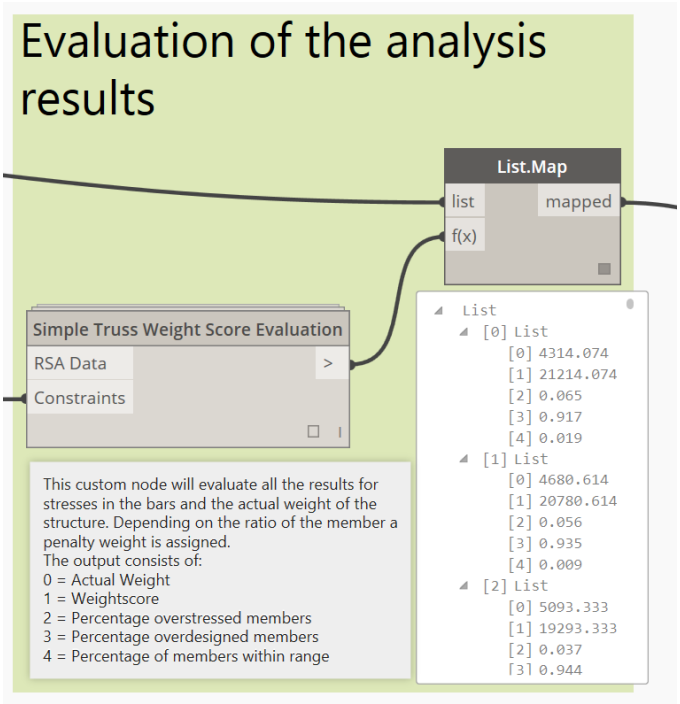
(3) The first element in the sub list, contains a list of results for the maximum stresses for all bars, for all load cases. In this case there are 3 load cases, so three results per bar.
Syntax
a[0][1][0] = the specific bar
a[0][1][0][0] = the result for that bar

All the design options are saved in separate .rtd files in the folder that was indicated in the *Directory path* node in Step 2 - Constant input values and constraints. These models can be reviewed in Robot Structural Analysis. The file names are based on the parameter list that is defined inside of the custom node (see also Case 3 - Step 4 – Analysis of the model)


Parametric Run
File
Home
Share
View
Copy
Paste
Cut
Copy path
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
New folder
New item
Easy access
New
Properties
Open
Edit
History
Open
Select all
Select none
Invert selection
Select
This PC
Local Disk (C:)
Results
Armadillo
Parametric Run
Search Parametric Run
Favorites
This PC
Network
Name
Date modified
Type
Size
TH 2.300000_TW 1.600000_CHS 219.1x8_CHS 88.9x6.3_145612.rtd 10/11/2015 14:56 Autodesk RTD Do... 1,488 KB
TH 2.300000_TW 1.600000_CHS 193.7x6_CHS 88.9x6.3_145448.rtd 10/11/2015 14:54 Autodesk RTD Do... 1,480 KB
TH 2.300000_TW 1.600000_CHS 168.3x6_CHS 88.9x6.3_145323.rtd 10/11/2015 14:53 Autodesk RTD Do... 1,476 KB
TH 2.300000_TW 1.600000_CHS 139.7x6_CHS 88.9x6.3_14521.rtd 10/11/2015 14:52 Autodesk RTD Do... 1,476 KB
TH 2.300000_TW 1.500000_CHS 244.5x10_CHS 88.9x6.3_145024.r... 10/11/2015 14:50 Autodesk RTD Do... 1,476 KB
TH 2.300000_TW 1.500000_CHS 219.1x8_CHS 88.9x6.3_14497.rtd 10/11/2015 14:49 Autodesk RTD Do... 1,460 KB
TH 1.800000_TW 1.600000_CHS 193.7x6_CHS 88.9x6.3_132233.rtd 10/11/2015 13:22 Autodesk RTD Do... 1,100 KB
TH 1.800000_TW 1.600000_CHS 168.3x6_CHS 88.9x6.3_132227.rtd 10/11/2015 13:22 Autodesk RTD Do... 1,100 KB
TH 1.800000_TW 1.600000_CHS 139.7x6_CHS 88.9x6.3_132222.rtd 10/11/2015 13:22 Autodesk RTD Do... 1,100 KB
TH 1.800000_TW 1.500000_CHS 244.5x10_CHS 88.9x6.3_132216.r... 10/11/2015 13:22 Autodesk RTD Do... 1,100 KB
TH 1.800000_TW 1.500000_CHS 219.1x8_CHS 88.9x6.3_132211.rtd 10/11/2015 13:22 Autodesk RTD Do... 1,096 KB
TH 1.800000_TW 1.500000_CHS 193.7x6_CHS 88.9x6.3_13225.rtd 10/11/2015 13:22 Autodesk RTD Do... 1,096 KB
TH 1.800000_TW 1.500000_CHS 168.3x6_CHS 88.9x6.3_13220.rtd 10/11/2015 13:22 Autodesk RTD Do... 1,096 KB
TH 1.800000_TW 1.500000_CHS 139.7x6_CHS 88.9x6.3_132150.rtd 10/11/2015 13:21 Autodesk RTD Do... 1,012 KB
159 items
1 item selected 1.45 MB

Step 7 - Evaluation of the analysis results





When the results from the analysis are known it's time to evaluate them. The script that was initially set up in Case 3 - Step 6 – Evaluation of the results, is now merged into a custom node called *Simple Truss Weight Score Evaluation*. With this node it will be easier to evaluate multiple lists of results, and to return additional evaluation values.

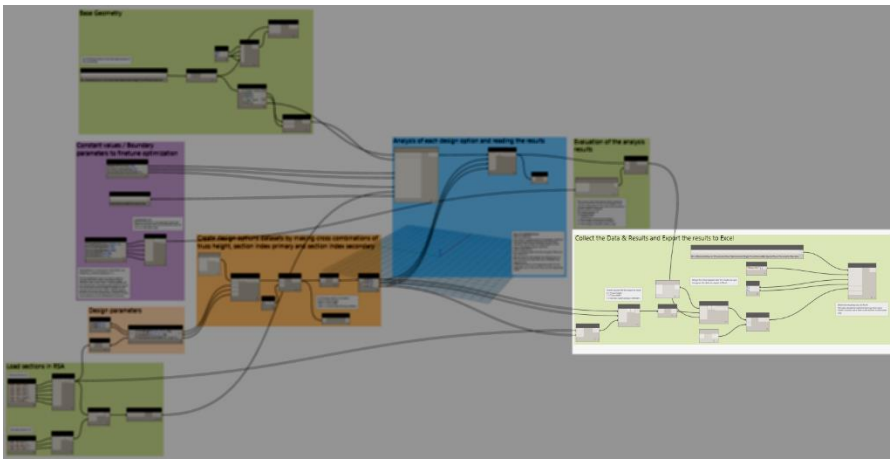
This custom node will evaluate all the results for stresses in the bars and the actual weight of the structure. Depending on the ratio of the member a penalty weight is assigned. The output of this node contains 5 results for each design option:

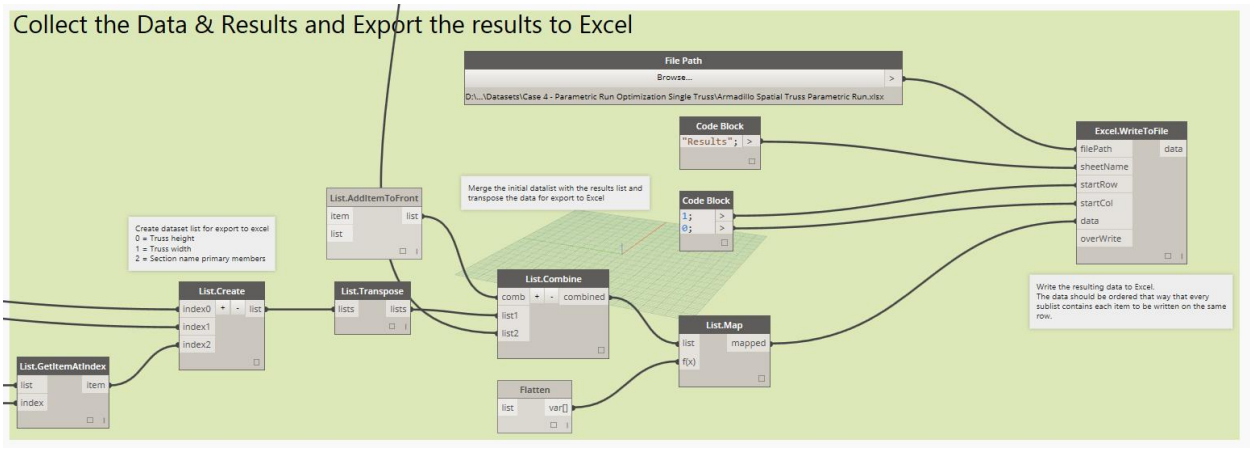
The output consists of:

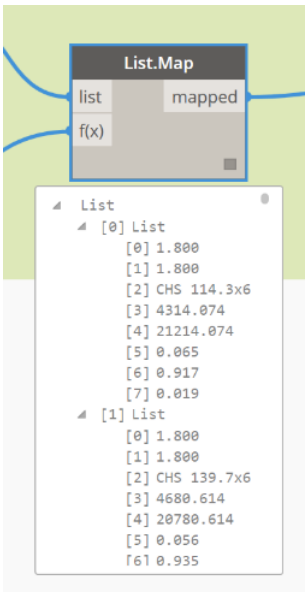
0 = Actual Weight
1 = Weightscore
2 = Percentage overstressed members
3 = Percentage overdesigned members
4 = Percentage of members within range

More details of this custom node can be found further in this document.

Step 8 - Export of evaluation results to Excel







In this final step all the initial data and evaluation results are collected and grouped in one list with the *List.Combine* and *List.AddItemToFront* nodes. As the results need to be written in a "row-direction" to Excel, the resulting data lists first need to go through the *List.Transpose* node (except for the evaluation results, which are in the right form already).

The final result is then exported to Excel with the *Excel.WriteToFile* node.

Before running this step, don't forget to point to the right path for the Excel file where you want to export the results to.

When the results are written to Excel, the post-processing can start. In this case, some diagrams are made to easily understand the results of all the design options.

Truss Height (m)	Primary Section	Secondary Section	Actual Weight (kg)	Weight Score	Overstressed Members	Overdesigned Members	Within Range Members
2.00	1.9	CHS 114.3x6	4451.52	21351.52	6.48%	91.67%	1.85%
2.00	1.9	CHS 139.7x6	4815.28	20415.28	5.09%	93.52%	1.39%
2.00	1.9	CHS 168.3x6	5224.86	18424.86	2.78%	94.44%	2.78%
2.00	1.9	CHS 193.7x6	5588.62	18038.62	1.85%	96.76%	1.39%
2.00	1.9	CHS 219.1x8	6936.90	19536.90	1.85%	98.15%	0.00%
2.00	1.9	CHS 244.5x10	8502.80	19102.80	0.00%	98.15%	1.85%
2.00	2	CHS 114.3x6	4496.47	21396.47	6.48%	91.67%	1.85%
2.00	2	CHS 139.7x6	4860.22	20460.22	5.09%	93.52%	1.39%
2.00	2	CHS 168.3x6	5269.81	18469.81	2.78%	94.44%	2.78%
2.00	2	CHS 193.7x6	5633.56	18083.56	1.85%	96.76%	1.39%
2.00	2	CHS 219.1x8	6981.85	19581.85	1.85%	98.15%	0.00%
2.00	2	CHS 244.5x10	8547.74	19147.74	0.00%	98.15%	1.85%
2.00	2.1	CHS 114.3x6	4542.34	21442.34	6.48%	91.67%	1.85%
2.00	2.1	CHS 139.7x6	4906.09	20506.09	5.09%	93.52%	1.39%
2.00	2.1	CHS 168.3x6	5315.68	18515.68	2.78%	94.44%	2.78%
2.00	2.1	CHS 193.7x6	5679.43	18129.43	1.85%	96.76%	1.39%
2.00	2.1	CHS 219.1x8	7027.72	19627.72	1.85%	98.15%	0.00%

FIG. 35 - RESULTS OF THE PARAMETRIC RUN IN EXCEL

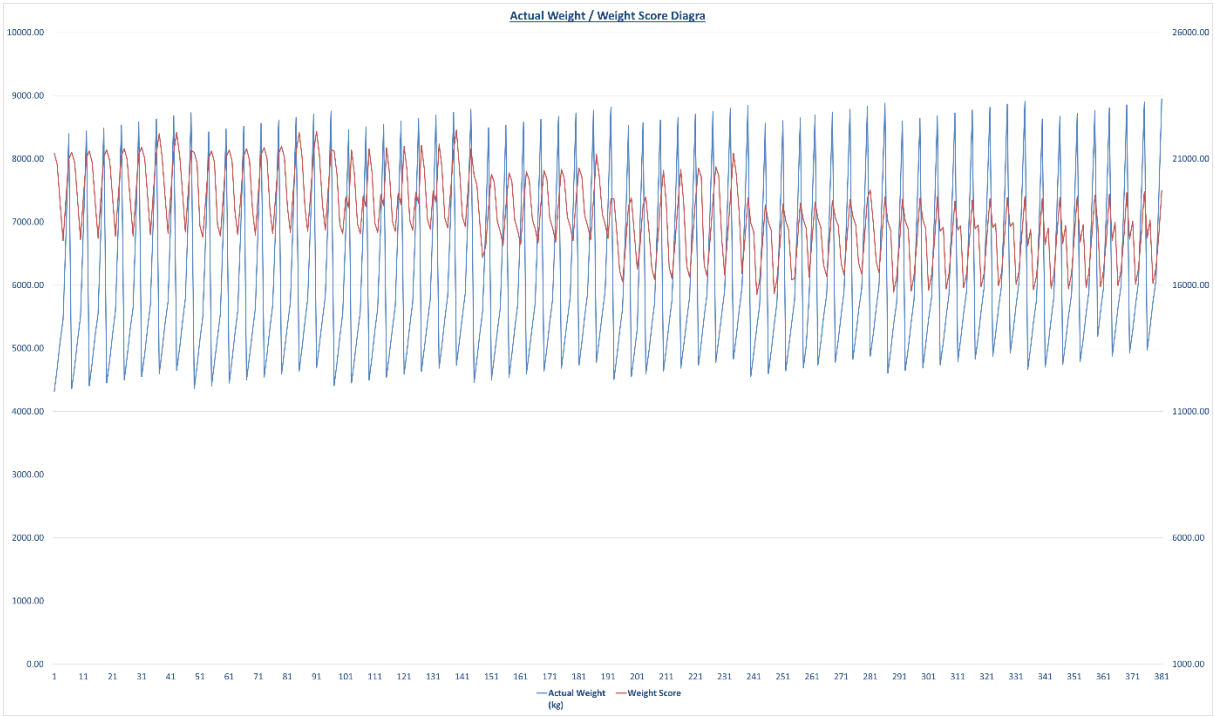
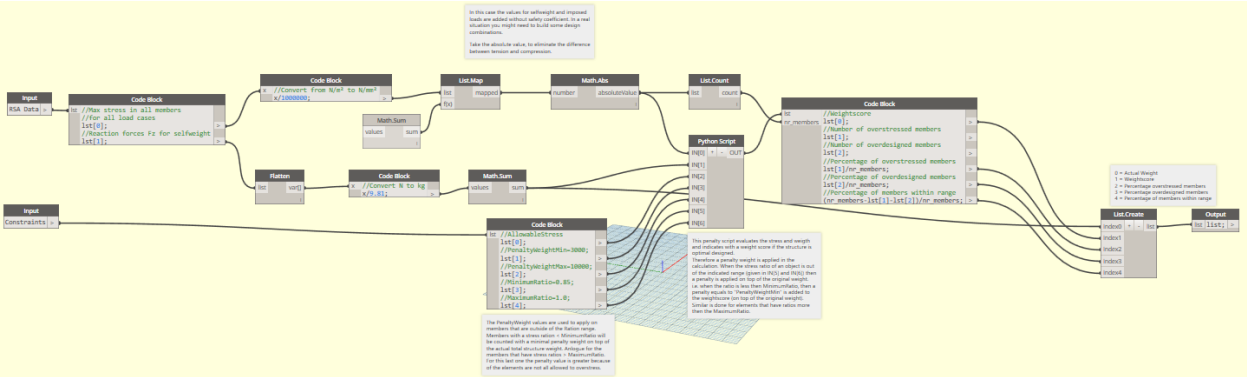
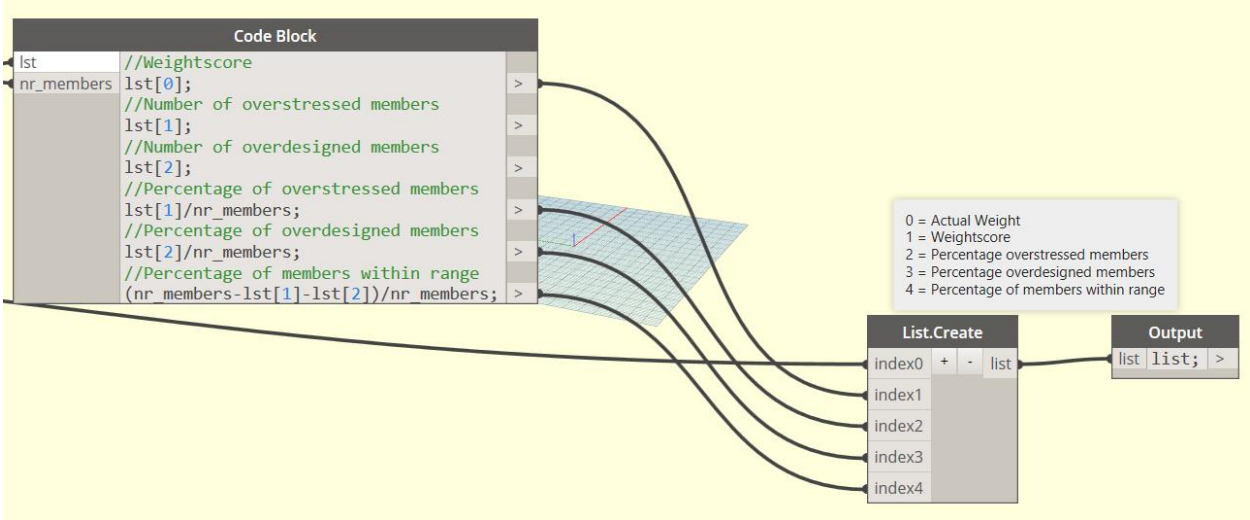


FIG. 36 - EVALUATION OF ACTUAL WEIGHT VS WEIGHT SCORE

For this particular case, it's clear that the structure is quite overdesigned. This is because of the diagonals modelled in the inner part of the truss. Their ratio is smaller than the required "Minimum Ratio" which causes them to set a penalty on the weight. You can change this by increasing the width of the acceptable Ratio range.

Simple Truss Weight Score Evaluation





The content of this custom node is almost similar to the group of nodes described in Case 3 - Step 6 – Evaluation of the results, except for the of the *Python script* node. This custom node will evaluate all the results for stresses in the bars and the actual weight of the structure. Depending on the ratio of the member a penalty weight is assigned.

The output of the Python script is a list that consists of the next indices:

0 = Actual Weight
1 = Weight score
2 = Percentage overstressed members
3 = Percentage overdesigned members
4 = Percentage of members within range

Case 5 – Genetic Optimization of a single complex truss

The "parametric run" method is very time consuming, as it considers every design "combination" option, even if the design option results into stress ratios that are beyond the allowable stress ratio range (read: "high weight score"). Besides that, the parametric run will only calculate the given set of combinations, and won't find alternatives that might be better.

To speed this up, and to find more accurate results we need to introduce Genetic Algorithms in the analysis process. The GA Optimization will use specific methods making a natural selection of the initial solution list and by using cross-over and mutation algorithms in order to find the optimal solution.

Compare it with "introducing the natural evolution theory of Darwin" into computational design.

The main parts of the script that has been made in Case 4, can be reused by adding specific nodes from the **Optimo for Dynamo** package. Read more about the use of this package via this link.

The base geometry of the roof is again the SAT export of the Mass object initially created in Revit by the architectural designer.

DATASETS

RSA



Start model:
Armadillo Platform - Single Truss Analysis_start
Finalized models:
Multiple files stored in the 'Genetic Optimization' subfolder

Dynamo

Workspace:
Armadillo Spatial Truss - 04 Genetic Algorithm Optimization.dyn

Custom nodes:
V bracings.dyf
Chained line.dyf
Armadillo Spatial Truss Fitness Function.dyf
Armadillo Spatial Truss Optimo Evaluation.dyf
Simple Truss Weight Score Evaluation.dyf

External input
Platform Roof.sat

To open the dataset files:

1. Launch the sandbox version of Dynamo 0.9 (C:\Program Files\Dynamo 0.9\DynamoSandbox.exe) or use Dynamo Studio 2016 (v. 0.9.3072)
2. Open the Dynamo script (.dyn). In the next chapters it will be specified which file to open.
3. Launch Robot Structural Analysis 2016
4. Open the RSA start model file. Without this step the script won't work!
5. Connect the wires as shown in the class and play with the parameters.

Representation of Optimization Problem

Before we start with building up the script, it is important to first represent the optimization problem which consists of an “objective”, “design variables” or “population”, “design variables domain” and finally the design “constraints”.

Objective

With the optimization we want to find a structural solutions that has a minimal weight and has stress ratios of the steel members which is below the allowable ratio.

This means we have somehow a double objective. This could be done with a MOO (Multi-Objective Optimization), but in that case we would get solutions that have a minimal weight and still are overstressed. Therefore the objectives are combined in a “weight score”. Finding the minimal “weight score” is the real objective in this script. This weight score is defined in the fitness function by adding penalty weights to the real weight. These penalty weights take the stress ratio into account by adding a specific value for overdesigned and overstressed elements.

Design Variables

The design variables for this optimization are expressed with three parameters:

- Truss Height
- Truss Width
- Primary Section Index (defines the cross section type that can be used, by referring to the section list using an index).

These are thus the variables that may be changed by the routine to get to an optimal solution.

Design Variables Domain

We need to give some restrictions for the design variables and give the optimization routine a boundary within to search an optimal solution for. In this case the domains are set as follows:

- 1.5 m ≤ Truss Height ≤ 3.0 m
- 1.5 m ≤ Truss Height ≤ 3.0 m
- 0 ≤ Primary Section Index ≤ n-1 with n = the length of the section list.

Constraints

As we don't want to have solutions with a minimal weight but with overstressed members, we need to introduce constraints. These constraints consist of some rules that will limit the usage of some design variables.

In this script final constraints are:

- MinimumRatio
- MaximumRatio

These constraints are at their way defined by the *AllowableStress* factor. When the resulting stress ratio of a steel member is lower than the MinimumRatio, then a penalty weight (= PenaltyWeightMin) is added to the real weight. When the resulting stress ratio of a steel member is greater than the MaximumRatio, then a penalty weight (= PenaltyWeightMax) is added to the real weight. This last one is a significantly greater value, as overstressed members are totally unacceptable in the solution list.

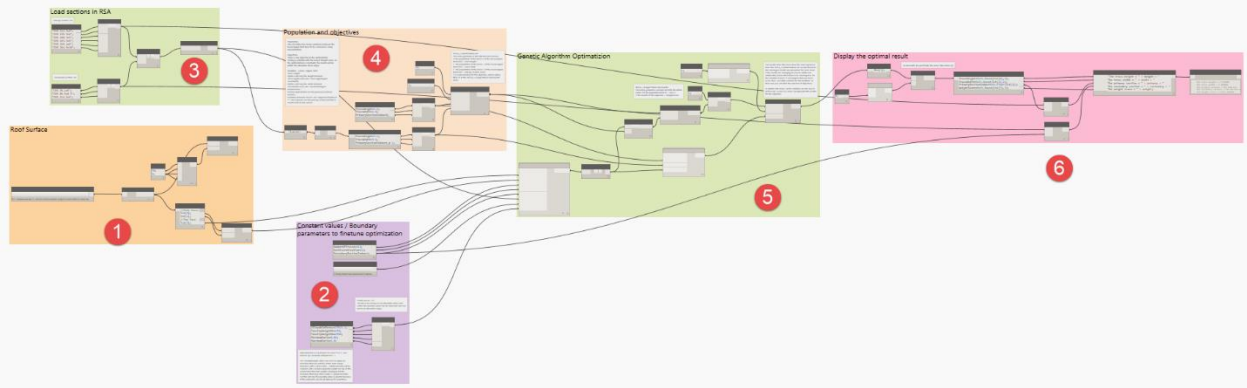
Code Block

```
AllowableStress=235/1.1;
PenaltyWeightMin=50;
PenaltyWeightMax=500;
MinimumRatio=0.80;
MaximumRatio=1.0;
```

Read more on Calculation of penalty weight

General overview of the script

In this script constant values and an initial solution list generated by Optimo will be connected to a so called “fitness function” which is processed by the Optimo NSGA algorithm nodes.



The total script consists of 6 big steps:

Step 1 = Input of the base roof geometry

Step 2 = Constant input values and constraints

Step 3 = Load sections to RSA

Step 4 = Generation of initial population list

Step 5 = Optimization using Genetic Algorithms

Step 6 = Interpretation of optimal result

Before you run the script, please read Important notice – Setting in Robot Structural Analysis

Step 1 - Input of the base roof geometry

This is already explained in Case 3 - Step 1 – Input parameters.

Step 2 - Constant input values and constraints

This is already explained in Case 4 - Step 2 - Constant input values and constraints

More about the application of the penalty weight: Calculation of penalty weight

For the optimization process, the values of the PenaltyWeight are very important. They indicate the importance of the fitness of a solution. In this case the PenaltyWeightMax = 1000. This means that stress ratios that exceed 1.0 are totally unacceptable.

The PenaltyWeightMin = 50, which is a very small value, but we are aware that a lot of secondary members will not meet the MinimumRatio as they serve more for constructional reasons.

Code Block

```
AllowableStress=235/1.1;
PenaltyWeightMin=50;
PenaltyWeightMax=1000;
MinimumRatio=0.80;
MaximumRatio=1.0;
```

You can of course play with these values to achieve another optimal result or to finetune your current result even.

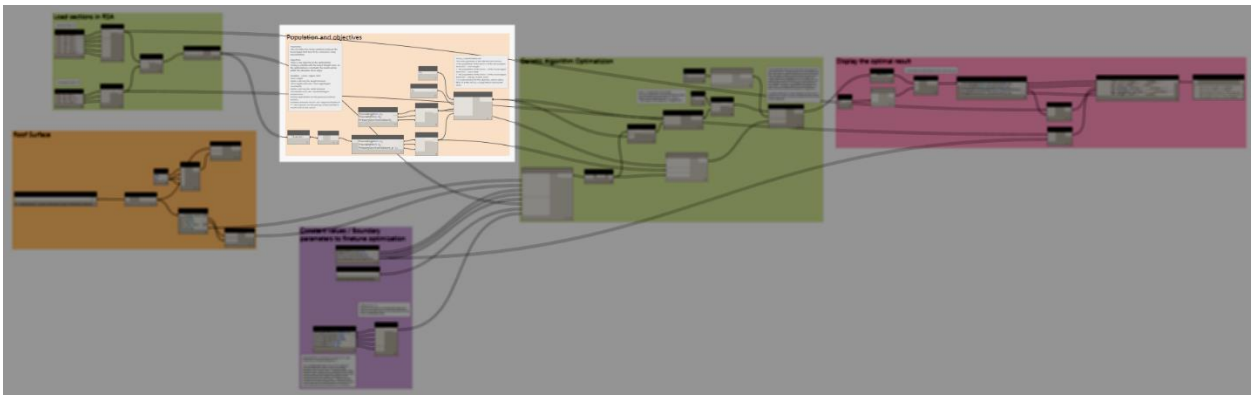
Code Block

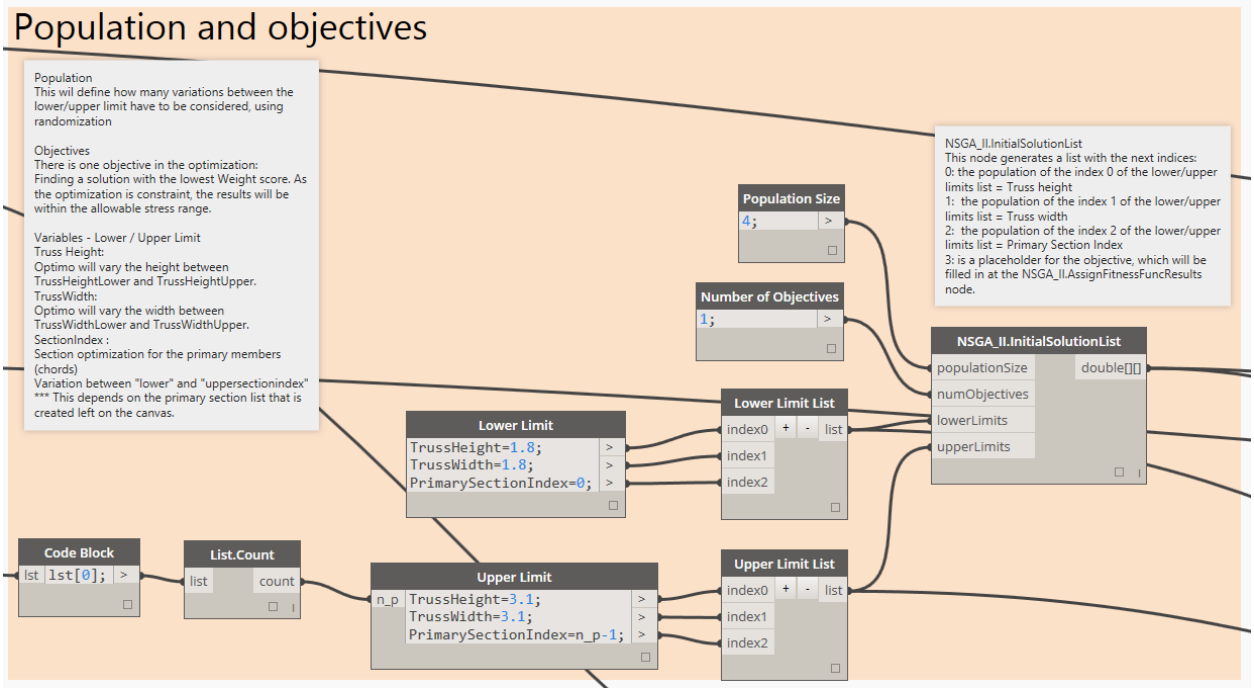
```
AllowableStress=235/1.1;
PenaltyWeightMin=100;
PenaltyWeightMax=1000;
MinimumRatio=0.25;
MaximumRatio=0.95;
```

Step 3 - Load sections to RSA

This is already explained in Case 2 - Step 12 – Load sections in RSA

Step 4 - Generation of initial population list





The goal of this step is to generate the initial set of random variables within the provided range and with the size of population defined by user. Therefore the Design Variables need to be defined and set up within a Design Variables Domain.

The *NSGA_II.InitialSolutionList* node creates a list of randomized numbers (initial population) within the range specified and with the number of defined variables.

This node has 4 input ports:

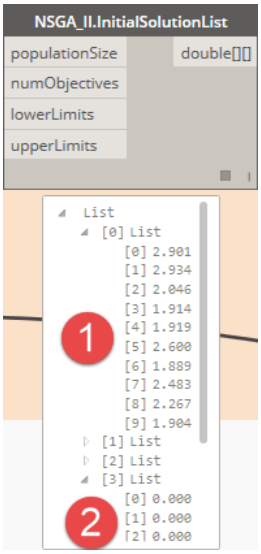
- **PopulationSize** : this is the amount of numbers that need to be generated for each variable.
- **numObjectives *:*** this is the amount of fitness functions that need to be optimized. This needs to be equal to the number of list input ports of the *List of Fitness Functions* node (which is a List.Create node).
- **lowerLimits** : this represents a list of the minimal values of the variables that need randomization
- **upperLimits** : this represents a list of the minimal values of the variables that need randomization

The size of the list for lowerLimits and upperLimits needs to be the same.

The output of the *NSGA_II.InitialSolutionList* node consists of a list containing:

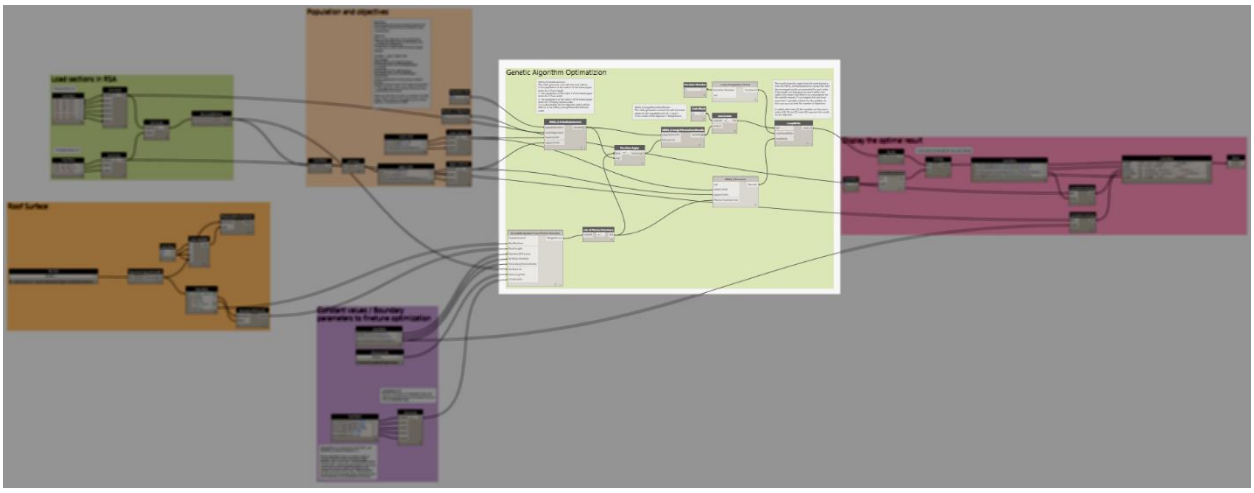
(1) an amount of sub lists, one for each variable with an amount (PopulationSize) of randomized numbers (between lowerLimit and upperLimit). This is called the "parents solution list".
In the example below: 3 sub lists with 10 random numbers each (PopulationSize = 10)

(2) the sub list with the last indices represent the objectives. The number of sub lists depends on the numObjectives value. The objectives are all zero when they come out of InitialSolutionList and will be overwritten in the *NSGA_II.AssignFitnessFuncResults* node.
In the example below: 1 sub list (for 1 objective) with 10 zero values.

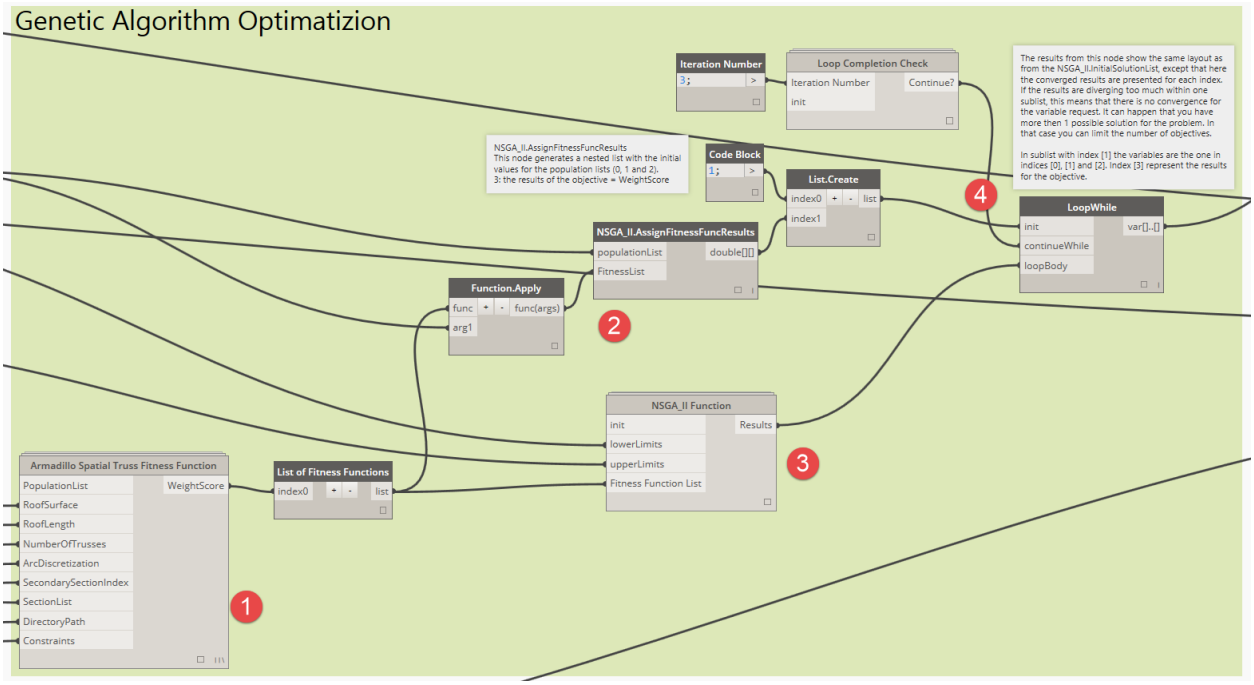


The number of variables can be changed by adding or removing an index to both the *Lower Limit List* and *Upper Limit List* nodes (they must have the same number of indexes).

Step 5 - Optimization using Genetic Algorithms



This is the core of the optimization script. The nodes in this part are implemented with the NSGA algorithms. NSGA is a non-domination based genetic algorithm for multi-objective optimization. It works in the following way: An initial population is initialized and sorted. Each individual in the population is sorted based on their fitness (determined by the fitness functions). The parents for the next generation of population are selected by using binary tournament selection. These parents create the next generation with crossover and mutation operators. This continues until a goal is reached or until a set number of generations has been created.



Tag 1 - Fitness Functions

The *Armadillo Spatial Truss Fitness Function* is a custom node that is designed that way to read the population list variables (in this case a list of three variables: Truss height, Truss width and Primary Section Index). Each of these lists contains a given amount of randomized values (amount is equal to the population size). The custom node contains the *Armadillo Spatial Truss Optimo Evaluation* function, which is the backbone of this calculation.

The *List of Fitness Functions* node represents the number of functions to be optimized.

Tag 2 - Assign Fitness Function Results

The *NSGA_II.AssignFitnessFuncResults* node gets the list of objectives as input parameters and assigns them to the *populationList* input port. By using the *Function.Apply* node, this workflow enables Optimo to accept objective functions as nodes or packages of nodes.

The population list is also assigned to the *Armadillo Spatial Fitness Function* through the *Function.Apply* node, but in the custom node, only the variables lists are considered and not the objectives lists. This node evaluates the fitness of each member of the population and sends that data to the AssignFitnessFuncResults node, where the zeroes that were created in the initial population node get overwritten.

Tag 3 – NSGA Genetic Algorithm Function

The content of the *NSGA_II Function* node is displayed below. The function of this node is to run the NSGA II algorithm recursively to generate the specified number of generations.

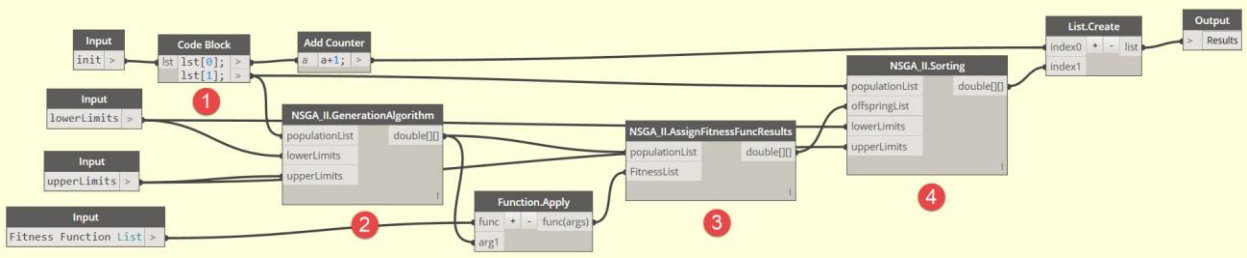


FIG. 37 - CONTENTS OF THE CUSTOM NODE NSGA_II FUNCTION

(1) The first input to this custom node, "init" is a list containing two elements:
The first element in the list (index [0]) is a counter which keeps track of how many times the function has run. This is needed in the *Loop Completion Check* node (discussed later).
The second element (index [1]) in the list is the population from the previous generation.

(2) This second element from the *init* input, gets fed into the *NSGA_II.GenerationAlgorithm* node as an input along with the *lowerLimits* and *upperLimits* (which serve the same purpose as in the main workspace). The Generation Algorithm creates a new population based on the previous generation, called “children solution set” or “offspring”

(3) The fitness of this new generation is then tested with the same fitness functions as the main Dynamo file (*Armadillo Spatial Truss Fitness Function*), and the fitness function results override the objectives in this new generation.

(4) The *NSGA_II.Sorting* node takes the previous generation as well as the current generation, and re-arranges them in order based on their fitness (best results first), using Pareto Front sorting.

The result gets added to a list which contains all of the previous generations' data.

Tag 4 – Loop completion check

The *Loop Completion Check* node simply compares the counter from the *NSGA_II Function* node and compares it to the *Iteration Number* (the max number of iterations that the algorithm should run). When the counter reaches the *Iteration Number*, the algorithm stops looping and displays the result at the output of the *LoopWhile* node.

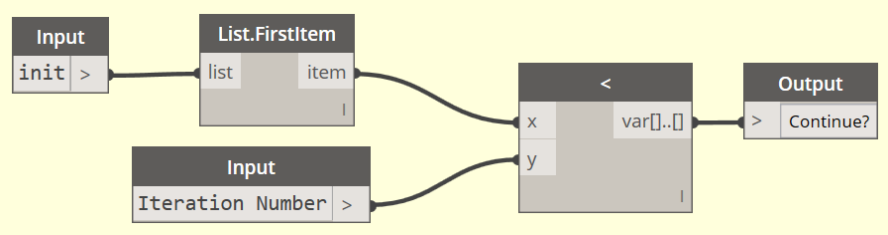
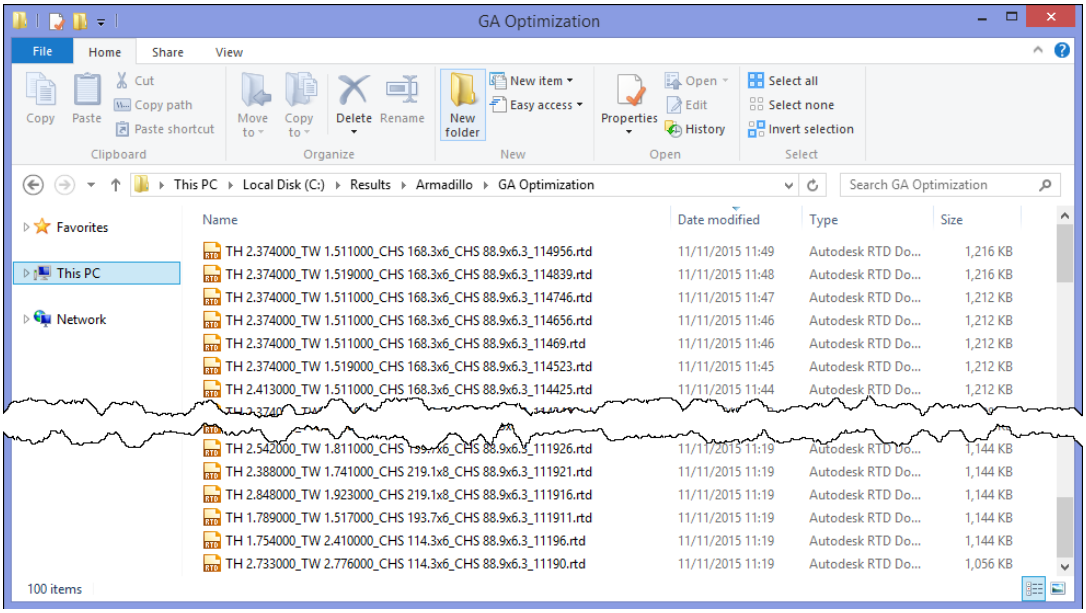


FIG. 38 - CONTENTS OF THE CUSTOM NODE LOOP COMPLETION CHECK

Solution list files

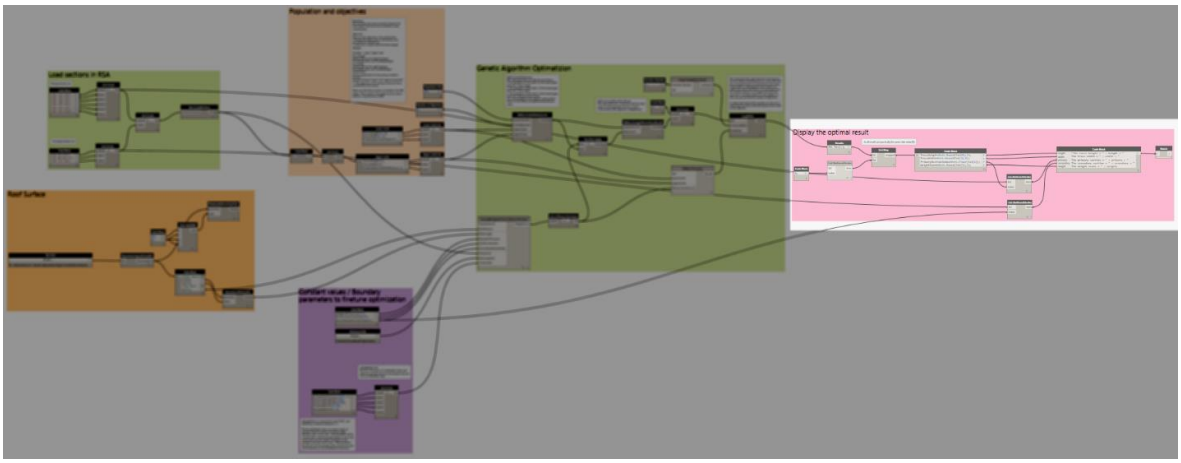
During the progress of the script, every solution is saved in a separate file in the previously indicated path.



With this it is possible to review your models, but it is also possible to see how many models have been created and what time it took to perform the whole optimization process.

In this case 100 models were calculated in 30 mins.

Step 6 - Interpretation of optimal result



When you add some watch nodes to the output of the *LoopWhile* node (see tag 4 from Step 5 - Optimization using Genetic Algorithms), then you get results in this form:

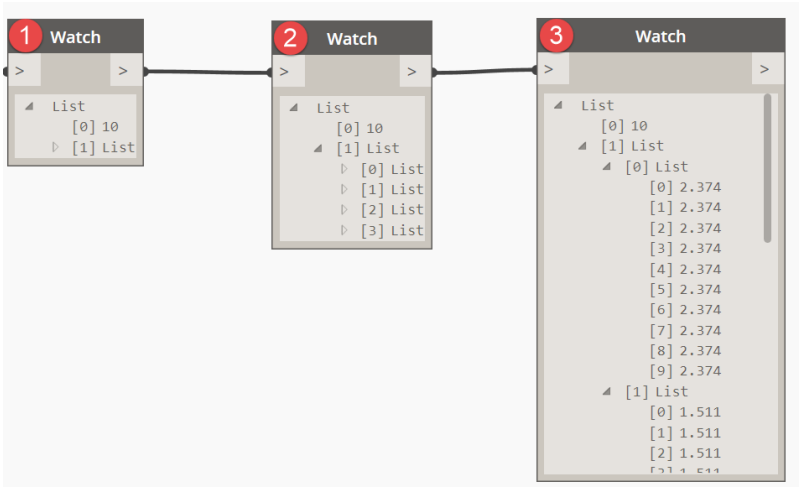


FIG. 39 - RESULTS AFTER OPTIMIZATION

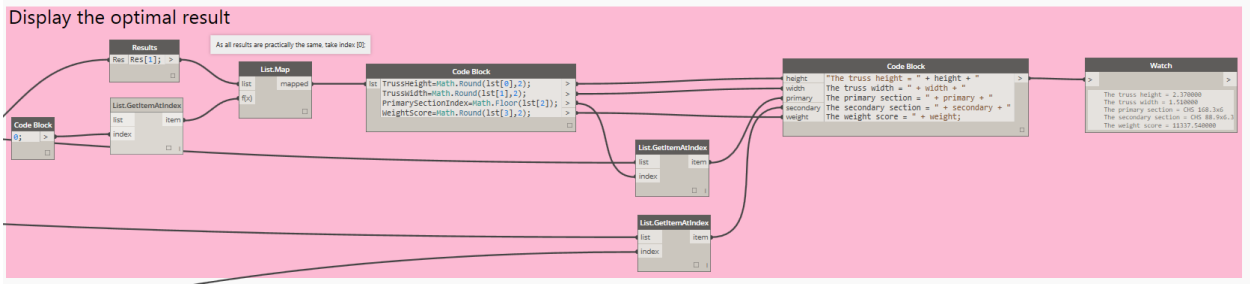
The results list consists of two main lists:

(1) The first item (a[0]) is the number of iterations that has been applied. In this case the genetic algorithms have been called for 10 runs.

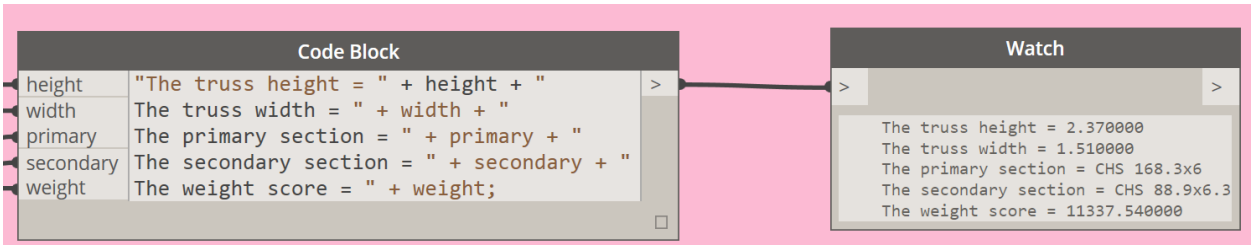
(2) The second item (a[1]) represents the final population list (variables) and the achieved objectives (weightscore). In this case the population size used is 10.
The first 3 sub lists contains the variables:
a[1][0] = the result for the optimal truss height
a[1][1] = the result for the optimal truss width
a[1][2] = the result for the optimal primary section index
The last sub list (a[1][4]) contains the resulting weightscore for this optimal solution.

There is a single optimal solution if all values in one sub list are equal.

These results are then read in the next group of nodes…



… and end up in a *Watch* node displaying the optimal result for the configured constraints.



When the results are not all equal (like below), then there is no real good convergence found, which could mean there are multiple solutions, or which means that the *Number of Iterations* that has been set is too small.

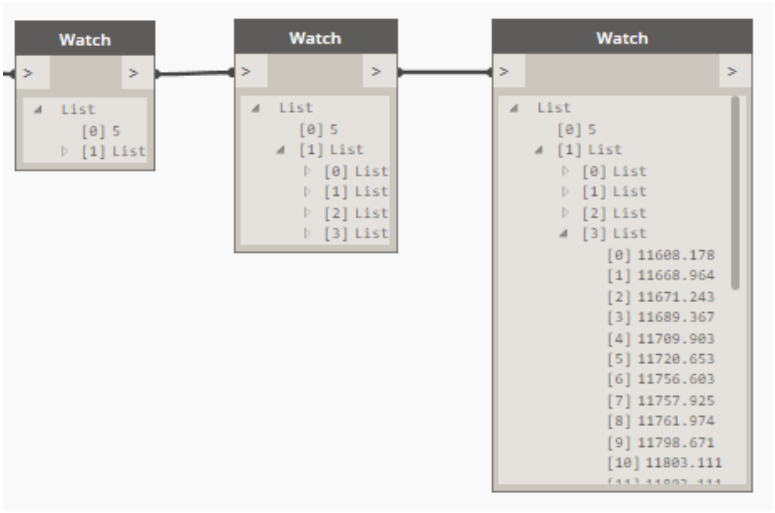
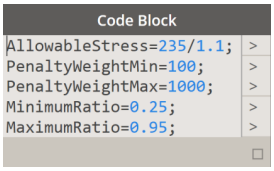
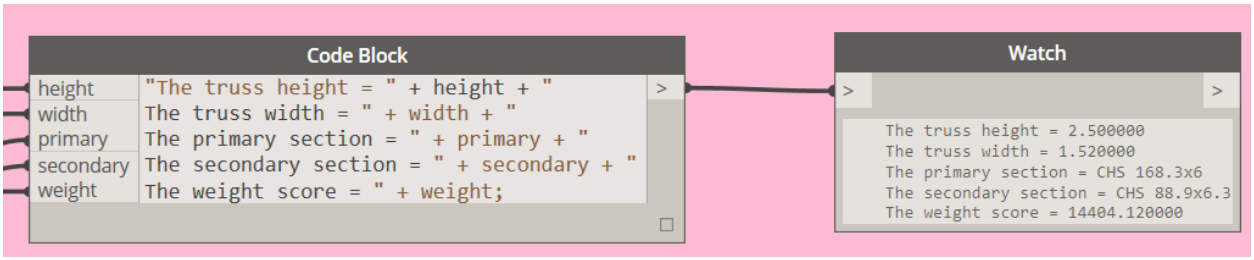


FIG. 40 - INCOMPLETE OPTIMIZATION RESULTS

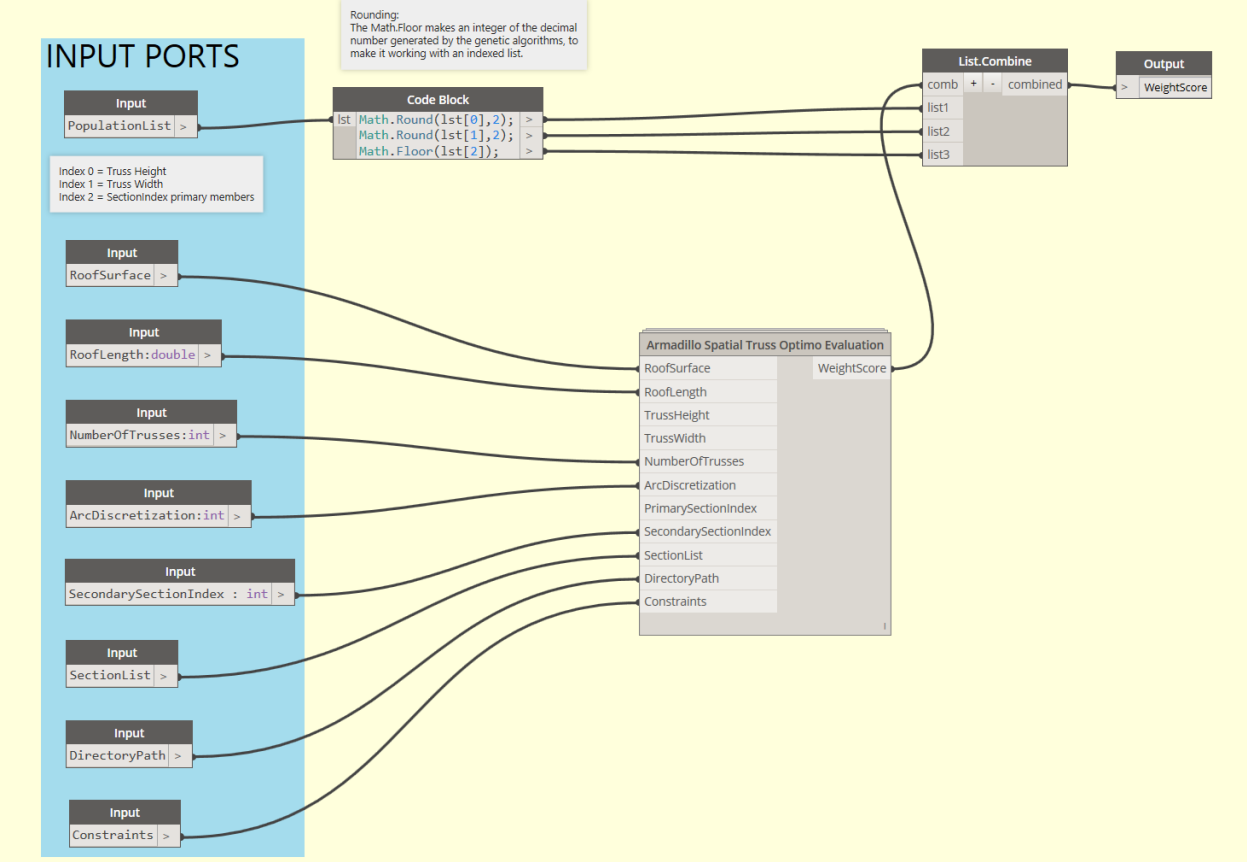
By playing with the constraints, the population size and the number of iterations, you can get other results, which might be more fine-tuned or accurate.

This one for example uses different constraints and a population of 20 with 10 iterations of the genetic algorithm.





Armadillo Spatial Truss Fitness Function



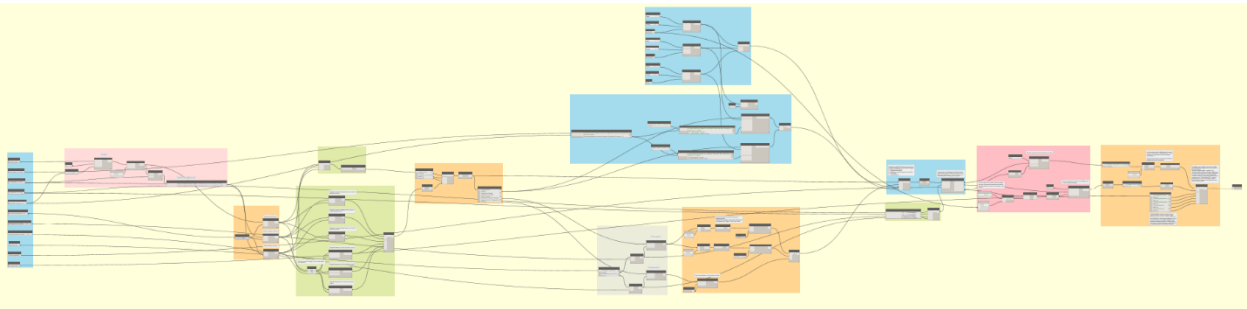
As explained in Step 4 - Generation of initial population list, the population list consists of lists of variables and a list of objectives.

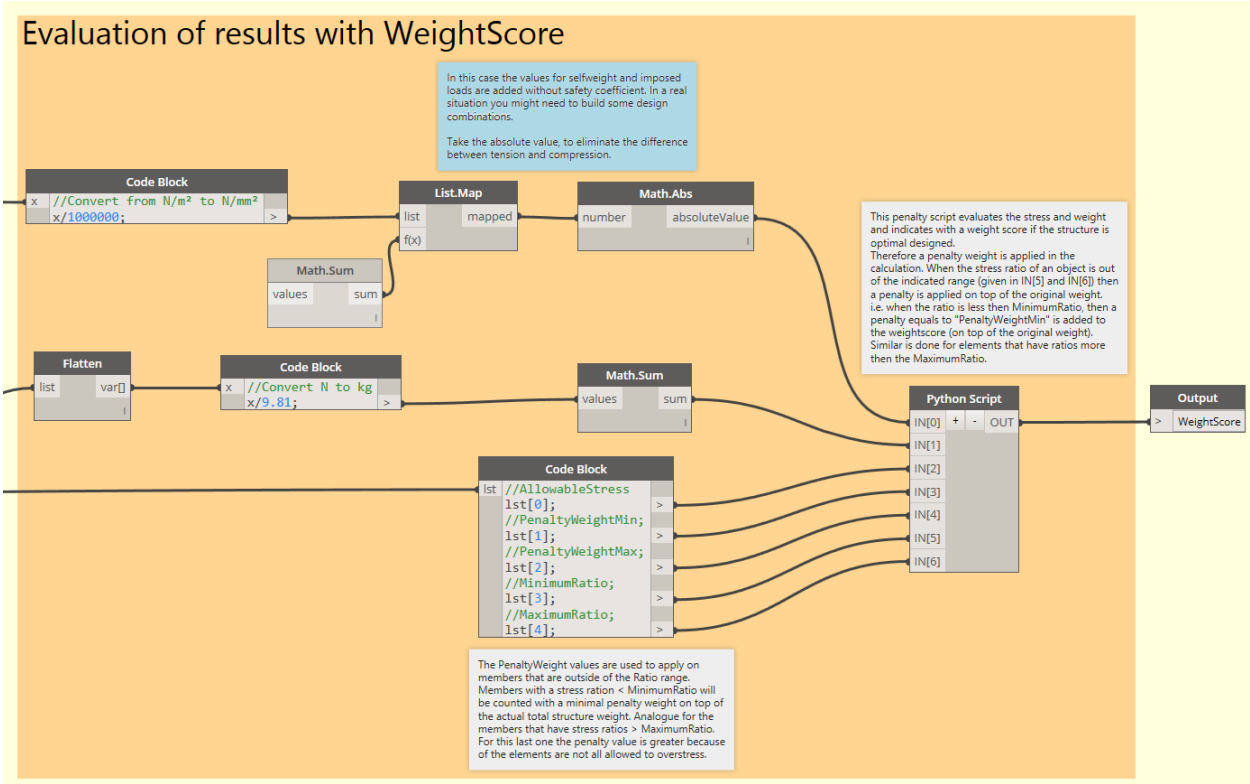
The *Code Block* node, takes only the variables from this population list.
Index [0] = the variables for the truss height
Index [1] = the variables for the truss width
Index [2] = the variables for the primary section index

These values are expressed as “doubles”. To have constructible dimensions the height and width are rounded to 2 digits. The section index needs to be an integer, so we round this one to the first integer number below the actual variable value.

The lists of three variables are then combined with the *Armadillo Spatial Truss Optimo Evaluation* custom node to test their fitness.

Armadillo Spatial Truss Optimo Evaluation





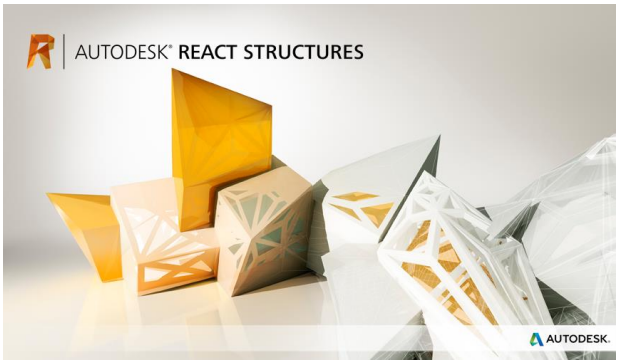
This custom node is a copy of the *Armadillo Spatial Truss Analysis* custom node containing the geometry outline and simulation model generation. This is explained completely in Case 4 - Step 6 - Analysis of the design options.

One difference: in this custom node the group of nodes for the evaluation of the results with a weight score, is added at the end, so that the output represents a value of the weight with addition of penalty weights. More info on that can be found in Case 3 - Step 6 – Evaluation of the results. The output of this Python script consists only of the WeightScore.

The main intention of this Optimo script is to find a solution set which represents the lowest score.

React Structures

The scripts that are used in this handout, can also be used with the Technology Preview of Autodesk React Structures, which can be downloaded here: http://react.autodesk.com/



In that case you need to perform next steps:

1. Launch Autodesk React Structures TP2
2. Open the React start file *Armadillo_start.rstructure* which is included in the datasets of Case 2. This is needed to have all the pre-defined custom *Supports* available in the software.
3. Open the Dynamo add in via the Model tab
4. Open the specific Dynamo script (.dyn).
5. Connect the wires as shown in the class and play with the parameters.

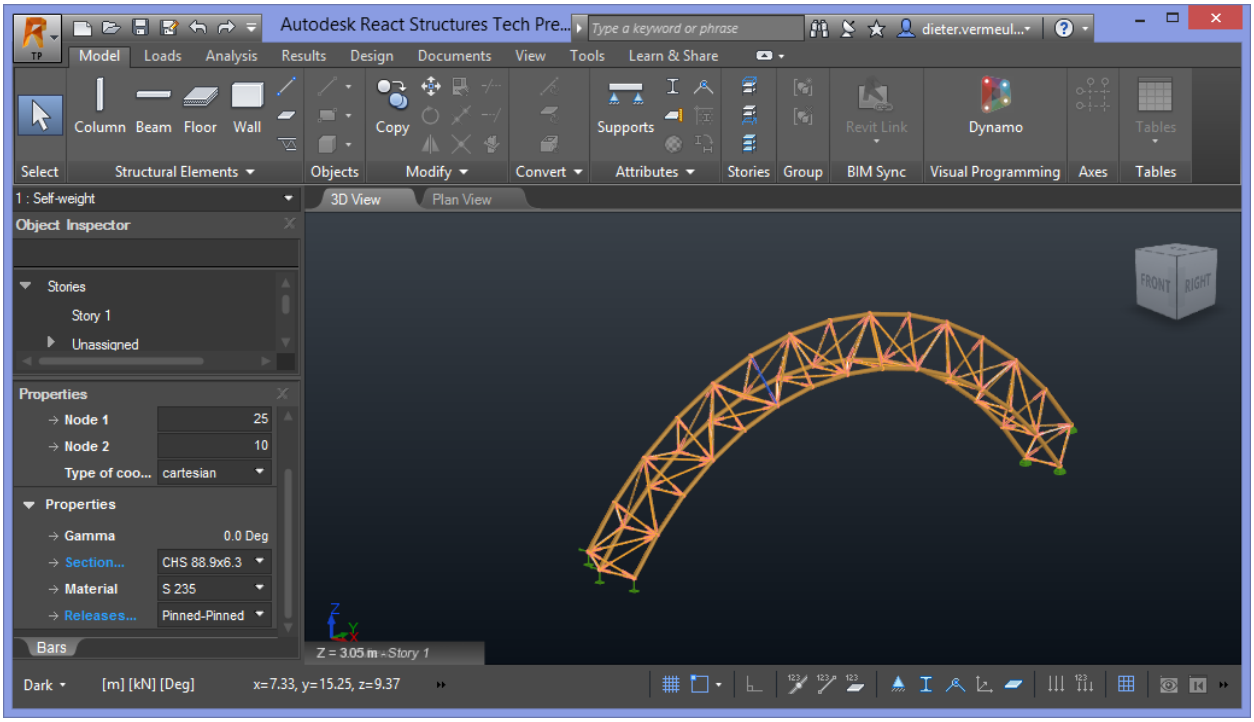


FIG. 41 - ANALYSIS MODEL IN AUTODESK REACT STRUCTURES

Learning resources

More learning resources on the products that have been used in this class can be found below:

Robot Structural Analysis	http://autodesk.typepad.com/bimtoolbox/2015/06/robot-structural-analysis-learning-resources.html
React Structures Tech Preview	http://react.autodesk.com/learning/
Dynamo	http://au.autodesk.com/au-online/classes-on-demand/class-catalog/2013/vasari/ab2551#chapter=0 http://dynamobim.com/learn/ http://dynamoprimer.com/ http://dynamobim.com/forums/forum/dyn/ http://www.revitforum.org/dynamo-bim/24005-dynamo-learning-resources.html#post136270
Optimo for Dynamo	http://bim-sim.org/Optimo/index.html http://github.com/BPOpt/Optimo/wiki/0_-Home
Structural Analysis for Dynamo	https://www.youtube.com/watch?v=yZrAGo089B0