ABSTRACT

SHAPE FINDING OF FABRIC STRUCTURES USING GRID METHOD

by
Armin Bischof

Fabric structures can span large distances with a very low self-weight, which makes them very efficient for covering huge areas. They have attractive shapes and an appealing architecture. These properties make them very popular and they are used in many different kinds of buildings. Airports, stadiums and halls are typical examples where the use fabric structures is often recommendable.

Fabric structures are built mainly out of fabric and cables; two structural members that can only take tension stresses. Therefore these structural members must be prestressed in such a way that their shape can carry all loads that might affect the structure and maintain the tension. To find this shape is one important step of the design process of fabric structures.

The design process of fabric structures includes three steps: shape finding, static analysis and patterning. This thesis in concentrates on the shape finding of fabric structures using the Grid method. Different mathematical methods and computer codes are used to perform the shape finding of a fabric structure. These methods and codes are compared and discussed in order to find their advantages and disadvantages. Finally there are a few words about the nonlinear static analysis of fabric structures. One result of this thesis is two new Java Codes to compute the shape of fabric structures using the Grid method and Gaussian elimination. The code shapeApp.java has been written for fabric structures with fixed boundaries and the code edgeshapeApp.java, for fabric structures with edge cables.

SHAPE FINDING OF FABRIC STRUCTURES USING GRID METHOD

by
Armin Bischof

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Civil Engineering

Department of Civil and Environmental Engineering

May 2003

Blank Page

APPROVAL PAGE

SHAPE FINDING OF FABRIC STRUCTURES USING GRID METHOD

Armin Bischof

Dr. William R. Spillers, Master Thesis Advisor — Date
Distinguished Professor of Civil and Environmental Engineering, NJIT

Dr. Ala Saadeghvaziri, Committee Member — Date
Professor of Civil and Environmental Engineering, NJIT

Dr. Methi Wecharatana, Committee Member — Date
Professor of Civil and Environmental Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Armin Bischof

Degree: Master of Science

Date: May 2003

Undergraduate and Graduate Education:

- Master of Science in Civil Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2003

- Diplom-Ingenieur (FH) Bauingenieurwesen,
  University of Applied Science Biberach, Biberach (Riss), Germany, 2002

Major: **Civil Engineering**

Presentations and Publications:

Armin Bischof,
"Analysis of multi-planes of glass loaded by climatic and mechanical loads with Finite Element Method,"
Diploma Thesis at the University of Applied Science Biberach, Biberach (Riss), Germany, June 2002

Armin Bischof,
"Analysis of multi-planes of glass loaded by climatic and mechanical loads with Finite Element Method,"
Presentation at the annual SOFISTIK user seminar 2002, Nuremberg, Germany, May 2002

To my family and friends

ACKNOWLEDGEMENT

I would like to express my deepest appreciation to Dr. William R. Spillers, who not only served as my thesis advisor, providing valuable and countless resources, insight, and intuition, but also constantly gave my support, encouragement, and reassurance. Special thanks are given to Dr. Ala Saadeghvaziri and Dr. Methi Wecharatana for actively participating in my committee.

Many of my fellow graduate students in the Department of Civil and Environmental Engineering are deserving of recognition for their support. Special thanks goes to my fellow graduate students in the Master of Science Program in Civil Engineering (major: structure engineering) for all their help, support, and advice during my graduate studies at New Jersey Institute of Technology. Special thanks goes also to the Fulbright Commission, who enabled my studies at NJIT by their financial and administrative support.

TABLE OF CONTENTS

Chapter		Page
1	INTRODUCTION TO TENSILE STRUCTURES	1
1.1	Tensile Structures	1
1.2	Phases of the Tensile Structures	2
1.3	A Few Words about Cable Nets	4
1.4	Fabric Structures	6
1.5	Materials and Elements of Construction	7
1.6	The Main Principles of the Analysis of Fabric Structures	9
1.6.1	The Analysis of Fabric Structure Must be Nonlinear	9
1.6.2	Shape Finding (1st Step of the Design Analysis)	10
1.6.3	Static Analysis of the Structure (2nd Step of the Design Analysis)	11
1.6.4	Patterning (3rd Step of Design Analysis)	13
2	THE GRID METHOD	14
2.1	Principles and Properties of the Grid Method	14
2.2	Shape Finding for One Fabric Structure Using the Grid Method	14
3	GRID METHOD FOR FABRIC STRUCTURE WITH FIXED BOUNDARIES	19
3.1	Comparison of Two Different Mathematical Methods	19
3.2	Grid Method Using Jacobi's Method (Existing Fortran Code)	20
3.2.1	The Principles of Jacobi's Method	20
3.2.2	Fortran Code of Program HLAY.FOR	21
3.2.3	Description of the Fortran Code HLAY.FOR	21
3.2.4	Computation of One Example Problem Using Code HLAY.FOR	24
3.2.5	Results of the Example Computation Using HLAY.FOR	26
3.3	Grid Method Using Gaussian Elimination (new Java code)	28
3.3.1	The Principles of the Gaussian Elimination	28
3.3.2	Description of the Java Code shapeApp.java	29

TABLE OF CONTENTS
(Continued)

Chapter		Page
	3.3.3 Computation of One Example Problem Using Code shapeApp.java	33
	3.3.4 Results of the Example Computation Using shapeApp.java	36
3.4	Refinement of the Java Code shapeApp.java	39
	3.4.1 Computation of the Example Problem Using the Refined Java Code	39
	3.4.2 Results of the Computation Using the Refined Java Code	44
3.5	Comparison of the Three Methods (Codes), Results and Conclusions	47
4	GRID-METHOD FOR FABRIC STRUCTURES WITH EDGE CABLES	49
4.1	The Grid for a Fabric Structure with Edge Cables	49
4.2	Definition of the Basic Grid	50
4.3	Plane Geometry of the Edge Cable	51
	4.3.1 Loading of the Edge Cable	51
	4.3.2 Define the Geometry of the Edge Cable	54
4.4	Computation of Intersection Between Edge Cable and Basic Grid	56
	4.4.1 Java Code to Compute the (Plane) Boundary Coordinates	58
	4.4.2 Java Code edgecableApp.java	59
	4.4.3 Input of the Java Code edgecableApp.java	59
	4.4.4 Output of the Java Code edgecableApp.java	60
4.5	Computation of the Force in the Boundary Members of the 2-D Grid	61
4.6	Java Code shapeedgeApp.java for Fabric Structure with Edge Cables	66
4.7	Example Computation of a Fabric Structure with Edge Cable	69
	4.7.1 Computation of the Boundary Node Coordinates	70
	4.7.2 Definition of Node and Member Numbers for the Grid	75
	4.7.3 Computation of the Forces in the Member of the Plane Grid	76
	4.7.4 Input Section of the Example Computation	80

TABLE OF CONTENTS
(Continued)

Chapter **Page**

4.7.5 Output Section of the Example Computation 82

5 NONLINEAR STATIC ANALYSIS OF FABRIC STRUCTURES..... 85

5.1 The Mathematical Methods to Perform a Nonlinear Analysis..... 85

5.1.1 The Principles of the Euler's Method (Incremental Solution)..... 85

5.1.2 The Principles of the Newton's Method (Iterative Solution)..... 86

5.2 Nonlinear Static Analysis of Fabric Structures 87

5.3 Implementation of the Nonlinear Static Analysis 90

5.4 Stiffness Properties of the FEM-Element of the Membrane 90

APPENDIX A FORTRAN CODE OF PROGRAM HLAY.FOR..... 94

APPENDIX B JAVA CODE OF PROGRAM edgecableApp.java 100

REFERENCES..... 106

LIST OF FIGURES

Figure		Page
1.1	Classes of tensile structures	1
1.2	Cable net clamp	4
1.3	Edge cable clamps	5
1.4	Mesh geometry of the cable net	5
1.5	Different types of fabric structures	7
1.6	Air-supported fabric structures	7
1.7	Tension fabric structures	7
1.8	Cable system without deflection	9
1.9	Cable system with deflection	9
2.1	Horizontal grid of the fabric structure with fixed boundaries	15
2.2	Arch geometry of the fabric structure with fixed boundaries	16
2.3	3-D grid of the fabric structure with fixed boundaries (one quarter)	17
3.1	Sketch of typical member	21
3.2	Free node and its equilibrium	31
3.3	3-D graphic of the fabric structure with fixed boundaries	47
4.1	Sketch of a basic grid including edge cables	49
4.2	Sketch of the grid of a fabric structure with edge cables	50
4.3	Uniform forces acting on edge cables	52
4.4	Uniform forces acting on edge cables	52
4.5	Static of cables under uniform load	53
4.6	Positive and negative max. displacements	54
4.7	Convert support forces into forces in x- and y-direction	62
4.8	Sketch of the boundary members and node of one edge cable	64
4.9	Grid of fabric structure with edge cable for example computation	69
4.10	Define node numbers for grid of fabric structure	75

LIST OF FIGURES
(Continued)

Figure		Page
4.11	Define member numbers for grid of fabric structure	75
4.12	Shape of fabric structure with edge cable	84
5.1	Typical member of structure with nonlinear behavior	88
5.2	Nonlinear effects of the geometrical stiffness matrix	88
5.3	Linear effects of the elastic stiffness matrix	89
5.4	Plane stress/strain finite element	91
5.5	Out-of-plane rotations of plane finite element	92

CHAPTER 1

INTRODUCTION TO TENSILE STRUCTURES

1.1 Tensile Structures

Tensile structures can be characterized as structures that carry the loads mainly through tensile members, which cannot take compression or bending moments. The main load-carrying members of tensile structures transmit the applied loads to the supporting structures by direct tensile stress without any flexure or compression. These supporting members are foundations or structures that transmit the forces to foundations. The shear rigidity, flexural rigidity and buckling resistance of the main load-carrying members of the tensile structures are negligible. There are two classes of tensile structures. The first one is the cable structures, which are comprised of uniaxially stressed members. The second one is the fabric structures, which are comprised of biaxial stressed members.

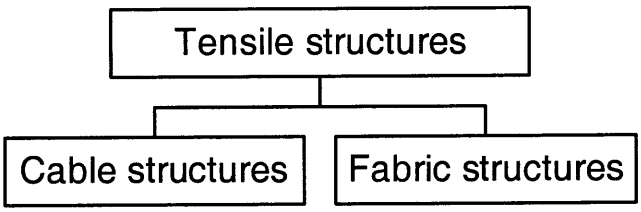

Figure 1.1 Classes of tensile structures

This report deals with the fabric structures; however, most of the principles for fabric structures are also valid for cable structures. The differences between fabric structures and cable structures are shown in chapter one.

Tensile structures are highly susceptible to large motion due to concentrated loads and dynamic effects. The reduced stiffness characteristics are the main reason for this behavior. The mechanics of tensile structures works through great deflections of the structures until it is in equilibrium. Tensile structures have therefore a highly nonlinear static behavior, regardless of the linearity of the material behavior. The applied forces are prestressed forces and in-service forces. Prestressed forces are those forces, which act on the unloaded structure and create the geometry of the tensile members. They stabilize the structure and provide stiffness for further deflections. Examples of prestressed forces are edge loads, self-weight or pressure. In-service loads are all variable live loads, which the structure encounters during its service life. These loads might be static and dynamic. The principle of superposition is not allowed in the analysis of tensile structures, because of the nonlinear behavior of tensile structures.

1.2 Phases of the Tensile Structures

The physical behavior of a tensile structure during its lifetime can be divided in three primary phases. Phase one is the deployment phase. This is the stage, when the tensile members are in a state of incipient stresses after it is unfolded from its compact configuration. Phase two is the prestressing phase. This is the stage, when the erected tensile structure is prestressed and in its equilibrium position under the action of dead load. The tensile structure in equilibrium has its final geometry (shape) for the load combination without live loads in phase two. Phase three is the in-service phase. This is the stage, when the fully prestressed system is subjected to variable live loads (dynamic and static) during its service life.

Deployment phase

The cable and/or fabric of the tensile structure are unfolded with external forces during the deployment phase. Inertial forces counterbalance these external forces. The structure is stress-free during this dynamic process. The expansion of the tensile structure is highly nonlinear and that analysis of this phase shows no unique solution.

Prestressing phase

Prestressing is the second phase in the erection of tensile structure. The structure undergoes displacement from its initial state into a static equilibrium state. The static equilibrium state is dictated by the prestressed forced and the geometry of the structure members. The prestressing phase is, because of the displacement, a nonlinear problem and has a unique solution.

Geometry

The geometry of the tensile structure must be defined to enable a stress analysis of this structure. There are two options: The first option is to use the initial geometry and the second option is to use the prestressed geometry of the tensile structure. The first option causes more difficulties in stress analysis, because the prestressed geometry must be computed first by incremental or iterative methods before the stress analysis can be performed. However, the first option has advantage from the fabrication point of view. The second option is much better for the stress analysis, but the initial shape of the structure members are much more difficult to fabricate.
The Gaussian curvature G describes the geometry of the surface of the fabric structure. G is nothing else than the inverse product of the two principle radii of the fabric structure surface. There are three principle cases:

- The center of the curvature of the two principle directions is on the same side of the surface, if $G < 0$. This geometry is called synclastic, elliptic, or positive-gaussian.

- The center of the curvature of the two principle directions is on the two opposite sides of the surface, if G > 0. This geometry is called anticlastic, hyperbolic, saddlelike, or negative-gaussian.
- One or more radius of curvature is infinite, if G = 0. This geometry is called developable, parabolic, or zero-gaussian.

The elliptic (positive gaussian) tensile structures require external loads to keep their prestressed position. The external loads could be dead load, pressure and so on. The hyperbolic (negative gaussian) tensile structures don't require external loads, because the opposing curvature provides counterstressing. The counterstressing is provided by prestressed configurations of cables or membranes curved in the opposition to each other. The result is the geometry of the tensile structure that is independent from external loads. The zero-gaussian surfaces are the only tensile structure, which geometry can be obtained from a fabricated flat surface, without straining the tensile members.

Static and kinematic

The static analysis must be nonlinear and can be performed by iterative or incremental methods. The static problem of the prestressed phase is to find the initial geometry and stresses of the tensile structure, which is only loaded by the dead load (no live load). The dead loads include constant pressures and prestressing. The solution of this static analysis is the initial and prestressed tensile structure. The strains will be small in most cases, but there will be large relative rotations. These large displacements occur during the prestressing phase, because of the extreme flexibility of tensile structures (no bending resistance and small cross sections).

The two definitions of strain that can be used are engineering strain and true strain. The true strain is a more accurate representation of the kinematics. However, the engineering strain can be used mostly, because of the small strains that are usually assumed. The engineering strain is the change in differential length divided by the original length, while the true strain is the change of integrated differential length divided by the square of the original length.

In-service phase

The in-service phase is the third phase in the behavior of tensile structures. This is the phase, where the live static and dynamic live loads (snow load, wind load and so on) occur during the service live of the structure. The live loads create additional stresses and deflections in the tensile structures. The static and dynamic behavior of the tensile structure during this phase is nonlinear and the analysis has to be performed in consideration of this fact. The computed initial geometry and stresses at the prestressed phase are used as the basis for the stress calculations during the in-service phase. The mechanical behavior of tensile structures is not always statically determinate. Therefore kinematic and constitutive equations must be considered as well during the design analysis. Slack cables and the wrinkling of membranes should be avoided, or if unavoidable, their extent and effect on global stability of the tensile structure must be

predicted and restricted. Flutter and dynamic wrinkling of tensile structures especially under dynamic loads (for example wind loads) are one important aspect in the design analysis of tensile structures and they must be restricted.

1.3 A Few Words about Cable Nets

Cable nets are not the subjects of the thesis. However, cable nets are related with fabric structures and these two kinds of structures have a lot in common. The static and dynamic behavior is similar. Many principles and rules in the design of cable nets can also be used for the design process of cable nets. Because of this fact, this section about cable nets in included in this thesis. This section is about the properties of cable nets and shows important points that must be considered in the design process. The goal of this section is the draw a connection between cable nets and fabric structures. The rest of the thesis deals with fabric structures.

The members of cable nets are cables that can only take tension stresses. These members have small cross section and their bending and shear resistances are negligible (is practice equal to zero). The connection between the cables is made by cable net clamps. The connection between the edge cables and the interior cables of the cable net are made by edge cable clamps. Forces that are parallel to the edge cable must be transmitted via friction from the edge cable clamps to the edge cable. No other transmission of forces between edge cable clamps and edge cable is allowed. Therefore a certain clamp length and prestressed clamp force is required to achieve the friction that is needed.

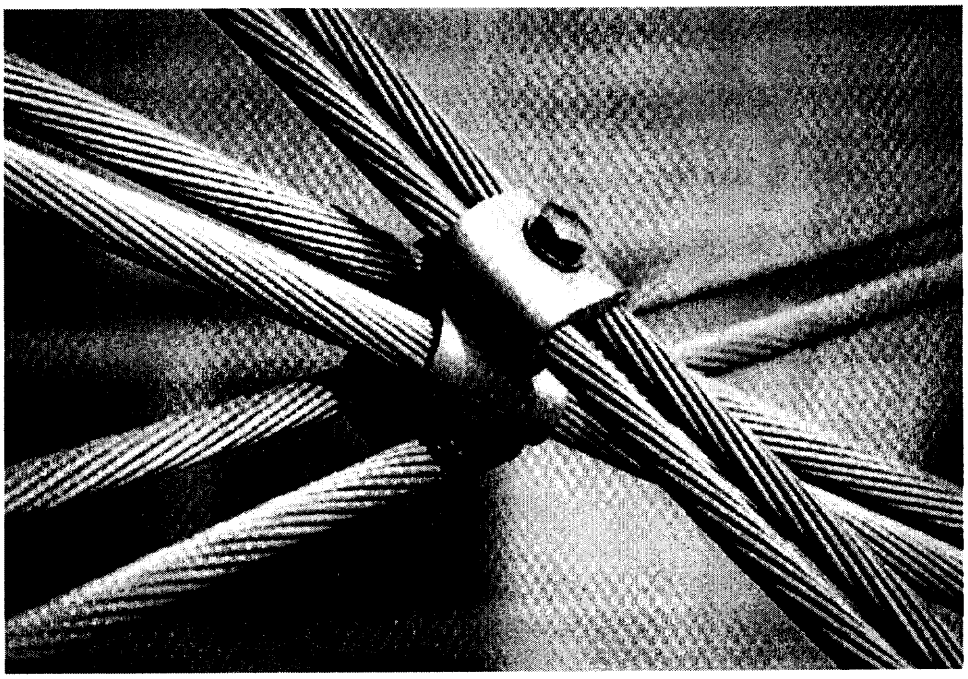

Figure 1.2 Cable net clamp
(Reference 20: University Stuttgart)

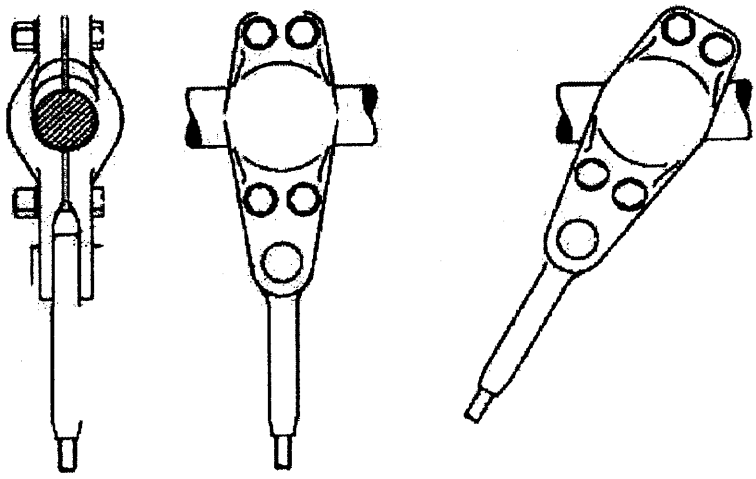

Figure 1.3 Edge cable clamps
(Reference 20: University Stuttgart)

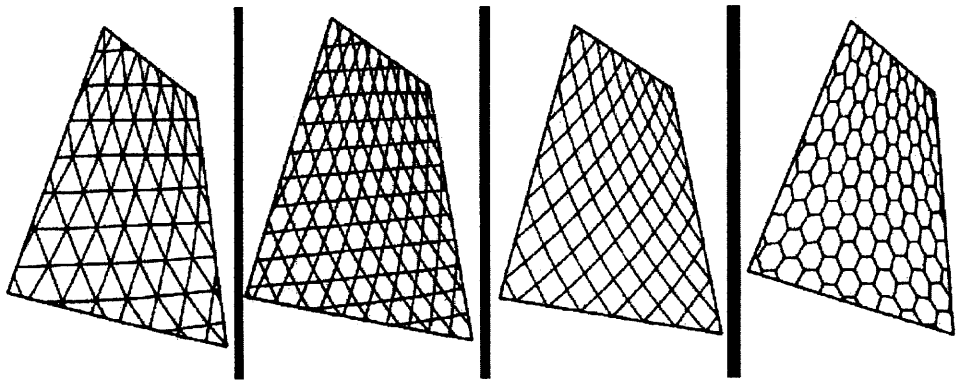

Figure 1.4 Mesh geometry of the cable net
(Reference 20: Univerity Stuttgart)

The most common mesh geometry for cable nets are meshes with four corners. The meshes of these cable nets become often quadratic, if the cable net is transformed into a plane. However, all kinds of mesh geometries are possible for cable nets. There are more points that must be considered in order to choose the best mesh geometry for a certain cable net. First, the requirement for the coverage between the cables must be considered. Other points are the kinematics of the cable net, the equality of the mesh geometry and aspects for the erection and construction phase of the cable net.

Almost all cable nets are mechanical prestressed. This means that most of them have opposing curvature, which provides counterstressing. The Gaussian curvature of these cable nets are $G < 0$ and can be described as hyperbolic, anticlastic or saddlelike.

1.4 Fabric Structures

Fabric structures make it possible to span large areas without any columns in this area. The main advantage of fabric structures is the very low weight of these structures. The structure that is holding the fabric structure can also be design very thin and economically, because of the low weight from the fabric structure. The main use of fabric structures is where a roof must be spanned over a large area with few or no supports between. This are typically sports stadiums, but the architects are more and more interested in using this structure in shopping malls, harbors, airports and so on. The reason for this development is the very interesting view of fabric structures which the architects use for their buildings.

Air-supported structures and tension structures are the two different types of fabric structures. Air-supported structures use a membrane, supported by air pressure, to act as the walls and/or roof of the structure. The stability of tension structures is provided by counterstressing of the membrane or the supporting cables. The two main curvatures of tension structures are on the two opposing sides of the membrane surface and prestressing of this curved membrane creates the counterstressing and the stable fabric structure.

The support forces of air-supported structures are mainly provided by different air pressure inside the structure in comparison with the air pressure outside the structure. The inside of the structure is pressurized like a balloon. This pressure differential is not greater than that of ordinary barometric fluctuations. Therefore the difference of air pressure between inside and outside the structure is not uncomfortable for the occupants of the structure. As stated above, air pressure is used to support and stabilize air-supported structures. The air that is under pressure exerts a uniform force in all directions. This air-pressure is used to support the membrane. These membranes are called hybrid membrane, if the structure includes cables. These cables are not used to support the air-supported structure, however they are necessary to prevent uplifting and hold the membrane down. The hybrid membrane transfers the stresses from the fabric to the cables and the cable transfers these stresses to the support points, which take these uplifting forces. The main advantage of air-supported structures is the considerable low costs in comparison to with conventional structures. The lower construction and supporting structures cost are the main reasons for the efficiency of these structures. A huge disadvantage of air-supported structures is local deflection because of accumulated loads at one point. These local deflections can be huge and the initial geometry of the air-supported structure might get lost.

Tension structures consists of membranes and sometimes supporting cables that are in tension. The prestressed tension is provided by counterstressing of the membranes and the supporting cables. Every part of the membranes and cables must be prestressed with tension stresses, because these members can only take tension stresses and it is the only way to provide the stability of these structures. The main disadvantage of tension structures is that the design can only performed with the help of computer programs, because the shape of these structures is governed by complex differential equations. The basic types of tension structures are cable domes, mast supported, arch supported, radial tent and saddle roof.

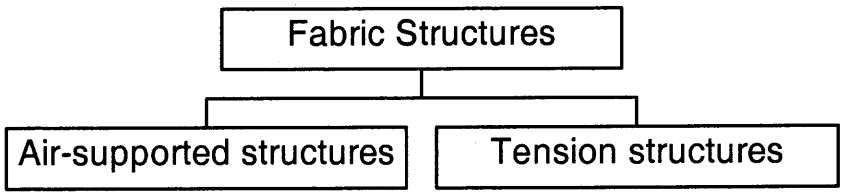

Figure 1.5 Different types of fabric structures

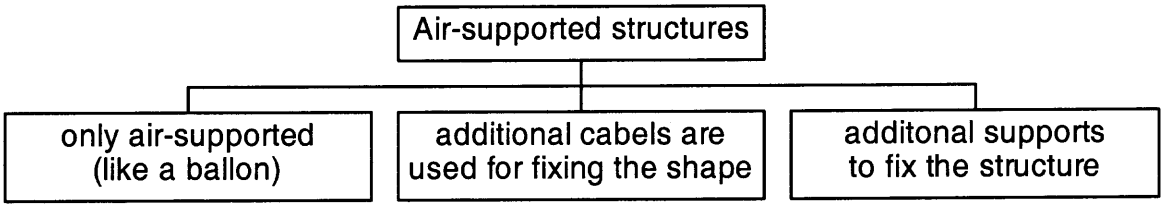

Figure 1.6 Air-supported fabric structures

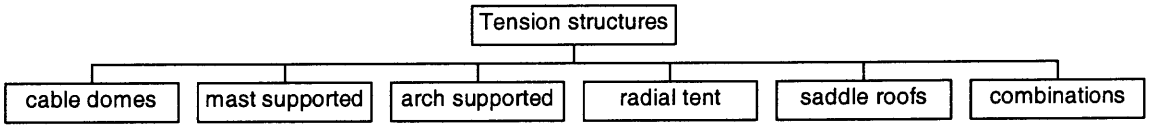

Figure 1.7 Tension fabric structures

1.5 Materials and Elements of Construction

Cables

The cables are usually made from high-strength steel, whose properties are given by the AISI (American and Iron Steel Institute). The main advantage of steel is its availability, the low cost and long life. Another possibility is to use the material Kevlar or fiberglass, which are stronger and stiffer. The disadvantage of these materials are that they are much more expensive and they degrade when exposed to ultraviolet light. The cables that are used for the construction of tensile structures should have the following important properties:

- small cross sections
- low weight
- long fatigue life
- resistance to corrosion and abrasion

- high flexibility
- good stretch and rotational behavior

Cables act as axial elements. In cable materials the linear stress-strain relationship holds only for a portion of the usable strength. The proportional relationship doesn't hold beyond a certain elastic limit and the material behavior becomes nonlinear at this point. Cables can show a huge plastic behavior and with large ductility. They must be prespanned during the production process in order to prevent large plastic deformations of the cable during the construction and in-service time.

Fabric

Different fabrics are used to build fabric structures. Today permanent fabric structures are usually built out of entirely synthetic fabrics. The following structural materials are available:

- Vinyl-coated polyester fabric or scrim
- Vinyl-coated polyester scrim laminate with or without Teflon film
- Vinyl-coated fiberglass fabric
- Vinyl-coated Kevlar fabric
- Teflon-coated fiberglass fabric
- Silicon-coated fiberglass fabric

The most used fabrics for membranes are fiberglass and polyester. Fiberglass is strong, durable and the better material, but it is expensive and deteriorates when exposed to moisture. Polyester is less expensive but it is not as strong and degrades when exposed to sunlight. The fabrics are coated by silicon, teflon or vinyl. The properties of fabrics are typically within the following range:

Strip tensile:	20 to 1000 lb/in (35 to 1750 N/cm)
Grap tensile:	Typically 30 percent higher than strip tensile
Tear strength:	1 to 200 lb/in (2 to 350 N/cm)
Adhesion strength:	2 to 40 lb/in^2 (0,1 to 2 N/mm^2)

It's not possible to produce and ship the fabrics for large structures in one piece. Therefore there are produced in sheets and jointed together at the building site. Mostly the standard lap joint is used to make this connection between the single fabric sheets. The two pieces of fabric are overlapped by three inches and Teflon FEP is inserted between them. The joint is then heat welded together. This joint is stronger that the fabric and completely water and airtight. These fabrics have other qualities, which are important for well-design buildings. These additional properties include low heat absorbance, translucency and reflectivity. The low heat absorbance ensures a comfortable climate in the building and cost the cost of air conditioning can be reduced. The translucency allows a lot of sunlight to come through the fabric and creates a sense of outdoors inside the building. The reflectivity reduces the artificial light that is needed, because of the reflection of the light. It helps therefore to reduce energy costs.

1.6 The Main Principles of the Analysis of Fabric Structures

The following section shows the main steps for the analysis and design of fabric structures. The section 1.4.1 is a short description of why a non-linear analysis of these structures is mandatory. The following is a presentation of the three main steps, which are necessary for the complete analysis of fabric structures. More accurate explanation of the analysis of these structures can be found in the later chapters.

1.6.1 The Analysis of Fabric Structures Must be Nonlinear

First Reason:

Linear Elastic Theory approximates the length change of a bar by the dot product of the direction vector and the displacement. But in this situation, you can see from figures 1a and 1b, that they are perpendicular to each other therefore dot product = 0. This would mean that the bar did not change length, which from observation is untrue. It is therefore necessary to use nonlinear analysis.

Second Reason:

The cable in figure 1a can't take the load to the supports without a deflection of the cable. The vertical part of the support forces is independent of the deflection of the cable. But the horizontal part of the support forces is dependent from the deflection of the cable. If the deflection of the cable is very small, the horizontal support forces will become huge. The horizontal support force will on the other side become very small, if we have a large deflection of the cable. Therefore it is necessary to use nonlinear analysis, to get the right support and cable forces.

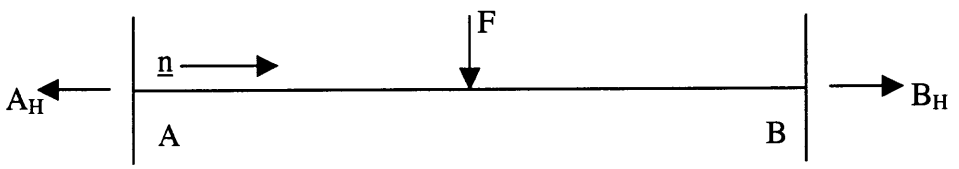

Figure 1.8 Cable system without deflection

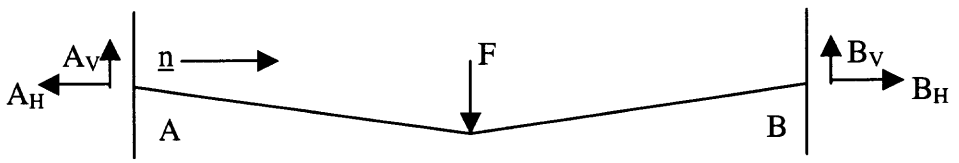

Figure 1.9 Cable system with deflection

Comment to Figure 1.8:

Cable can't carry shear-forces. Therefore the only force in the cable is an axial force. This is the reason why the structure can't take the vertical load F, if there is no deflection and the axis of the cable stays horizontal.

Comment to Figure 1.9:

Though the deflection (change of the cable length), the axis of the cable stays no longer horizontal. Therefore the cable now carries horizontal and vertical force components. The structure can carry the vertical load F, after the deflection. The horizontal support forces are dependent from amount of the length change of the cable, because of the vertical support forces that are fixed.

1.6.2 Shape Finding (1st Step of the Design Analysis)

The shape of the fabric structure must be known, before the analysis of this structure can be started. Therefore the shape must be defined, which can be done with the calculation of many coordinates (points) of the structure. These points should be part of a grid or something like that. The process of shape finding starts after the frame of the fabric structure is defined. The fixed points of the frame and the equations of equilibrium are the basis to calculate the coordinates of the other points, which define the shape of the fabric structure.

In former times, physical models have been used for the design of fabric structure and cable nets. It is known that these small-scale models are not accurate enough for a good design of these structures. The computer enables us today, to calculate the coordinates of a fabric structure (shape finding) much more accurately, which is the foundation of a better and more exact design of these structures. A few computer-based approaches are introduced in the following sections.

Deformed Shape

This method works through applying loads and stretch the fabric (elastic sheet) to a deformed fabric, which is the shape that was desired. This practical process can be calculated with every non-linear structural analysis program. The result is a shape that is in equilibrium and fits on the frame of the fabric structure. The problem in this method is that the deformation of the fabric can cause great stress concentrations, which are not desired in this structure. Additionally most fabric structures cannot take large strains without tearing.

Force Density Method

The force density method works with the fact that if ratio of the bar force to its length (force density) in a cable net is held constant, the associate geometry of this structure can be found by solving a system of linear equations. This method also works with fabric structures. The equations for the solution of the problem and finding the shape of the structure are:

- x-component $\Rightarrow F_{ix} = \frac{F_i}{L_i}(X_A - X_C)$ (Equation 1.1)
- y-component $\Rightarrow F_{iy} = \frac{F_i}{L_i}(Y_A - Y_C)$ (Equation 1.2)
- z-component $\Rightarrow F_{iz} = \frac{F_i}{L_i}(Z_A - Z_C)$ (Equation 1.3)

These equations show clearly that we have only linear equations, if we know the force density Fi / Li of the particular part of the structure. As the length of the considered part of the structure will vary during the process of shape finding, this problem is going to become more complicated. This fact leads to nonlinear equations that must be solved to find the shape with the force density method.

Grid Method

The grid method is a very simple way to find the shape of a fabric or cable structure. It based on the fact, that if horizontal equilibrium is satisfied over a horizontal plane grid, the vertical equilibrium can be used to compute the level of the structure at its grid nodes. The shape of the structure can be found by combining the calculated coordinates of the grid, which is now spatial.

The first step of the grid method is to find the member forces, which must be assumed or determined so that the grid (cable net or finite element system) in the horizontal plane is in equilibrium. These member forces and also the levels of the fixed supports or arches must be defined or computed. Once these points are known, it is possible to determine all the other points by using the grid method. It should be noted that only the levels of nodes within the determined grid can be calculated with this method. Therefore it is important to choose the right plane grid when starting the shape finding process.

Smoothing

Smoothing can be used when a shape is given or determined. It is a method to optimize and improve the shape of a fabric structure or cable net. A non-linear computer program analyzes the given shape and smooth out the forces and stresses. The aim of this process is to get a structure with almost uniform stresses.

1.6.3 Static Analysis of the Structure (2nd Step of the Design Analysis)

The static and dynamic behavior of fabric structures is nonlinear. This nonlinear behavior must be considered in the static analysis. Therefore the static analysis must be nonlinear. Membranes can be considered as two-dimensional spaces. However, loading occurs in a three-dimensional world, therefore, if components of load transverse to the membrane are to be supported, the two-dimensional membrane must be curved. The behavior of membrane structure is therefore inherently nonlinear. Numerical methods can be used for

the static and dynamic analysis of fabric structures. Iterative solutions, incremental solution or a combination of iterative and incremental solutions can be used to perform these numerical methods. The standard method for the iterative solution is Newton's method and the standard method for the incremental solution is the Euler's method. These two mathematical methods will be described more accurately in one of the following chapters. The basis for the static analysis of fabric structures is a set of differential equations that describes the equilibrium conditions of the nonlinear system of the fabric structure. The node displacement method is used for the definition of the nonlinear equilibrium conditions and the computation of the fabric structure. The following equations describe the equilibrium conditions using the node displacement method.

Linear structural analysis:

$$N^T * F = P \qquad \text{(Equation 1.4)}$$

$$\Rightarrow \qquad K_E * \delta = P \qquad \text{(Equation 1.5)}$$

Nonlinear structural analysis:

$$dN^T * F + N^T * dF = dP \qquad \text{(Equation 1.6)}$$

$$\Rightarrow \qquad K_E * \delta + K_G * \delta = P \qquad \text{(Equation 1.7)}$$

=> where:

K_E = elastic stiffness matrix (linear effects)
K_G = geometrical stiffness matrix (nonlinear effects)
P = node forces
F = member forces
N = equilibrium "operator" (unit vectors)
δ = node displacement

The term $K_E*\delta$ reflects the change of the member forces with the matrix N fixed, which is simply linear elastic analysis. These equations are just derived from the relation between the member forces (F) and the "equilibrium operator" (N), which defines the linear elastic theory of linear structures. The term $K_G*\delta$ computes the nonlinear effects of the structure. The simple superposition of these two terms results in the solution for the static analysis of a nonlinear static system.

The best way to perform a static analysis of a fabric structure is to model its membranes using the finite element method. Membrane finite elements must be modeled in order to create a basic system for the finite element analysis. This basic system of finite elements is the foundation for the nonlinear membrane finite element equation, which describes the equilibrium of the membrane structure. The solution of this set of nonlinear equations is the result of the nonlinear static analysis of the fabric structure. Effective computer programs are needed for this nonlinear static analysis.

1.6.4 Patterning (3rd Step of the Design Analysis)

The last step in the design of fabric structures is patterning. The basic problem of patterning is the construction of a curved surface out of flat pieces of material. Additionally it must be considered that the single pieces of fabric change their length when they are prestressed. This means that, for example, a fabric must be produced "too short" so that it will have the right length in the prestressed structure. The process of patterning implies finding the right collection of pieces of fabric, which create together the desired shape of the structure and to find the right area of these pieces of fabric under consideration that the area will change in the prestressed fabric structure. Patterning is a complex nonlinear process and is performed by computer codes.

The results of the shape finding and static analysis (steps one and two of the design analysis) are the basis for this computation. The curved surface of the membrane must be divided in triangles. The shape of the fabric structure is now modeled as a collection of flat triangles and the coordinates of the edge nodes of these triangles are the input for the patterning analysis. The next step is to select "strips" of triangles over the structure. These strips must be selected in this way that enables an easy production of these membranes. The strips are the single spots of membrane that are later connected with each other in order to get the membrane for the complete fabric structure. The principle behind this process is that strips of single flat triangles can be deformed into a flat sheet without stretching.

CHAPTER 2

THE GRID METHOD

2.1 Principles and Properties of the Grid Method

The grid method is a very simple way to determine the shape of a fabric structure. The basis of this method is a horizontal plane grid. This grid must cover the area of the membrane of the fabric structure and all fixed points must match a grid node. The plane basic grid must be prestressed with horizontal forces in such a way that the grid is in horizontal equilibrium. No vertical forces are allowed in this stage. This prestressed plane grid is the basis for the next steps of the grid method. The grid, which was only defined in a 2D-system must now be transferred into a 3D-system. This can be achieved by defining the level (3rd spacial coordinate of the grid nodes) of all fixed points of the fabric structure on this grid. The result is a 3D-system (spacial grid) with vertical forces. These vertical forces are not in equilibrium. The equilibrium equations can be formulated to depend on the grid node levels. The resulting system of equilibrium equation makes it possible to compute the grid node levels of the spatial grid (3D-system) that is in complete (horizontal and vertical) equilibrium. The results of this analysis are 3D-coordinates for all grid nodes, which create together the shape of the fabric structure. The follwing list describes the single steps of the grid method:

(1) definition of a horizontal plane grid, which covers the area of the fabric structure and all fixed point matches with one grid node
(2) prestress of all grid members in such a way that the grid is in horizontal equilbrium (no vertical forces are allowed in this stage)
(3) definition of the level (vertical coordinate of the grid nodes) of all fixed point to make the grid spatial
(4) grid is not in vertical equilibrium
(5) set up vertical equilibrium equations which depend on the vertical coordinates of the grid nodes
(6) use the set of equilibrium equations to compute the vertical coordinates of the grid, which is in horizontal and vertical equilibrium
(7) combine the computed vertical coordinates of the grid nodes with the horizontal coordinates of the grid nodes
(8) the 3D grid nodes define together the shape of the fabric structure

2.2 Shape Finding Example for One Fabric Structure Using Grid Method

An arch supported tension structure is used for the example problem. All boundary nodes are fixed points (supports), whose 3D-coordinates are known. Additionally the nodes of the grid that are on the two arches are fixed nodes with known 3D-coordinates as well. The structure is 40 feet long and 20 feet wide. The grid that is used to find the shape of the fabric structure has 6 * 6 fields.

Sketch of the grid and the fabric structure:

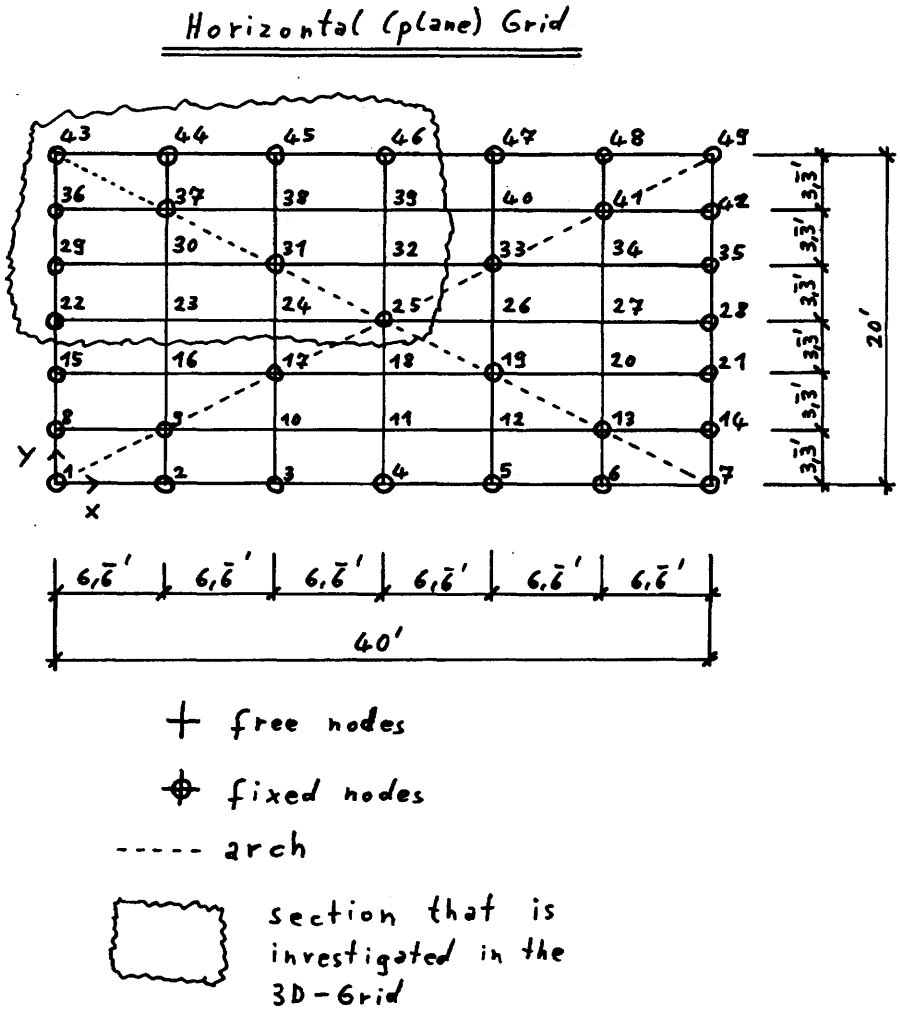

Figure 2.1 Horizontal grid of the fabric structure with fixed boundaries

All edge nodes are support nodes at the level zero. In other words, all edge node are fixed point with $z_i = 0$. Therefore the following nodes have a defined level and the nodes are fixed:

$$Z_1 = Z_2 = Z_3 = Z_4 = Z_5 = Z_6 = Z_7 = 0$$
$$Z_{43} = Z_{44} = Z_{45} = Z_{46} = Z_{47} = Z_{48} = Z_{49} = 0$$
$$Z_8 = Z_{15} = Z_{22} = Z_{29} = Z_{36} = 0$$
$$Z_{14} = Z_{21} = Z_{28} = Z_{35} = Z_{42} = 0$$

The fabric structure is supported by two arches. This first arch goes through the nodes 7, 13, 19, 25, 31, 37 and 43. The second arch goes though the nodes 1, 9, 17, 25, 33, 41 and 43. These nodes are fixed points as well.

The following sketch shows the geometry of the first arch (the geometry of the second arch is the same):

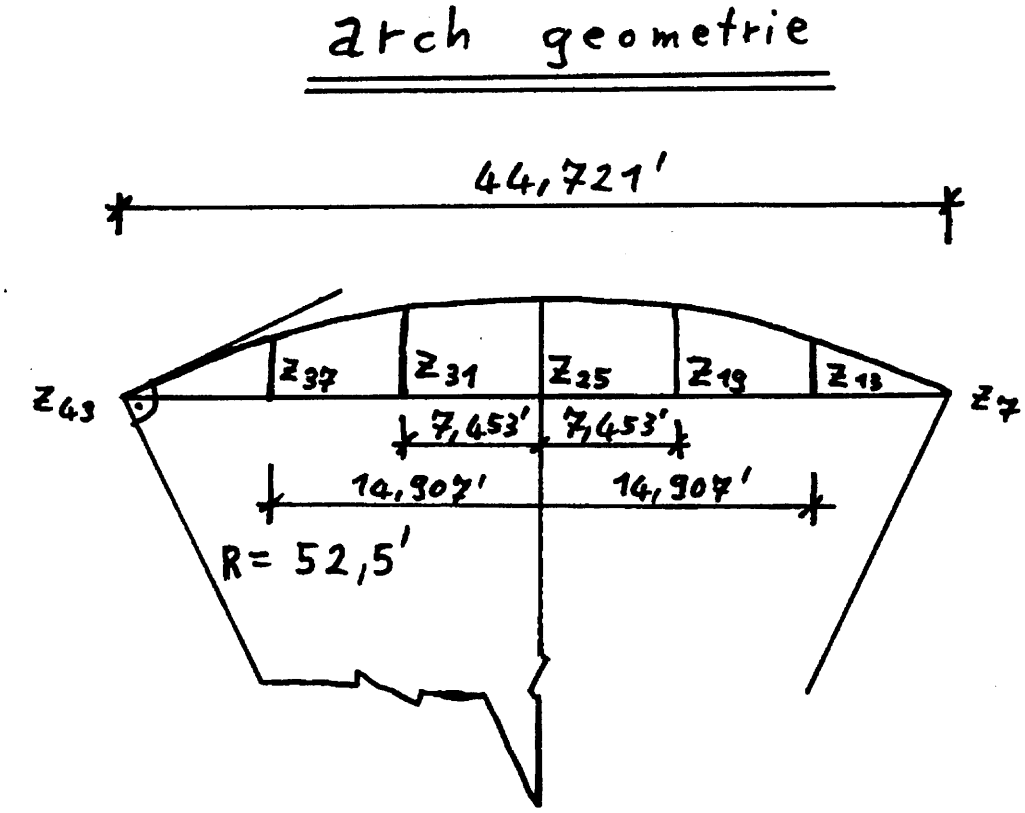

Figure 2.2 Arch geometry of the fabric structure with fixed boundaries

The nodes of the arch are fixed and they have the following levels:

$Z_9 = Z_{13} = Z_{37} = Z_{41} = 2{,}839$ feet

$Z_{17} = Z_{19} = Z_{31} = Z_{33} = 4{,}468$ feet

$Z_{25} = 5$ feet

The fabric structure and the grid are in both horizontal directions symmetric. Therefore it is enough to compute the shape of the fabric structure only at one quarter of the grid. In this case the section between the nodes 22, 25, 46 and 43 is chosen to compute the shape of the structure. This section is marked in the sketch of the horizontal grid.

If this quarter of the grid is investigated and the level of this nodes are computed, all nodes coordinates of the grid are known because of the doubled symmetry of the fabric structure.

Sketch of the computed quarter of the grid:

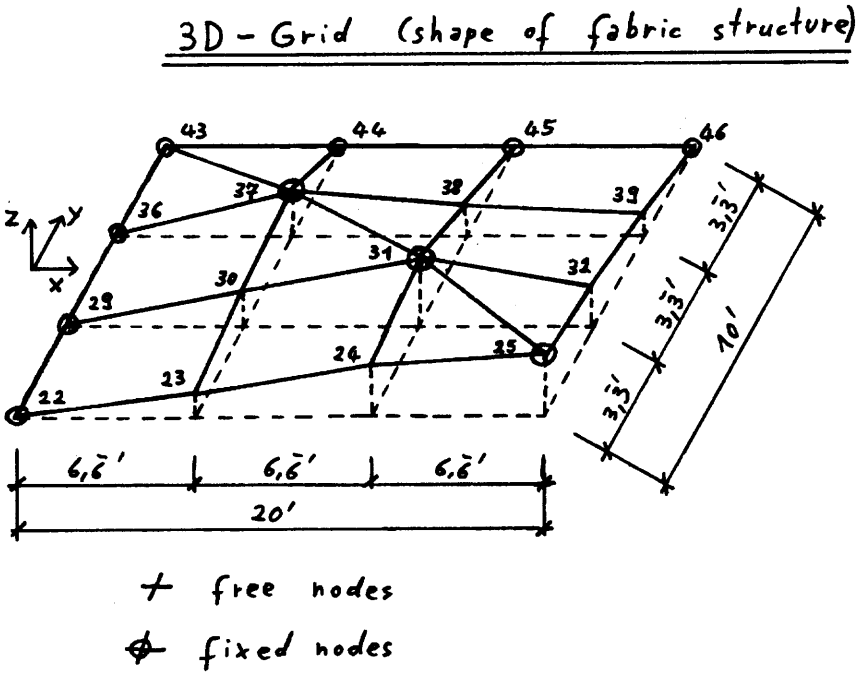

Figure 2.3 3-D grid of the fabric structure with fixed boundaries (one quarter)

Equilibrium equations of the free nodes:

The horizontal force must not be determined and considered, because this force is in every grid-line the same.

Node 23 = $$\frac{Z_{24}-Z_{23}}{6{,}66667}+2*\frac{Z_{30}-Z_{23}}{3{,}33333}-\frac{Z_{23}}{6{,}66667}=0$$

Node 24 = $$\frac{5-Z_{24}}{6{,}66667}+2*\frac{4{,}468-Z_{24}}{3{,}33333}+\frac{Z_{23}-Z_{24}}{6{,}66667}=0$$

Node 30 = $$\frac{4{,}468-Z_{30}}{6{,}66667}+\frac{2{,}839-Z_{30}}{3{,}33333}+\frac{Z_{23}-Z_{30}}{3{,}33333}-\frac{Z_{30}}{6{,}66667}=0$$

Node 32 = $$\frac{5-Z_{32}}{3{,}33333}+2*\frac{4{,}468-Z_{32}}{6{,}66667}+\frac{Z_{39}-Z_{32}}{3{,}33333}=0$$

Node 38 = $$\frac{4{,}468-Z_{38}}{3{,}33333}+\frac{2{,}839-Z_{38}}{6{,}66667}+\frac{Z_{39}-Z_{38}}{6{,}66667}-\frac{Z_{38}}{3{,}33333}=0$$

Node 39 = $$\frac{Z_{32}-Z_{39}}{3{,}33333}+2*\frac{Z_{38}-Z_{39}}{6{,}66667}-\frac{Z_{39}}{3{,}33333}=0$$

Linear Equation System:

The 6 node equilibrium equations create together a linear equation system with 6 equation and 6 unknown variables. This system can be solved easily and this system has a unique solution.

Solution:

The above linear equation system (node equilibrium equations) leads to the following solution:

$Z_{23} = 2{,}350$	$Z_{24} = 4{,}203$	$Z_{30} = 2{,}474$
$Z_{32} = 3{,}838$	$Z_{38} = 2{,}304$	$Z_{39} = 2{,}047$

Level (z-coordinates of the free nodes):

$Z_{23} = Z_{27} = 2{,}350$ feet

$Z_{24} = Z_{26} = 4{,}203$ feet

$Z_{30} = Z_{16} = Z_{20} = Z_{34} = 2{,}474$ feet

$Z_{32} = Z_{18} = 3{,}838$ feet

$Z_{38} = Z_{10} = Z_{12} = Z_{40} = 2{,}304$ feet

$Z_{39} = Z_{11} = 2{,}047$ feet

CHAPTER 3

GRID METHOD FOR FABRIC STRUCTURES WITH FIXED BOUNDARIES

3.1 Comparison of Two Different Mathematical Methods

This chapter is divided into three main sections:

- The first one is section 3.2 (Grid method using Jacobi's method), where the shape of the fabric structure is found using the Fortran Code HLAY.FOR. This code uses the Jabobi's method (interative mathematical method) to perform the grid method.
- The second one is section 3.3 (Grid method using Gaussian Elimination), where the shape of the fabric structure is found using the Java Code shapeApp.java. This code uses the Gaussian Elimination (solving an linear equation system) to perform the grid method.
- The third one is section 3.4 (Refinement of the code shapeApp.java), where a modified version of the Java Code shapeApp.java (section 3.2) is used. This code uses also Gaussian Elimination (solving an linear equation system) to perform the grid method.

The chapter 3 conludes with a grapic of the solution (shape of the computed fabric structure) and a comparison between the 3 computer codes (HLAY.FOR, shapeApp.java and improveshapeApp.java) in section 3.5.

The main purpose of chapter 3 is to compare the interative methods and the linear equation systems for the shape finding analysis of fabric structure and find its advantages and disadvantages. The start point is the already existing Fortran Code HLAY.FOR, which was developed by the Department of Civil Engineerin at NJIT. Section 3.2 describes who this code works and the proberties of this code are investigated. There are also explanations to the mathematical method that was used and finally there is an example computation using the Fortran Code HLAY.FOR. A fabric structure with fixed boundaries is used for the example computation and it is used for the comparison of the results of the different code for the shape finding analysis in chapter 3.

The basic idea of chapter 3 is to improve the analysis for shape finding of fabric structure and develop a more effiecient code for this computation. The interativ way of Jabobi's method need a lot of interation steps and therefore a lot of operations until the final result. Therefore, the code using the Jacobi's method takes a lot of compuation time and is not very efficient. The innovative idea is to improve the efficiency of this analysis by unsing a linear equation system. Chapter 3 shows that the Grid method can be defined and performed by a linear equation system. Solving this linear equation system can be done without iteration and this code needs therefore much less operations to get a result. The code development of this code is documented and the properties of this code are descripted in section 3.3. There are also explanations to the mathematical method that was used in this code. The program language Java has been choosen to write this code,

because of its much better capabilities to create 3-dimensional graphics of the shape of the fabric structure and neat formated outputs. Experiences form the past has shown that the Java language has the much better equipment than Fortran to perfom these tasks. The new devoloped Java code for the Grid mehtod is called shapeApp.java. Finally the same fabric structure with fixed boundaries (form section 3.2) is used to perform an example computation and compare the result of this analysis with the results of the other codes.

The Java code improvedshapeApp.java in section 3.4 is a refinement of the first developed Java code shapeApp.java. The goal of developing this additional code was to optimize the linear equations system, reduce the equations as far as possible and get more accurate results. The development and the proberties of the Java code are documented and describted in section 3.4. At the end of this section there is also an example computation of the fabric structure with fixed boundaries (from section 3.2) using the new code improvedshape.App. The results of this example analysis (shape finding) is used to compare the result of this code with the results of the previous example computations. Especially to compare the result of the analysis using the two new Java codes and find out if the refinement in the code improvedshapeApp.java have led to a better accurancy in comparison to the result of the code shapeApp.java.

3.2 Grid Method Using Jacobi's Method (Existing Fortran Code)

This section is considering the already existing Fortran code HLAY.FOR. This code was developed by the Department of Civil Engineering at New Jersey Institute of Technology, which is the start point of chapter 3. This section describes the code and its proberties. It also explans the mathematical principle (iteration using Jacobi's mehtod) on which this code is founded and how it works. Additionally an example computation for the shape finding of a fabric structure with fixed boundaries is perfomed in this section. The fabric structure with fixed boundaries form chapter two (section 2.2 shapefinding example) is also used in chapter 3. This analysis is perfomed using the existing Fortran code HLAY.FOR. The results of this example analysis are the basis for the comparision of the different codes and method to find the shape of fabric strucutres using the grid method.

3.2.1 The Principles of the Jacobi's Method

Jacobi's Method is an iterative method for solving linear equation systems. It's a fixed point iteration that solves the system $A*x = b$ by the iterative equation $x^{(k+1)} = T*x^{(k)} + c$. This iteration works very good for systems with a lots of zeros.

We divide the matrix A into the three matrices L, D and U:

- All values below the diagonal of matrix A are taken into matrix L
- All values above the diagonal of matrix A are taken into matrix U
- All values on the diagonal of matrix A are taken into matrix D
- $A = L + D + U$

If we combine the matrix equations A*x = b and A = L+D+U to get the equation (L+D+U)*x = b. We transform this equation to D*x = b –(L+ U)*x and divide the result by D to get the following relationship:

$$x = D^{-1}*(b - (L+U)*x) \qquad \text{(Equation 3.1)}$$

This relation can be used as iteration formula:

$$x^{(k+1)} = D^{-1}*(b - (L+U)*x^{(k)}) \qquad \text{(Equation 3.2)}$$

or

$$x^{(k+1)} = D^{-1}*b - D^{-1}*(L+U)*x^{(k)} \qquad \text{(Equation 3.3)}$$

Therefore $T = D^{-1}*(L+U)$ and $c = D^{-1}*b$.

Some point about Jacobi's Method:

- The determinate of D must be non-zero
- The method does not converge, if the spectral radius of T => 1
- The method does converge very slowly, if the spectral radius of T is close to 1
- The convergence is linear at the rate of the largest eigenvalue

3.2.2 Fortran Code of Program HLAY.FOR

=> Fortran code of program HLAY.FOR is printed in APPENDIX A
(Reference 18: Department of Civil Engineering at NJIT)

3.2.3 Description of the Fortran Code HLAY.FOR

Sketch of the typical member:

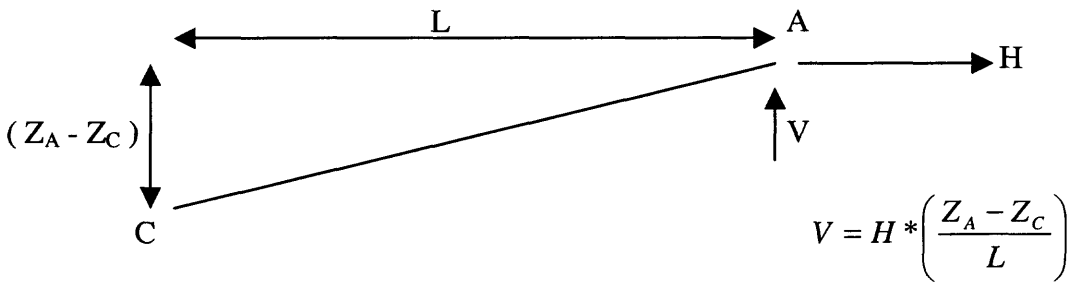

Figure 3.1 Sketch of typical member

The vertical force at node A due to typical member is $V = H*(z_A-z_C)/L$

The term for the typical member j is:

$$(H_j*z_{Aj})/L_j = (H_j*z_{Cj})/L_j \qquad \text{(Equation 3.4)}$$

The Fortran-Code puts the equilibrium on every free node in the system matrix of the investigated structure. Every row of the matrix stays for the equilibrium of one free node. The right hand side (b) of the system equation A*x = b is zero, because b is the resulting vertical force. The vertical horizontal force must be zero at the free nodes, otherwise the node would move.

The constant $c = D^{-1}*b$ is zero, if b = 0 !!!!!!

We have already proved, that b (right hand side) must be zero for this system. Therefore we can reduce the Jacobi's method in this case to the iteration equation:

$$x^{(k+1)} = D^{-1}*(L+U)*x^{(k)} \quad (= T) \qquad \text{(Equation 3.5)}$$

Every equation (row of matrix A) describes the equilibrium conditions at one free node. Every column of the matrix A belongs to one node of the system and its factor shows the influence to the equilibrium at node of this row. The free node of a certain row is always placed on the diagonal of the matrix A.

If we look back to the terms of a typical member, we realize that the term:

$(H_j*z_{Aj})/L_j$ describes the influence of the free node of its own vertical equilibrium

$(H_j*z_{Cj})/L_j$ describes the influence of the neighbor nodes to the vertical equilibrium at free node

The nodes, which are not directly neighboring the free node, have no influence to the vertical equilibrium of the free node and are set to zero

If we divide the matrix A into three parts as explained in the description of the Jacobi's method (matrices L, U and D), we will get the following distribution:

- The component of the free node to the equilibrium of the same node (equations $(H_j*z_{Aj})/L_j$) is placed in the matrix D
- The component of the neighboring nodes to the equilibrium of the free node (equations $(H_j*z_{Cj})/L_j$) is placed in the matrix (L + U)

The computer program adds the components for the whole system. Every equation is set up on the assumption that only one node is free and all other nodes are fixed. The two sums in our Fortran-Code are:

$SUM = \Sigma (H_j / L_j)$	(only for the components that are connected to the free node)
$SUM1 = \Sigma (H_j * z_{Cj} / L_j)$	(only for the components that are connected to the free node)

The Code computes the level of the free node finally with the following equation:

$$z_A = (SUM1 / SUM) = (\Sigma (H_j * z_{Cj} / L_j) / \Sigma (H_j / L_j)) \qquad \text{(Equation 3.6)}$$

This formula is noting else as the Jacobi's method, with the variables:

Jacobi's method: $x^{(k+1)} = D^{-1} * (L+U) * x^{(k)}$ (only term T, because b is zero)

Variables:

$x^{(k+1)} = z_A$

$D^{-1} = (1 / SUM) = (1 / \Sigma (H_j / L_j))$ (Equation 3.7)

$(L+U) * x^{(k)} = SUM1 = (\Sigma (H_j * z_{Cj} / L_j)$ (Equation 3.8)

The problem at this equation is, that we have only the level of the real fixed points. We have only assumptions for the level for all free points, that are assumed to be fix in the calculation. If we perform more steps of this iteration, we will get better assumptions for the level of the free points. The results converge and will become more accurate. Finally, the results (level of the free points) won't chance anymore and we have the exact level of this point after many iteration steps.

The method works quite good, but it takes a lot of computation time as it needs up to 200 iteration steps. Therefore the performance is not the best. This was the main reason, to look for another method to solve this problem without iteration. The new Java-Code, which was developed in this project, solves the linear equation system using the method of Gaussian Elimination.

3.2.4 Computation of One Example Problem Using Code HLAY.FOR

The fabric structure with fixed boundaries, which is used and describted in example calculation of the grid method by hand in chapter 2.2 is used in the example computation using the code HLAY.FOR (Fortran) as well. The usage of the same fabric structure for all the example analysis of the grid method with different codes in chapter 2 and 3 makes it possible to compare the different methods and the codes. Below is the input file, which includes the geometrie and numeration of the grid that is used for the shapefinding of the fabric structure by grid method.

Input-File of the Fortran Computation:

```
84   49   33
  0.0              0.0              100.0
  6.666666667      0.0              100.0
  13.33333333      0.0              100.0
  20.0             0.0              100.0
  26.66666667      0.0              100.0
  33.33333333      0.0              100.0
  40.0             0.0              100.0
  0.0              3.333333333      100.0
  6.666666667      3.333333333      102.839
  13.33333333      3.333333333      0.0
  20.0             3.333333333      0.0
  26.66666667      3.333333333      0.0
  33.33333333      3.333333333      102.839
  40.0             3.333333333      100.0
  0.0              6.666666667      100.0
  6.666666667      6.666666667      0.0
  13.33333333      6.666666667      104.468
  20.0             6.666666667      0.0
  26.66666667      6.666666667      104.468
  33.33333333      6.666666667      0.0
  40.0             6.666666667      100.0
  0.0              10.0             100.0
  6.666666667      10.0             0.0
  13.33333337      10.0             0.0
  20.0             10.0             105.0
  26.66666667      10.0             0.0
  33.33333333      10.0             0.0
  40.0             10.0             100.0
  0.0              13.33333333      100.0
  6.666666667      13.33333333      0.0
  13.33333333      13.33333333      104.468
  20.0             13.33333333      0.0
  26.66666667      13.33333333      104.468
  33.33333333      13.33333333      0.0
  40.0             13.33333333      100.0
  0.0              16.66666667      100.0
  6.666666667      16.66666667      102.839
  13.33333333      16.66666667      0.0
  20.0             16.66666667      0.0
  26.66666667      16.66666667      0.0
  33.33333333      16.66666667      102.839
```

40.0	16.66666667	100.0
0.0	20.0	100.0
6.666666667	20.0	100.0
13.33333333	20.0	100.0
20.0	20.0	100.0
26.66666667	20.0	100.0
33.33333333	20.0	100.0
40.0	20.0	100.0
1	2	
2	3	
3	4	
4	5	
5	6	
6	7	
1	8	
2	9	
3	10	
4	11	
5	12	
6	13	
7	14	
8	9	
9	10	
10	11	
11	12	
12	13	
13	14	
8	15	
9	16	
10	17	
11	18	
12	19	
13	20	
14	21	
15	16	
16	17	
17	18	
18	19	
19	20	
20	21	
15	22	
16	23	
17	24	
18	25	
19	26	
20	27	
21	28	
22	23	
23	24	
24	25	
25	26	
26	27	
27	28	
22	29	
23	30	
24	31	
25	32	

26	33
27	34
28	35
29	36
30	37
31	38
32	39
33	40
34	41
35	42
36	37
37	38
38	39
39	40
40	41
41	42
36	43
37	44
38	45
39	46
40	47
41	48
42	49
43	44
44	45
45	46
46	47
47	48
48	49
29	30
30	31
31	32
32	33
33	34
34	35

3.2.5 Results of the Example Computation Using HLAY.FOR

Below are the results of the shape finding analysis of the fabric strucutre using the grid method and the Fortran code HLAY.FOR, which works with the iterative Jacobi's method. The Fortran code has perform 200 interation steps to get the following result. The solution is nothing else as the level (z-coordinate) of all the grid points. The computed z-coordinate of the grid points and the already known x- and y- coordinate of these points supply together the searched 3-D coordinate of the grid points. The 3-D coordinates of the grid points can be combined to the searched 3-D shape of the fabric structure. The first three columns of the below solution of the computation via Fortran code are x-, y- and z- coordinates the fabric structure:

0.000	0.000	100.000	1	0	2	8	0	0	0	0	0
6.667	0.000	100.000	1	0	1	3	9	0	0	0	0
13.333	0.000	100.000	1	0	2	4	10	0	0	0	0
20.000	0.000	100.000	1	0	3	5	11	0	0	0	0
26.667	0.000	100.000	1	0	4	6	12	0	0	0	0

33.333	0.000	100.000	1	0	5	7	13	0	0	0	0
40.000	0.000	100.000	1	0	6	14	0	0	0	0	0
0.000	3.333	100.000	1	0	1	9	15	0	0	0	0
6.667	3.333	102.839	1	0	2	8	10	16	0	0	0
13.333	3.333	102.304	0	0	3	9	11	17	0	0	0
20.000	3.333	102.047	0	0	4	10	12	18	0	0	0
26.667	3.333	102.304	0	0	5	11	13	19	0	0	0
33.333	3.333	102.839	1	0	6	12	14	20	0	0	0
40.000	3.333	100.000	1	0	7	13	21	0	0	0	0
0.000	6.667	100.000	1	0	8	16	22	0	0	0	0
6.667	6.667	102.474	0	0	9	15	17	23	0	0	0
13.333	6.667	104.468	1	0	10	16	18	24	0	0	0
20.000	6.667	103.838	0	0	11	17	19	25	0	0	0
26.667	6.667	104.468	1	0	12	18	20	26	0	0	0
33.333	6.667	102.474	0	0	13	19	21	27	0	0	0
40.000	6.667	100.000	1	0	14	20	28	0	0	0	0
0.000	10.000	100.000	1	0	15	23	29	0	0	0	0
6.667	10.000	102.350	0	0	16	22	24	30	0	0	0
13.333	10.000	104.204	0	0	17	23	25	31	0	0	0
20.000	10.000	105.000	1	0	18	24	26	32	0	0	0
26.667	10.000	104.204	0	0	19	25	27	33	0	0	0
33.333	10.000	102.350	0	0	20	26	28	34	0	0	0
40.000	10.000	100.000	1	0	21	27	35	0	0	0	0
0.000	13.333	100.000	1	0	22	36	30	0	0	0	0
6.667	13.333	102.474	0	0	23	37	29	31	0	0	0
13.333	13.333	104.468	1	0	24	38	30	32	0	0	0
20.000	13.333	103.838	0	0	25	39	31	33	0	0	0
26.667	13.333	104.468	1	0	26	40	32	34	0	0	0
33.333	13.333	102.474	0	0	27	41	33	35	0	0	0
40.000	13.333	100.000	1	0	28	42	34	0	0	0	0
0.000	16.667	100.000	1	0	29	37	43	0	0	0	0
6.667	16.667	102.839	1	0	30	36	38	44	0	0	0
13.333	16.667	102.304	0	0	31	37	39	45	0	0	0
20.000	16.667	102.047	0	0	32	38	40	46	0	0	0
26.667	16.667	102.304	0	0	33	39	41	47	0	0	0
33.333	16.667	102.839	1	0	34	40	42	48	0	0	0
40.000	16.667	100.000	1	0	35	41	49	0	0	0	0
0.000	20.000	100.000	1	0	36	44	0	0	0	0	0
6.667	20.000	100.000	1	0	37	43	45	0	0	0	0
13.333	20.000	100.000	1	0	38	44	46	0	0	0	0
20.000	20.000	100.000	1	0	39	45	47	0	0	0	0
26.667	20.000	100.000	1	0	40	46	48	0	0	0	0
33.333	20.000	100.000	1	0	41	47	49	0	0	0	0
40.000	20.000	100.000	1	0	42	48	0	0	0	0	0

Comments:

The number of the line is the node number. The first and second columns are the x - and the y – coordinates. The third column is the computed level of the node. The level of zero was set up to 100 at the input-file, therefore the number in the third column subtract by 100 is the z – coordinate of this node in the fabric structure. The results are correct and the nearly same as in the previous computation (only negligible differences). The accuracy of the computation using the Fortran code (interative Jacobi's mehtod) is very good. The comparison to the other codes is made in the following sections.

3.3 Grid Method Using Gaussian Elimination (New Java Code)

This section documents the deveopement of a new java code for the computation of the grid method for shapefinding of fabric structures. This new java program is called shapeApp.java and it uses the Gaussian elimination to solve the linear equation system, which defines the grid of the fabric structures. In other words, the program uses a linear equation system and the Gaussian elimination to perform the grid method and find the shape of the fabric structure. The program language of java has been choosen for this task, because of it much better capability to create complex graphics and outputs. The development of the java code shapeApp.java, the proberties of this code and the used mathematical methods are detailed described and explained in this section. Finally the same fabric structure with fixed boundary conditions, which has aready been used for the example analysis in the section 2.2 and 3.2 (by hand and using Fortran code HLAY.FOR) is used for an example computation using the new Java code shapeApp.java (performing Grid method). The results of this new example computation are documented in this section and they are used to check the new developed code shapeApp.java. Additionally the results of the shape finding process using the new java code are compared with the results of the other computations.

3.3.1 The Principles of the Gaussian Elimination

The system of linear equation is set up in matrices to perform the computation. Gaussian Elimination can be used at the following system:

$$A*x = b \qquad \text{(Equation 3.9)}$$

This system can be solved using back substitution, if the matrix A is triangular.

The Gaussian Elimination transforms the matrix A to a triangular matrix by eliminate certain coefficients of the matrix. This makes it possible to solve the linear equation system. The elimination of single matrix coefficients at the Gaussian Elimination works by the following elementary row operation:

- Multiply a row by a constant
- Add/subtract one row form another
- Swap two rows

The following problems can arise when using Gaussian Elimination:

- Dividing by zero. There will be problems if you have a zero on the diagonal.
- Build up of errors due to small divisors leading to round-off errors.
- Ill-conditioned systems can lead to huge errors. Small chances on the right hand side cause large chances in the results.

These problems must be considered to prevent errors when using Gaussian Elimination.

3.3.2 Description of the Java Code shapeApp.java

Set up of the Grid:

First we need a grid. This grid can be generated automatically by the Java Code, which saves time at the input. The only input that is needed by the program is the total length of the grid in both directions and the fields of the grid in both direction, which are represented in the program by following variables:

lx = total length of grid in x-direction
ly = total length of grid in y-direction
nx = number of fields in which is the grid divided in x-direction
ny = number of fields in which is the grid divided in y-direction

Chosen datatyes:

double = The length of the grid fields should be computed as accurate as possible, therefore the data type double, which is most exact, is used.

int = The number of fields can only be a integer, therefore the data type integer is used.

The code generates the grid with all node coordinates and node numbers. The node number 1 has the coordinates (0, 0). The node number to has the coordinate y = 0 and a positive value for x. The node numbers are increasing in direction of the positive x-axes and y-axis. A certain row in x-direction must be first completed, before the nodes at the next y-level will be assigned.

Input of the fixed points

The program must recognize later, which are fixed and which are free nodes of the structure. Therefore the values for v should be nonzero for every fixed point. The program recognize the fixed points in this case, because all free nodes have assigned the initial value of v = 0. Therefore we add the value of 100 to every known level of a fixed point, to make ensure that the Java-Code makes a right distinction between fixed and the free nodes. All nodes that are on the edge of the grid are assigned as fixed points. The Java-Code assumes the level of all edge nodes automatically to zero. The level of the edge nodes can be overridden in the manual assignment of node levels, if it is different from zero. Finally, we assign the level of all other fixed nodes by manual input. Don't forget to add the value of 100 to the real levels of the fixed nodes. Finally we have an input in which the level of all fixed nodes in nonzero and the level of all free nodes is zero.

Transformation of the node levels to vertical forces:

We must consider one important thing, before we set up the system matrix. All levels of the fixed points are known, but we don't know the vertical forces. On the other side we know that the resulting vertical force at the free nodes must be zero, but we don't know the level of the free nodes. Let's have a quick look at the system matrix A*x = b and investigate its the members. The left side (A*x) is nothing else as the vertical equilibrium equations of all the nodes in matrix form. The matrix A contains the horizontal distances between the nodes and the horizontal forces of the grid members (plane geometry and equilibrium). The vector x contains the level (height) of the fixed nodes. Finally the vector b contains the resulting vertical forces at the nodes. We need to know the coefficients of the matrix A and vector b to solve this matrix system by Gaussian Elimination.

- The coefficients of matrix A and vector b are known for all free points:

  It is easy to set up the coefficients of matrix A for the free points, as we know the horizontal geometry of the grid and the horizontal forces (both are defined). The resulting vertical force of the free node must be zero, because otherwise the node would move. Therefore the coefficient of b must be zero for free node.

- The coefficients of vector b are not known for the fixed points:

  It would be also easy to set up the coefficients of matrix A for fixed points, but this wouldn't help us. We cannot solve this problem in this way, because we don't know the coefficient of vector b (resulting vertical force) for the fixed nodes.

Solution of this problem:

Every row of the system matrix belongs to one node. All rows that belong to a free node can be set up easily, because we know all coefficients that are needed. The goal of this computation is the get the vertical coordinates (levels) of the nodes. These are already known for the fixed points. We don't have to compute this values, but we need them in the system matrix to compute the level of the free nodes. Therefore we put for every fixed node only the solution (x = level of the fixed node) in the corresponding row of the system matrix. This can be made by the doing the following operations in every row that belongs to a fixed node. Set up the diagonal coefficient of the matrix in this row to one and the rest of the row to zero. Take the level (vertical coordinate) of the fixed point and define it to the coefficient of vector b in this row.

- Proof of the input:

  If you solve this row for x, you get the solution x = level of the fixed node.

Set up of the System matrix A*x = b

Every row (equation) of the system matrix belongs to one node. First, we assign all rows that belong to fixed nodes. The Code uses the if-method to recognize the fixed nodes. If the variable v_i in the row i is nonzero, the program assumes that the corresponding node is fixed. All coefficient of the matrix A and vector b are initially assigned to zero, but they can be overridden.

- Set up of the rows that belong to fixed nodes:
  - The diagonal coefficient of matrix A (A[i][i]) are assigned to one
  - The variable v [i] (level of node + 100) is taken for the value of b [i]
  - All other coefficient in this row are zero and x is unknown
- Set up of the rows that belong to free nodes:
  - Sketch of the typical free node and its equilibrium:

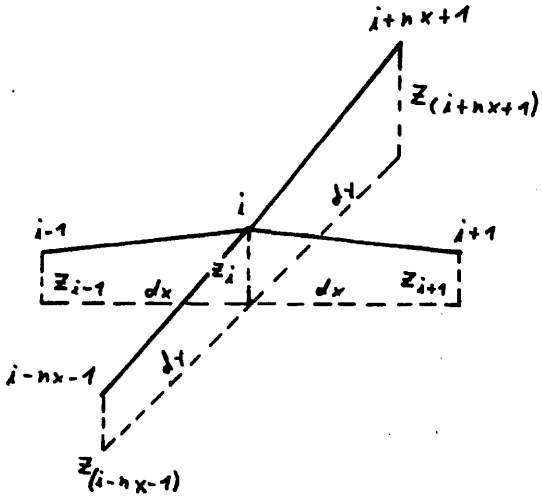

Figure 3.2 Free node and its equilibrium

Node equilibrium at node i:

$$z_i\left[\frac{2}{\left(\frac{lx}{nx}\right)}+\frac{2}{\left(\frac{ly}{ny}\right)}\right]-z_{i+1}\left[\frac{1}{\left(\frac{lx}{nx}\right)}\right]-z_{i-1}\left[\frac{1}{\left(\frac{lx}{nx}\right)}\right]-z_{i+nx+1}\left[\frac{1}{\left(\frac{ly}{ny}\right)}\right]-z_{i+nx+1}\left[\frac{1}{\left(\frac{ly}{ny}\right)}\right]=0$$

(Equation 3.10)

All free nodes are like this typical (sketched) free node and have 4 neighbor nodes. There is no other possibility, because all node at the edge of the grid are assigned as fixed nodes

The program assigns for every row that belongs to a free node the following coefficients:

- A[i][i] = (2/(lx/nx)) + (2/(ly/ny)) (Equation 3.11)
- A[i][i+1] = - (1/(lx/nx)) (Equation 3.12)
- A[i][i-1] = - (1/(lx/nx)) (Equation 3.13)
- A[i][i+nx+1] = - (1/(ly/ny)) (Equation 3.14)
- A[i][i-nx-1] = - (1/(ly/ny)) (Equation 3.15)
- v[i] is taken for b[i], which is zero for all free nodes
- All other coefficients in the row are zero and x is unknown

Java Code performs Gaussian Elimination

The Java Code multiplies rows by a constant and add or subtract rows from each other in order to eliminate coefficients of the matrix A and get a triangular matrix. This process is called Gaussian Elimination. More theoretically information about this can be read earlier in this report (section: A short review on the method Gaussian Elimination). The practical code and performance of this step can be followed in the below printed Java Code.

Java Code solves the system by back substitution

The computer program uses the output of the Gaussian Elimination (triangular matrix). It starts at this row, which has only one unknown variable in its equation. This unknown variable can be solved easily by one multiplication. If this variable is known, the program continues with the next row (equation with two unknown variables). This equation can be reduced to a equation with one unknown variable, because the other unknown variable have been computed one step earlier. As we have only one unknown variable, the equations can be solved easily. The program continues like that, until the complete system is solved and all values of the variables are known. The performance of this process in the program can be followed in the below printed Java Code.

Output

The Java Code creates an output in command prompt. The created output is printed below the Java Code.

3.3.3 Computation of One Example Problem Using Code shapeApp.java

The fabric structure with fixed boundaries that has already been used in the example analysis in section 2.2 and 3.2 (peforming grid method by hand and using Fortran code HLAY.FOR) is also used for the esample problem using code shapeApp.java. There variables of the fabric structure (grid) in this Java Code are exactly the same as in the example problems that have been computed by hand und using the Fortran Code (Jacobi's Method). The different codes and the result of the example analysis can be compared very easily, because the same fabric structure is used for the computation. This example problem has been computed using the Java Code shapeApp.java and the results are listed below the java code. The variables of the fabric structure and the used grid are included in the java code shapeApp.java.

Java Code for shape finding of fabric structure (input is in code included):

```
// Code: shapeApp.java
// Program for shape finding of fabric structures
// shapeApp.java

import java.io.*;
import java.applet.*;
import java.awt.*;

public class shapeApp {

// definition of Grid of Variables

static double lx = 40;          // total lenght of grid in x
static int nx = 6;              // fields in x
static double ly = 20;          // total lenght of grid in y
static int ny = 6;              // fields in y

// length and width of one field

static double dx = (lx/nx);
static double dy = (ly/ny);

// create array of Grid

static int mx = (nx+1);                 // nodes in x-direction
static int my = (ny+1);         // nodes in y-direction
static int m = (my*mx);                 // total amount of nodes
static int as = (m+1);          // size of array
static double x[] = new double [ as ];
```

```
static double y[] = new double [ as ];

// create array of vertical equilibrium

static double v[] = new double [ as ];
static double A[][] = new double [ as ][ as ];
static double z[] = new double [ as ];
static double t[] = new double [ as ];
static double h[] = new double [ as ];

        public static void main(String[] args) {

        // generation of node coordinates
                for ( int i = 0; i <= ny; i++ ) {
                        for ( int j = 0; j <= nx; j++ ) {
                                x[1+j+i*mx] = ((lx*j)/nx);
                        }
                }
                for ( int i = 1; i <= mx; i++ ) {
                        for ( int j = 0; j <= ny; j++ ) {
                                y[i+j*mx] = ((ly*j)/ny);
                        }
                }

        // definition of the fix nodes (array)
                for ( int i = 1; i <= mx; i++ ) {
                        v[i] = 100;
                        v[i+m-mx] = 100;
                }
                for ( int i = 1; i <= (my-2); i++ ) {
                        v[i*mx+1] = 100;
                        v[i*mx+mx] = 100;
                }

        // input level of the fix points
                v[9] = 102.839;
                v[13] = 102.839;
                v[17] = 104.468;
                v[19] = 104.468;
                v[25] = 105;
                v[31] = 104.468;
                v[33] = 104.468;
                v[37] = 102.839;
                v[41] = 102.839;
```

```
        // create Matrix A
                for ( int i = 1; i <= m; i++ ) {
                        if ( v[i] != 0 )
                                A[i][i] = 1;
                        else {
                                A[i][i] = ((2/dy)+(2/dx));
                                A[i][i-1] = (-1/dx);
                                A[i][i+1] = (-1/dx);
                                A[i][i-my] = (-1/dy);
                                A[i][i+mx] = (-1/dy);
                        }
                }

        // solve matrix (Gaussian Elimination)
                for ( int k = 2; k <= m; k++ ) {
                        for ( int i = k; i <= m; i++ ) {
                                double fa = (A[i][k-1]/A[k-1][k-1]);
                                for ( int j = k; j <= m; j++ ) {
                                        A[i][j] = (A[i][j]-(fa*A[k-1][j]));
                                }
                                A[i][k-1] = 0;
                                v[i] = (v[i]-(fa*v[k-1]));
                        }
                }

                z[m] = (v[m]/A[m][m]);
                for ( int k = (m-1); k >= 1; k-- ) {
                        for ( int i = (m-1); i >= 1; i-- ) {
                                t[i] = (t[i]+(z[k+1]*A[i][k+1]));
                        }
                        z[k] = ((v[k]-t[k])/A[k][k]);
                }
                for ( int i = 1; i <= m; i++ ) {
                        h[i] = (z[i]-100);
                }

                System.out.println( "Generation of Grid was successful" );
                System.out.println( "Output of the Node-Coordinates:" );
                for ( int i = 1; i <= m; i++ ) {
                System.out.println( "Node " + i );
                System.out.println( "X " + x[i] + "  Y " + y[i] + "  Z " + h[i] );
                }
        }
}
```

3.3.4 Results of the Example Computation Using shapeApp.java

Output of the results (3-D coordinates) of the compuation:

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\ab52>javac shapeApp.java

C:\Documents and Settings\ab52>java shapeApp
Generation of Grid was successful
Output of the Node-Coordinates:

Node 1
X 0.0  Y 0.0  Z 0.0
Node 2
X 6.666666666666667  Y 0.0  Z 0.0
Node 3
X 13.333333333333334  Y 0.0  Z 0.0
Node 4
X 20.0  Y 0.0  Z 0.0
Node 5
X 26.666666666666668  Y 0.0  Z 0.0
Node 6
X 33.333333333333336  Y 0.0  Z 0.0
Node 7
X 40.0  Y 0.0  Z 0.0
Node 8
X 0.0  Y 3.3333333333333335  Z 0.0
Node 9
X 6.666666666666667  Y 3.3333333333333335  Z 2.8389999999999986
Node 10
X 13.333333333333334  Y 3.3333333333333335  Z 2.303733333333355
Node 11
X 20.0  Y 3.3333333333333335  Z 2.0474000000000245
Node 12
X 26.666666666666668  Y 3.3333333333333335  Z 2.303733333333355
Node 13
X 33.333333333333336  Y 3.3333333333333335  Z 2.8389999999999986
Node 14
X 40.0  Y 3.3333333333333335  Z 0.0
Node 15
X 0.0  Y 6.666666666666667  Z 0.0
Node 16
X 6.666666666666667  Y 6.666666666666667  Z 2.4744074074074263
```

Node 17
X 13.333333333333334 Y 6.666666666666667 Z 4.4680000000000035
Node 18
X 20.0 Y 6.666666666666667 Z 3.8384666666666902
Node 19
X 26.666666666666668 Y 6.666666666666667 Z 4.4680000000000035
Node 20
X 33.333333333333336 Y 6.666666666666667 Z 2.4744074074074263
Node 21
X 40.0 Y 6.666666666666667 Z 0.0
Node 22
X 0.0 Y 10.0 Z 0.0
Node 23
X 6.666666666666667 Y 10.0 Z 2.350222222222243
Node 24
X 13.333333333333334 Y 10.0 Z 4.2037037037037095
Node 25
X 20.0 Y 10.0 Z 5.0
Node 26
X 26.666666666666668 Y 10.0 Z 4.2037037037037095
Node 27
X 33.333333333333336 Y 10.0 Z 2.350222222222243
Node 28
X 40.0 Y 10.0 Z 0.0
Node 29
X 0.0 Y 13.333333333333334 Z 0.0
Node 30
X 6.666666666666667 Y 13.333333333333334 Z 2.4744074074074405
Node 31
X 13.333333333333334 Y 13.333333333333334 Z 4.4680000000000035
Node 32
X 20.0 Y 13.333333333333334 Z 3.838466666666676
Node 33
X 26.666666666666668 Y 13.333333333333334 Z 4.4680000000000035
Node 34
X 33.333333333333336 Y 13.333333333333334 Z 2.474407407407398
Node 35
X 40.0 Y 13.333333333333334 Z 0.0
Node 36
X 0.0 Y 16.666666666666668 Z 0.0
Node 37
X 6.666666666666667 Y 16.666666666666668 Z 2.8389999999999986
Node 38
X 13.333333333333334 Y 16.666666666666668 Z 2.303733333333355
Node 39
X 20.0 Y 16.666666666666668 Z 2.0474000000000245

Node 40
X 26.666666666666668 Y 16.666666666666668 Z 2.3037333333333407
Node 41
X 33.333333333333336 Y 16.666666666666668 Z 2.8389999999999986
Node 42
X 40.0 Y 16.666666666666668 Z 0.0
Node 43
X 0.0 Y 20.0 Z 0.0
Node 44
X 6.666666666666667 Y 20.0 Z 0.0
Node 45
X 13.333333333333334 Y 20.0 Z 0.0
Node 46
X 20.0 Y 20.0 Z 0.0
Node 47
X 26.666666666666668 Y 20.0 Z 0.0
Node 48
X 33.333333333333336 Y 20.0 Z 0.0
Node 49
X 40.0 Y 20.0 Z 0.0

C:\Documents and Settings\ab52>

Comments and comparison with results of section 2.2 and 3.2:

The accuracy of the computation of the shape of the fabric structure using the java code shapeApp.java (Gaussian eliminaton) is very good. The results are correct and the nearly same as in the previous computation (performing Grid method by hand or using Fortran code HLAY.FOR). There are only negligible differences between the result of these compuations. This shows that the accurancy of all these three methods are around the same. It doesn't make a difference for the accurancy, which method is choosen for the shapefinding process of fabric structure by the Grid method. However, there are differences in the efficiency. The Fortran code HLAY.FOR works with the Jacobi's method and performs 200 iteration steps until it gets the final result. This leads to a huge amount of operations. The java code shapeApp.java works with Gaussian elimination and need therefore no iteration process. The result of this are much less operation. The java code is therefore much more efficient as the Fortran code. The computation by hand (section 2.2) is no longer considered in the comparison, because this option is would be to labour intensive and nearly impossible to realize for huge systems of fabric structures. Additionally it make not much sense to compare the amout of operations of a method perfomed by computer and a method performed by hand.

The results of the computation are 3-D coordinates of the grid nodes (points) of the fabric structures. The coordinates of the 3-D coordinates of the grid nodes can be combined. The combined grid nodes represent the computed shape of the fabric structure, which is nothing else than the 3-D shape of the defined grid.

3.4 Refinement of the Java Code shapeApp.java

The Java Code can be optimized by avoiding the lines of the matrix, which correspond to the fixed points. These lines contain no equations and these lines define only the level (z-coordinate) of the fixed node. Therefore it would be much more efficient to avoid this lines. The matrix would become much smaller and would only have lines that contain equations (lines that belongs to free nodes). As a result of this, the computer code might has a much better performance and the results could be more accurate. The improved Java-Code based on the first Java Code. The definition of the variables and the Gaussian Elimination are exactly the same as in the computation before. There are only two new (additional) sections in the Java Code shapeApp.java.

The first one kicks out the lines of the matrix that belongs to the fixed points and contains no equations. This section computes also the constant factors, that are caused in the other equations by this fixed nodes and set this factor in the concerning equations. Finally, the section creates a new and smaller matrix, which contains only the lines that correspond to the free nodes. This part is implemented by if-conditions.

The second section combines the results of the Gaussian Elimination (z-coordinates of the free nodes) and the already known z-coordinates of the fixed nodes. It selects the corresponding the level (z-coordinate) for every node number. This must be done because the lines of the matrix and the node number are messed up, because the fixed node have been removed from the matrix. Therefore the computer code must sort the node after the computation and find the right level (z-coordinate) for every grid point. This new sort-section of the code is also implemented by if-conditions.

3.4.1 Computation of the Example Problem Using Refined Java Code improvedshapeApp.java

The variables in the improved Java Code are the same as in the previous computation. The same fabric structure with fixed boundaries, which has already be used in the sections 2.2, 3.2 and 3.2 (example analysis by hand, Fortran code HLAY.FOR and java code shapeApp.java), has also been used for the example analysis in this section (using java code improvedshapeApp.java). The main advantage of using the exactly the same fabric structure with fixed boundaries for all example computation in chapter 2 and chapter 3 is, that the results of these different methods can be checked very quickly and the comparison of these codes and methods gets a very simple task. The reuslts need not to be transformed before a comparison. The computer code of this example problem (improvedshapeApp.java) and the results are listed below. The variables of the fabric structure and the used grid are included in the java code improvedshapeApp.java.

<u>Improved java code for shape finding of fabric structure (input is in code included):</u>

```
// Code: improvedshapeApp.java
// Program for shape finding of fabric structures
// improvedshape.java
```

```
// initialize variables

import java.io.*;
import java.applet.*;
import java.awt.*;

public class improvedshapeApp {

// definition of Grid of Variables

static double lx = 40;          // total lenght of grid in x
static int nx = 6;              // fields in x
static double ly = 20;          // total lenght of grid in y
static int ny = 6;              // fields in y

// length and width of one field

static double dx = (lx/nx);
static double dy = (ly/ny);

// create array of Grid

static int mx = (nx+1);                 // nodes in x-direction
static int my = (ny+1);         // nodes in y-direction
static int m = (my*mx);                 // total amount of nodes
static int as = (m+1);          // size of array
static double x[] = new double [ as ];
static double y[] = new double [ as ];

// create array of vertical equilibrium

static double v[] = new double [ as ];
static double A[][] = new double [ as ][ as ];
static double z[] = new double [ as ];
static double t[] = new double [ as ];
static double h[] = new double [ as ];
static double R[] = new double [ as ];
static double b[] = new double [ as ];
static double B[][] = new double [ m ][ m ];
static double u[] = new double [ m ];
static double L[] = new double [ m ];

        public static void main(String[] args) {
        {
        // generation of node coordinates
                for ( int i1 = 0; i1 <= ny; i1++ ) {
```

```
                for ( int j = 0; j <= nx; j++ ) {
                        x[1+j+i1*mx] = ((lx*j)/nx);
                }
        }
        for ( int i2 = 1; i2 <= mx; i2++ ) {
                for ( int j = 0; j <= ny; j++ ) {
                        y[i2+j*mx] = ((ly*j)/ny);
                }
        }

// definition of the fix nodes (array)
        for ( int i3 = 1; i3 <= mx; i3++ ) {
                v[i3] = 100;
                v[i3+m-mx] = 100;
        }
        for ( int i4 = 1; i4 <= (my-2); i4++ ) {
                v[i4*mx+1] = 100;
                v[i4*mx+mx] = 100;
        }

// input level of the fix points
        v[9] = 102.839;
        v[13] = 102.839;
        v[17] = 104.468;
        v[19] = 104.468;
        v[25] = 105;
        v[31] = 104.468;
        v[33] = 104.468;
        v[37] = 102.839;
        v[41] = 102.839;

// count amount of fix nodes and free nodes
        int fix = 0;
        int free = 0;
        for ( int i5 = 1; i5 <= m; i5++ ) {
                if ( v[i5] != 0 )
                        fix = (fix+1);
                else {
                        free = (free+1);
                }
        }

// create Matrix A
        for ( int i = 1; i <= m; i++ ) {
                if ( v[i] != 0 )
                        A[i][i] = 1;
```

```
                else {
                        A[i][i] = ((2/dy)+(2/dx));
                        A[i][i-1] = (-1/dx);
                        A[i][i+1] = (-1/dx);
                        A[i][i-my] = (-1/dy);
                        A[i][i+mx] = (-1/dy);
                }
        }

// calculate values at right side
        for ( int i = 1; i <= m; i++ ) {
                if ( v[i] != 0 ) {
                        for ( int j = 1; j <= m; j++ ) {
                                R[j] = (R[j]+(A[j][i]*v[i]));
                        }
                }
                else {
                }
        }

        for ( int k = 1; k <= m; k++ ) {
        b[k] = (v[k]-R[k]);
        }

// create new (smaller) matrix and linear equation system
        int c1 = 0;
        for ( int i = 1; i <= m; i++ ) {
                if ( v[i] == 0 ) {
                c1 = (c1+1);
                u[c1] = b[i];
                int c2 = 0;
                        for ( int j = 1; j <= m; j++ ) {
                                if ( v[j] == 0 ) {
                                c2 = (c2+1);
                                B[c1][c2] = A[i][j];
                                }
                                else {
                                }
                        }
                }
                else {
                }
        }
```

```
        // solve matrix (Gaussian Elimination)
                for ( int k = 2; k <= free; k++ ) {
                        for ( int i = k; i <= free; i++ ) {
                                double fa = (B[i][k-1]/B[k-1][k-1]);
                                for ( int j = k; j <= free; j++ ) {
                                        B[i][j] = (B[i][j]-(fa*B[k-1][j]));
                                }
                                B[i][k-1] = 0;
                                u[i] = (u[i]-(fa*u[k-1]));
                        }
                }

                L[free] = (u[free]/B[free][free]);
                for ( int k = (free-1); k >= 1; k-- ) {
                        for ( int i = (free-1); i >= 1; i-- ) {
                                t[i] = (t[i]+(L[k+1]*B[i][k+1]));
                        }
                        L[k] = ((u[k]-t[k])/B[k][k]);
                }

                int c3 = 0;
                for ( int i = 1; i <= m; i++ ) {
                        if ( v[i] == 0 ) {
                                c3 = (c3+1);
                                h[i] = (L[c3]-100);
                        }
                        else {
                                h[i] = (v[i]-100);
                        }
                }

        }

                System.out.println( "Generation of Grid was successful" );
                System.out.println( "Output of the Node-Coordinates:" );
                for ( int i = 1; i <= m; i++ ) {
                System.out.println( "Node " + i );
                System.out.println( "X " + x[i] + "  Y " + y[i] + "  Z " + h[i] );
                }
        }
}
```

3.4.2 Results of the Computation Using the Refined Java Code improvedshapeApp.java

Output of the results (3-D coordinates) of the compuation:

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\ab52>java improvedshapeApp
Generation of Grid was successful
Output of the Node-Coordinates:
Node 1
X 0.0  Y 0.0  Z 0.0
Node 2
X 6.666666666666667  Y 0.0  Z 0.0
Node 3
X 13.333333333333334  Y 0.0  Z 0.0
Node 4
X 20.0  Y 0.0  Z 0.0
Node 5
X 26.666666666666668  Y 0.0  Z 0.0
Node 6
X 33.333333333333336  Y 0.0  Z 0.0
Node 7
X 40.0  Y 0.0  Z 0.0
Node 8
X 0.0  Y 3.3333333333333335  Z 0.0
Node 9
X 6.666666666666667  Y 3.3333333333333335  Z 2.8389999999999986
Node 10
X 13.333333333333334  Y 3.3333333333333335  Z 2.303733333333355
Node 11
X 20.0  Y 3.3333333333333335  Z 2.0474000000000245
Node 12
X 26.666666666666668  Y 3.3333333333333335  Z 2.303733333333355
Node 13
X 33.333333333333336  Y 3.3333333333333335  Z 2.8389999999999986
Node 14
X 40.0  Y 3.3333333333333335  Z 0.0
Node 15
X 0.0  Y 6.666666666666667  Z 0.0
Node 16
X 6.666666666666667  Y 6.666666666666667  Z 2.4744074074074263
Node 17
X 13.333333333333334  Y 6.666666666666667  Z 4.4680000000000035
```

Node 18
X 20.0 Y 6.666666666666667 Z 3.838466666666676
Node 19
X 26.666666666666668 Y 6.666666666666667 Z 4.4680000000000035
Node 20
X 33.333333333333336 Y 6.666666666666667 Z 2.4744074074074263
Node 21
X 40.0 Y 6.666666666666667 Z 0.0
Node 22
X 0.0 Y 10.0 Z 0.0
Node 23
X 6.666666666666667 Y 10.0 Z 2.3502222222222287
Node 24
X 13.333333333333334 Y 10.0 Z 4.2037037037037095
Node 25
X 20.0 Y 10.0 Z 5.0
Node 26
X 26.666666666666668 Y 10.0 Z 4.203703703703724
Node 27
X 33.333333333333336 Y 10.0 Z 2.350222222222243
Node 28
X 40.0 Y 10.0 Z 0.0
Node 29
X 0.0 Y 13.333333333333334 Z 0.0
Node 30
X 6.666666666666667 Y 13.333333333333334 Z 2.474407407407412
Node 31
X 13.333333333333334 Y 13.333333333333334 Z 4.4680000000000035
Node 32
X 20.0 Y 13.333333333333334 Z 3.838466666666676
Node 33
X 26.666666666666668 Y 13.333333333333334 Z 4.4680000000000035
Node 34
X 33.333333333333336 Y 13.333333333333334 Z 2.4744074074074263
Node 35
X 40.0 Y 13.333333333333334 Z 0.0
Node 36
X 0.0 Y 16.666666666666668 Z 0.0
Node 37
X 6.666666666666667 Y 16.666666666666668 Z 2.8389999999999986
Node 38
X 13.333333333333334 Y 16.666666666666668 Z 2.303733333333355
Node 39
X 20.0 Y 16.666666666666668 Z 2.0474000000000245
Node 40
X 26.666666666666668 Y 16.666666666666668 Z 2.3037333333333407

Node 41
X 33.333333333333336 Y 16.666666666666668 Z 2.8389999999999986
Node 42
X 40.0 Y 16.666666666666668 Z 0.0
Node 43
X 0.0 Y 20.0 Z 0.0
Node 44
X 6.666666666666667 Y 20.0 Z 0.0
Node 45
X 13.333333333333334 Y 20.0 Z 0.0
Node 46
X 20.0 Y 20.0 Z 0.0
Node 47
X 26.666666666666668 Y 20.0 Z 0.0
Node 48
X 33.333333333333336 Y 20.0 Z 0.0
Node 49
X 40.0 Y 20.0 Z 0.0

C:\Documents and Settings\ab52>

Comments and comparison with the results of sections 2.2, 3.2 and 3.3:

The accuracy of the computation of the shape of the fabric structure using the java code improvedshapeApp.java (a modified Gaussian eliminaton) is very good. The results are correct and the nearly same as in the previous computation (performing Grid method by hand, using Fortran code HLAY.FOR or java code shapeApp.java). There are only negligible differences between the result of these compuations. This shows that the accurancy of all these four methods are around the same. It doesn't make a difference for the accurancy, which method is choosen for the shapefinding process of fabric structure by the Grid method. However, there are differences in the efficiency. The comparison in section 3.3 compares the efficiency of the computation by hand, using Fortran code HLAY.FOR and using java code shapeApp.java. The section shows code shapeApp.java has the best efficiency of this three methods. The other two methods are no longer considered in this comparison. Therefore only the two java codes shapeApp.java and improvedshapeApp.java are left for this consideration. The code improvedshapeApp.java needs additional operations for the modifications and refinements of the original code shapeApp.java. However, it doesn't have a better accurancy. Therefore the refinements and modifications didn't improve the computation but it is less efficient. The java code shapeApp.java has the best efficiency and is the most competitive method in chapter 3.

The results of the computation are 3-D coordinates of the grid nodes (points) of the fabric structures. The coordinates of the 3-D coordinates of the grid nodes can be combined. The combined grid nodes represent the computed shape of the fabric structure, which is nothing else than the 3-D shape of the defined grid.

3.5 Comparison of the Three Methods (Codes), Results and Conclusions

Result of compuations in chapter 3

The result of the the computations in chapter 3 is a 3-D Grid, which represents the shape of the fabric structure. The shape of the fabric structure with fixed boundaries that was computed in chapter 3 (result of the computations) are shown in the graphic below.

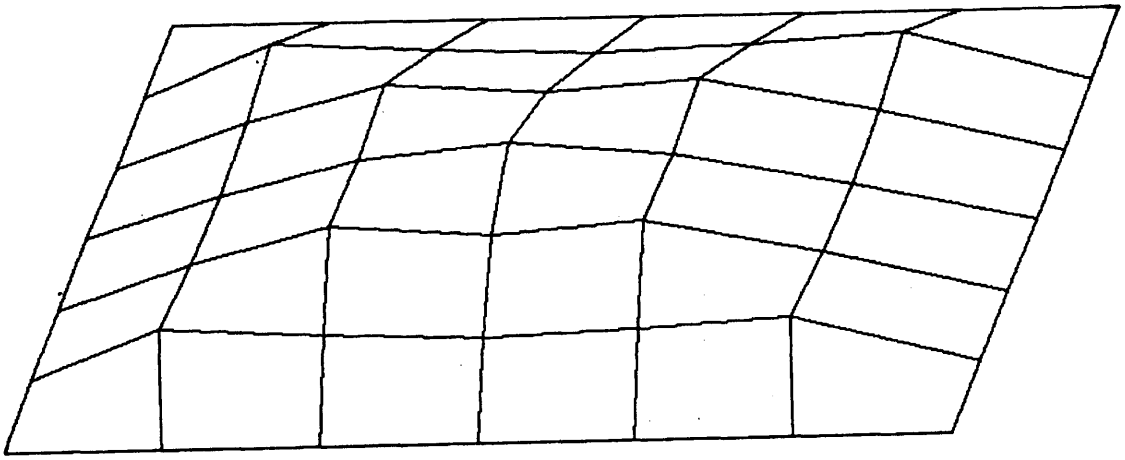

Figure 3.3 3-D graphic of the fabric structure with fixed boundaries

Comparision of the three different methods (codes) in chapter 3

The code shapeApp.java (from section 3.3) for shape finding of fabric structures is most efficient method in chapter 3. It's the best method in this chapter to perform the grid method for fabric structure with fixed boundaries. This method is additionally very stable and accurate. The code shapeApp.java is based on a simple Gaussian elimination algorithm to solve a linear equation system, which represents the equilibrium conditions of the three dimensional grid of the fabric structure.

The stability and the accuracy of the code shapeApp.java (using Gaussian elimination) is around the same as for code HLAY.FOR (using Jacobi's method) and improvedshapeApp.java (using modified Gaussian elimination). All three methods (codes) are very stable and compute very accurate results. However, there is a huge difference in the efficiency of this three methods. The code shapeApp.java (section 3.3) has a much better efficiency and shows the performance. This code (shapeApp.java) solves a linear equation system using the simple Gaussian elimination algorithm. The Fortran code (HLAY.FOR) uses the Jacobi's method and need many iteration steps until it get a accurate results. This code performs up to 100 iteration steps and more until it gets a result. Therefore this method need much more operations and can never match the efficiency of the code shapeApp.java. The other java code (improveshapeApp.java) can nearly match the efficiency of the java code (shapeApp.java). It has only a few more operations, which results from the modifications and refinements of the simple Gaussian elimination algorithm in shapeApp.java. But this modifications and refinements lead

unfortunaltely not to a better accurancy of the result or more stability of the computation. Therefore the modifications and refinements, which results in additional operations in the code improvedshapeApp.java are useless. The code shapeApp.java (section 3.2) that uses the simple Gaussian eliminaton is the best and most efficient code (in chapter 3) to perform the grid method and find the shape of a fabric structure.

A few words about the stability and accuracy of the methods using Gaussian elimination

The following expressions must be less than one, if the maximum slope of the fabric structure is less than 45°:

$$\frac{\Delta z_i}{\left(\frac{lx}{nx}\right)} < 1 \qquad \text{(Equation 3.16)}$$

$$\frac{\Delta z_i}{\left(\frac{ly}{ny}\right)} < 1 \qquad \text{(Equation 3.17)}$$

The condition number of the linear equation system (matrix) is less than 8, if the equations 3.16 and 3.17 are correct (expressions are less than 1). The statement that the condition number is less than 8 can be proved by the single coefficients of the typical node equilibrium (see figure 3.2 and equation 3.10 at page 31). The small condition number (condition number is less than 8, if the slopes of the fabric structure are less than 45°) ensures that the possible rounding errors of the computation are very small.

The linear equation system (matrix) with the smal condition number (less than 8) is also very stable and reliable. This provides the java codes improvedshapeApp.java and shapeApp.java very robust and reliable probertis compared with accurate results. This are perfect features for very computer code. However, we should nor forget the there might be problems for fabric structures with very steep slopes. The results of such structure could even be a "ill-condition system" in very bad cases. Therefore the results of fabric structure with very steep slopes must be checked carefully in order to realize a inaccurate or even wrong results immediately. The problem must be removed from the system or the shape must be computed by another method, if wrong or inaccurate results are found. However, this cases will be very rare. The codes in chapter 3 that are using Gaussian elimination are very reliable and they produce very good results most of the fabric structures.

Conclusion

The two new java codes for perfoming the grid method (shape finding of fabric structure) are very reliable, stable, accurate and efficient. They have both a better efficiency than the Fortran code HLAY.FOR, which has the same stability and accurancy than the two new java codes. The java code shapeApp.java (section 3.2) has the highest efficiency and is the best method (in chapter 3) for finding tha shape of fabric structure.

CHAPTER 4

GRID-METHOD FOR FABRIC STRUCTURES WITH EDGE CABLES

4.1 The Grid for a Fabric Structure with Edge Cable

The grid for a fabric structure with edge cable is a composition of a basic grid and the edge cables. The fixed points of the fabric structure must be on a section of two or more basic grid members. The edge cables and this parts of the basic grids that are surrounded by the edge cables, create together the grid for the shape finding process of the fabric structures with edge cables. All horizontal and vertical lines on the following sketch are the members of the basic grid. The bending lines are the edge cables. The arrows point to the fixed points (supports) of the fabric structure.

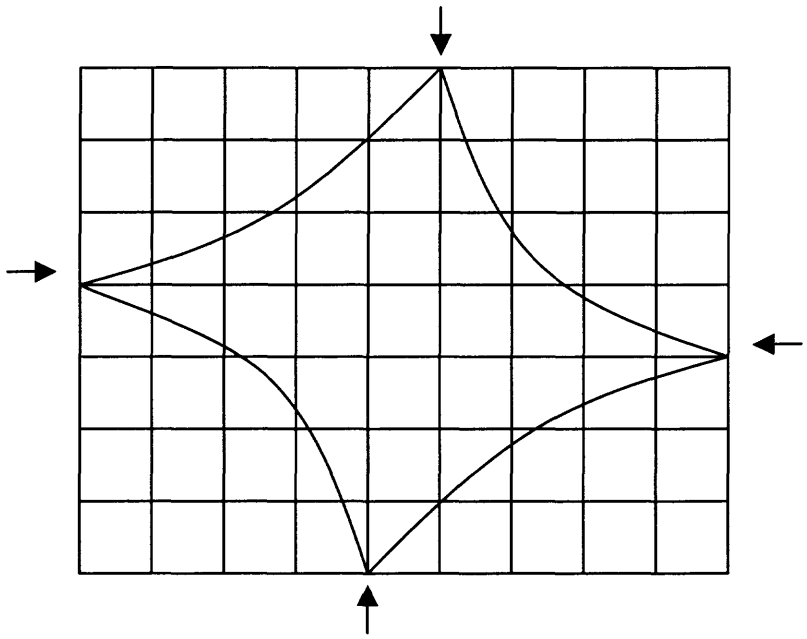

Figure 4.1 Sketch of a basic grid including edge cables

The grid of the fabric structure consists out of the edge cable and the parts of the basic grid that are surrounded by the edge cable. Therefore we get the grid of the fabric structure with edge cable as soon as we delete the parts of the basic grid that are not surrounded by the edge cable.

The coordinates of the basic grid are known. The fixed points of the edge cables are known as well. The coordinates of the fixed points must match with coordinated of the basic grid. In other words, the fixed points must be exactly on a coordinates of the section of two or more basic grid members. The function of the geometry of the edge cable are determined by the equilibrium equations of the edge cables. However, we need

to know the coordinates where the basic grid intersects the edge cables. The coordinates of the points of intersection can be found by defining and solving quadratic equations. More information about computing these coordinates and finding these quadratic equation are in the later sections. The edge cables are modeled in the final grid for the fabric structure by straight lines between these intersection points between basic grid and edge cable.

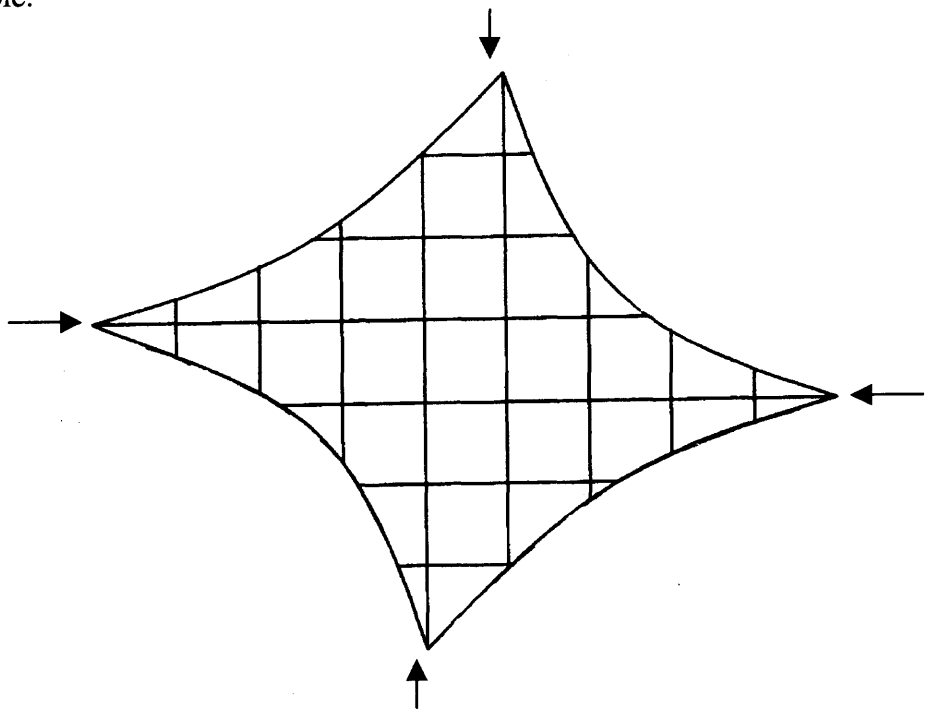

Figure 4.2 Sketch of the grid of a fabric structures with edge cables

4.2 Definition of the Basic Grid

The basic grid exists out of equally spaced horizontal and vertical members. Therefore only the amount of horizontal members, the amount of vertical members, the spacing between the horizontal members and the spacing between the vertical members must be determined. The complete basic grid is determined, once this four variables are known. However, there are a few point that must be considerated when defining the basic grid. First, all fixed points of the fabric structure match with a coordinate of the basic grid. Therefore we have to choose a grid, where all fixed points of the fabric structure will hit a coordinate of this basic grid. Second, the basic grid must cover the complete area of the fabric structure.

The four variables that determine the basic grid are:

- dx = spacing between the vertical members in feet
- dy = spacing between the horizontal members in feet
- nx = amount of vertical members
- ny = amount of horizontal members

The formula to compute the coordinates of the basic grid members are:

- vertical members (array: from i = 1 to i = nx; step size = 1)

$$xv[i] = (i * dx) \qquad \text{(Equation 4.1)}$$

- horizontal members (array: from i = 1 to i = ny; step size = 1)

$$yh[i] = (i * dy) \qquad \text{(Equation 4.2)}$$

4.3 Plane Geometry of the Edge Cable

One major difference between the grid method for fabric structures with fixed boundaries and the fabric structures with edge cables is the geometry of the horizontal grid, which is the starting point for this method. The grid geometry for computing the shape of fabric structure with fixed boundaries are completely determined. Only horizontal and vertical members are used in this case and all members are carrying the same forces. The coordinates of all members are already known. This is completely different in fabric structures with edge cables. The forces and geometry of the edge cables are unknown. Therefore the coordinates of the boundary nodes and the stresses in the boundary members are unknown. The geometry of the horizontal grid for starting the grid method is unknown, because the of the unknown boundary conditions. However, the grid method cannot be performed without a determined geometry and static equilibrium of the horizontal grid.

The geometry in which the edge cable is in static equilibrium must be determined in order to determine the variables of the basic horizontal grid for performing the grid method. It is enough to compute only the plane geometry of the edge cable, because only the horizontal (2-D) coordinates of the edge cables are needed for the horizontal grid of the fabric structure. This section shows how to compute the plane geometry of the edge cables. An equal prespanned fabric is assumed, which to edge cables that are loaded by an uniform load. This uniform load and the maximum deflection of the edge cable must be chosen in order to compute the plane geometry of the edge cables. The geometry and the stresses in the members of the horizontal grid can be computed easily, once the plane geometry of the edge cable is known. All grid members which are not boundary members are carrying the same stresses in order to model the equally prespanned fabric.

4.3.1 Loading of the Edge Cable

The start point of the grid method is the plane grid (structure) that is in horizontal equilibrium. The stresses in the grid members are the equal in the complete structure, with the exception of the boundary (edge) members. The edge members present the edge cables while all other members present the fabric of the structure. We can assume exact the same tension force in every part of the fabric, because all other members of the grid

contain the same stresses. This means that the edge cable is loaded by a constant, vertical load. This load reflects the equally pre-stressed fabric that is supported by the edge cable. The following sketches shows the constant, vertical loading of the corresponding edge cables, if σ_x and σ_y are the basic stresses (pre-stressed initial forces of the fabric):

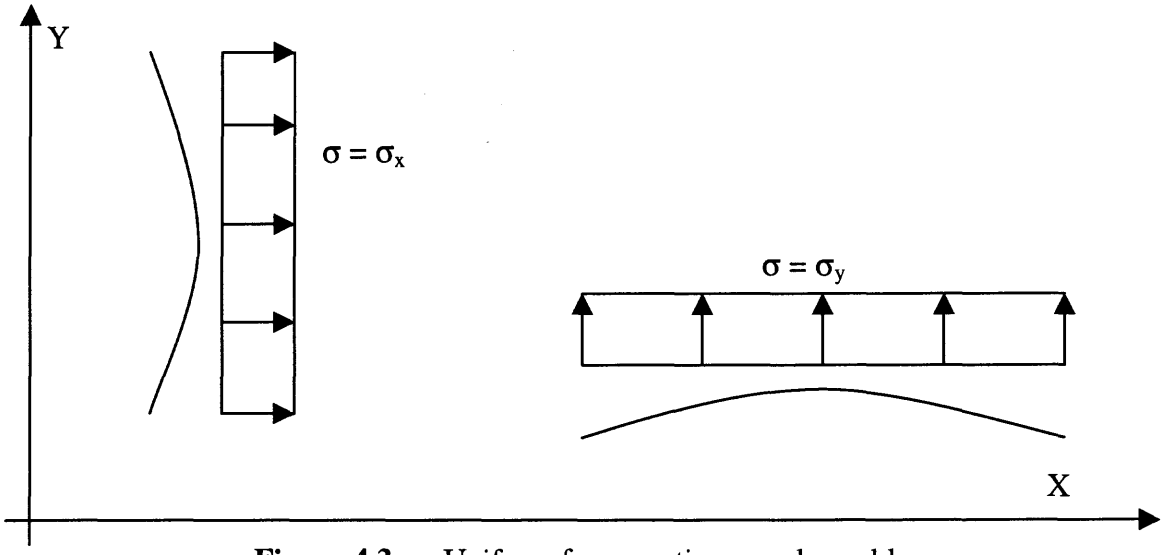

Figure 4.3 Uniform forces acting on edge cable

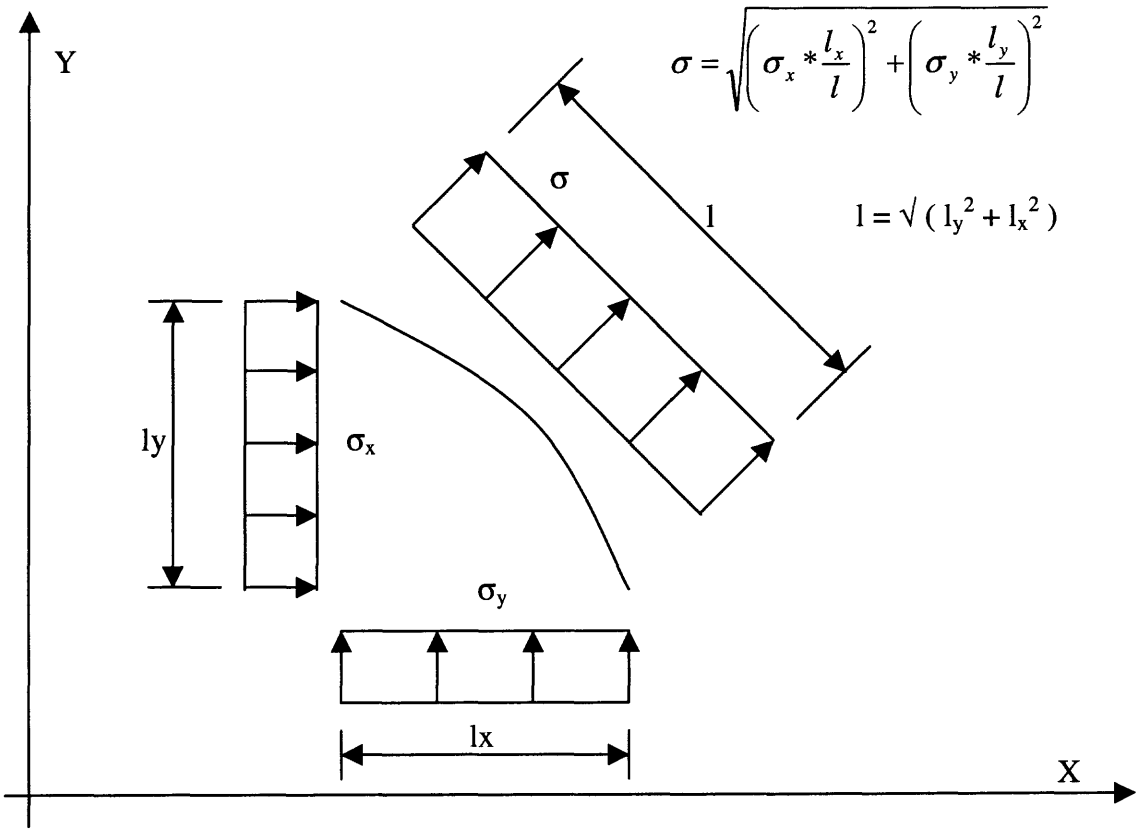

Figure 4.4 Uniform forces acting on edge cables

Static analysis of the edge cable:

The weight and the elongition of the cable can be neglected at this moment of the analysis (grid method). However, these factors must be considered in the next steps of the design process of fabric structures. The following equations describe the equilibrium conditions, if the self-weight and elongition of the cable can be neglected. These equations are used for the first design step (shape finding of fabric structure).

The displacements of the edge cable (q = σ):

$$y(x) = \frac{q * x}{2 * H} * (L - x) \qquad \text{(Equation 4.3)}$$

The displacement in the middle of the cable is (q = σ):

$$y\left(\frac{L}{2}\right) = f = \frac{q * L^2}{8 * H} \qquad \text{(Equation 4.4)}$$

The horizontal force at the cable support is (q = σ):

$$H = \frac{q * L^2}{8 * f} = \frac{q * L^2}{8 * y\left(\frac{L}{2}\right)} \qquad \text{(Equation 4.5)}$$

The geometry of the cable is given by the following function (q = σ):

$$y(x) = \frac{4 * f * x}{L^2} * (L - x) \qquad \text{(Equation 4.6)}$$

(The cable geometry is expressed by the linear equation of the second order)

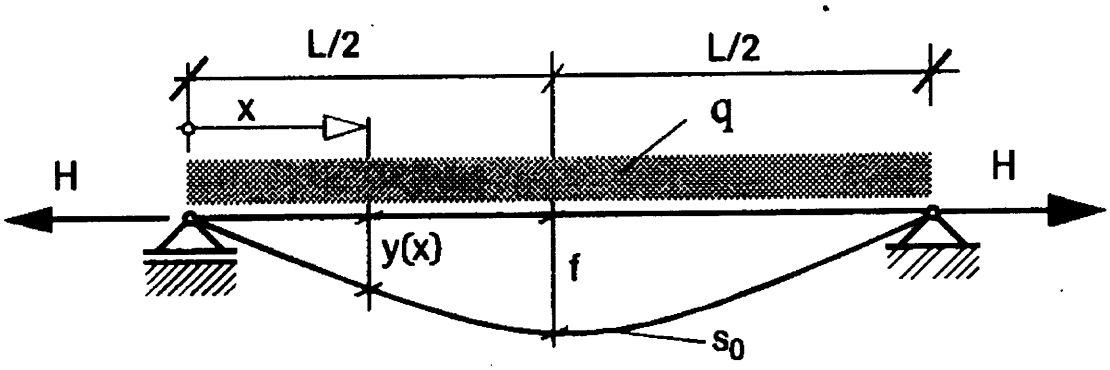

Figure 4.5 Static of cables under uniform load

4.3.2 Define the Geometry of the Edge Cables

Every edge cable has two fix supports. The geometry of this support point are known. The coordinates of this determined edge points are declared in the following section as:

- Point one with the coordinate variables X_1 and Y_1
- Point two with the coordinate variables X_2 and Y_2

The coordinates of one more point are required to determine the function of the edge cable geometry. The middle point of the cable span (half of the cable length) with the maximum cable displacement third point that is chosen for the determination of the cable geometry.

- One must choose the maximum cable displacement that is wanted. The coordinates of the third point can be computed, once this maximum displacement is known. The variable f represents the maximum cable displacement.
- The third point of the cable has the coordinate variables X_1 and Y_1

Formulas for the computation of indeterminate variables:

$$x_m = \frac{x_2 + x_1}{2} \qquad \text{(Equation 4.7)}$$

$$y_m = \frac{y_2 + y_1}{2} \qquad \text{(Equation 4.8)}$$

$$\alpha = \arctan\left(\frac{|y_2 - y_1|}{|x_2 - x_1|}\right) \qquad \text{(Equation 4.9)}$$

$$s = slope = \frac{y_2 - y_1}{x_2 - x_1} \qquad \text{(Equation 4.10)}$$

Definition, if variable f is a positive or negative value:

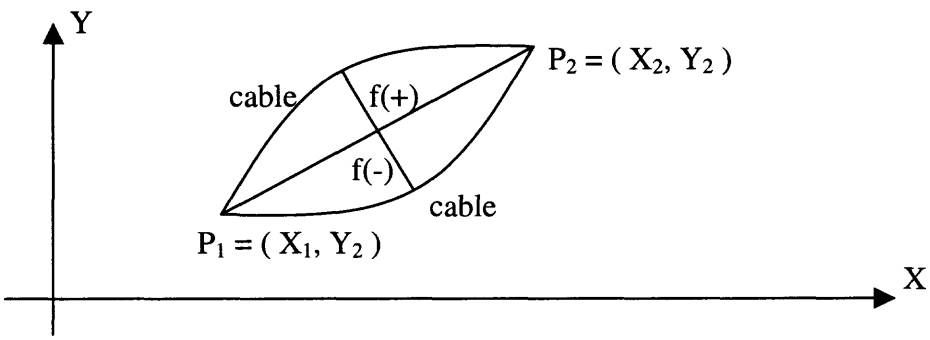

Figure 4.6 Positive and negative max. displacements

If X_2 is not equal to X_1 (s is not infinite), then

- the maximum displacement f must be defined as a positive value, if the cable deflects to the positive y-direction of the coordinate system
- the maximum displacement f must be defined as a negative value, if the cable deflects to the negative y-direction of the coordinate system

If $X_2 = X_1$ (s is infinite), then

- the maximum displacement f must be defined as a positive value, if the cable deflects to the positive x-direction of the coordinate system
- the maximum displacement f must be defined as a negative value, if the cable deflects to the negative x-direction of the coordinate system

Computation of the coordinates of the third point:

⇨ If s = slope < 0, then

$$X_3 = X_m + \sin(\alpha) * f \qquad \text{(Equation 4.11)}$$

$$Y_3 = Y_m + \cos(\alpha) * f \qquad \text{(Equation 4.12)}$$

⇨ If s = slope = 0, then

$$X_3 = X_m \qquad \text{(Equation 4.13)}$$

$$Y_3 = Y_m + f \qquad \text{(Equation 4.14)}$$

⇨ If s = slope > 0, then

$$X_3 = X_m - \sin(\alpha) * f \qquad \text{(Equation 4.15)}$$

$$Y_3 = Y_m + \cos(\alpha) * f \qquad \text{(Equation 4.16)}$$

⇨ If ($X_2 - X_1$) = 0 (no result for s, because s = infinite), then

$$X_3 = X_m + f \qquad \text{(Equation 4.17)}$$

$$Y_3 = Y_m \qquad \text{(Equation 4.18)}$$

4.3.3 Computation of the Function y(x) and x(y) of the Edge Cable Geometry

Computation of the function y(x) of the edge cable geometry

Step one:
Solve the following linear equation system by Gaussian Elimination

$$a * x_1^2 + b * x_1 + c = y_1 \qquad \text{(Equation 4.19)}$$
$$a * x_2^2 + b * x_2 + c = y_2 \qquad \text{(Equation 4.20)}$$
$$a * x_3^2 + b * x_3 + c = y_3 \qquad \text{(Equation 4.21)}$$

(x_i and y_i are known point coordinates)

Step two:
Define the function y(x) using the results from step one (variable a, b and c)

$$y(x) = a * x^2 + b * x + c \qquad \text{(Equation 4.22)}$$

Computation of the function x(y) of the edge cable geometry

Step one:
Solve the following linear equation system by Gaussian Elimination

$$d * y_1^2 + e * y_1 + f = x_1 \qquad \text{(Equation 4.23)}$$
$$d * y_2^2 + e * y_2 + f = x_2 \qquad \text{(Equation 4.24)}$$
$$d * y_3^2 + e * y_3 + f = x_3 \qquad \text{(Equation 4.25)}$$

(x_i and y_i are known point coordinates)

Step two:
Define the function x(y) using the results from step one (variable a, b and c)

$$x(y) = d * y^2 + e * y + f \qquad \text{(Equation 4.26)}$$

4.4 Computation of the Intersection Between Edge Cable and Basic Grid

The intersection coordinates can be found by setting the function of the edge cable geometry equal to coordinate of the basic grid members. There are two different kinds of intersection coordinates. These two kinds of intersection coordinates are the crossing points between edge cable and horizontal basic grid members, and the crossing points between edge cable and vertical basic grid members.

Intersection coordinates between edge cable and horizontal basic grid members

The y-coordinates of these intersection points are the known y-coordinates of the horizontal basic grid members. The x-coordinates of these intersection points can be found by setting the y-coordinates of the horizontal basic grid member equal to the geometry function of the edge cable y(x). The result is a quadratic equation and the x-coordinates of the intersection points between horizontal basic grid members and edge cable can be found by solving this equation.

Equations to compute the intersection coordinate (edge cable – horizontal basic grid members):

Set y(x) equal to yh[i]:

$$y(x) = a*x*x + b*x + c = yh[i] \qquad \text{(Equation 4.27)}$$

$$a*x*x + b*x + (c - yh[i]) = 0 \qquad \text{(Equation 4.28)}$$

Solve the quadratic equation:

$$s1 = \frac{\left(-b - \sqrt{b^2 - 4*a*(c - yh[i])}\right)}{(2*a)} \qquad \text{(Equation 4.29)}$$

$$s2 = \frac{\left(-b + \sqrt{b^2 - 4*a*(c - yh[i])}\right)}{(2*a)} \qquad \text{(Equation 4.30)}$$

The intersection coordinate is:

- if xmin <= s1 and s1 <= xmax, then xh[i] = s1
- if xmin <= s2 and s2 <= xmax, then xh[i] = s2
- the result can become ambiguous, if both above conditions are correct (xh[i] = s1 and xh[i] = s2). The correct xh-coordinate can be found by hand in this case.

Note:

- xmin is the smallest value out of X1, X2 and X3
- xmax is the greatest value out of X1, X2 and X3

Intersection coordinates between edge cable and vertical basic grid members

The x-coordinates of these intersection points are the known x-coordinates of the vertical basic grid members. The y-coordinates of these intersection points can be found by setting the x-coordinates of the vertical basic grid member equal to the geometry function of the edge cable x(y). The result is a quadratic equation and the y-coordinates of the intersection points between vertical basic grid members and edge cable can be found by solving this equation.

Equations to compute the intersection coordinate (edge cable – vertical basic grid members):

Set x(y) equal to xv[i]:

$$x(y) = d*y*y + e*y + f = xv[i] \qquad \text{(Equation 4.31)}$$

$$d*y*y + e*y + (f - xv[i]) = 0 \qquad \text{(Equation 4.32)}$$

Solve the quadratic equation:

$$s1 = \frac{\left(-e - \sqrt{e^2 - 4*d*(f - xv[i])}\right)}{(2*d)} \qquad \text{(Equation 4.33)}$$

$$s2 = \frac{\left(-e + \sqrt{e^2 - 4*d*(f - xv[i])}\right)}{(2*d)} \qquad \text{(Equation 4.34)}$$

The intersection coordinate is:

- if ymin <= s1 and s1 <= ymax, then yv[i] = s1
- if ymin <= s2 and s2 <= ymax, then yv[i] = s2
- the result can become ambiguous, if both above conditions are correct (yv[i] = s1 and yv[i] = s2). The correct yv-coordinate can be found by hand in this case.

Note:

- ymin is the smallest value out of Y1, Y2 and Y3
- ymax is the greatest value out of Y1, Y2 and Y3

4.4.1 Java Code edgecableApp.java to Compute the (Plane) Boundary Coordinates

The boundary coordinates of the 2-D grid are nothing else as the coordinates of the intersection points between the basic grid members and the edge cables. The following Java code computes these boundary coordinates of the 2-D grid. The input values are already included in this Java code and can be found (and chanced) at the beginning of the code. The input must include the geometry of the basic grid and the geometry of the edge cable. The geometry of the basic grid is determined in this program by the amount of vertical grid members, amount of horizontal grid members, spacing between vertical members and spacing between horizontal members. The geometry of the edge cable is determined by the coordinates Y1, Y2, X1 and X2, which are the coordinates of the two fixed points at the end of the cable and the coordinates Y3 and X3, which are the coordinates of the middle of the cable at the maximum displacement.

Variables of the Java code:

dx = spacing between vertical members
dy = spacing between horizontal members
nx = amount of vertical members
ny = amount of horizontal members

y1, y2, x1 and x2 = coordinates of the fixed points at the two ends of the cable
y3 and x3 = coordinate of the middle of the cable at the maximum displacement

4.4.2 Java Code edgecableApp.java

=> Java code of program edgecableApp.java is printed in APPENDIX B

4.4.3 Input for the Java Code edgecableApp.java

The input is already included in the Java code edgecableApp.java (as mentioned in section 4.4.1) and is stated at the beginning of the code. The variables, which must be defined in the input section, are dx, dy, nx, ny, y1, y2, y3, x1, x2 and x3. These variables are all declared static and they are global. These variables have to be defined inside the code edgecableApp.java in order to perform a computation and check the code. Below is the input that is used for to check the Java code edgecableApp.java. It is the original section in the code that defines the grid of the fabric structure.

<u>Input section of the code edgecableApp.java:</u>

```
// definition of the basic grid variables

static double dx = 1;          // spacing between vertical members in feet
static double dy = 1;          // spacing between horizontal members in feet
static int nx = 6;             // amount of vertical members
static int ny = 6;             // amount of horizontal members

// definition of the edge cable points

// fixed point P1
static double x1 = 4;          // must be different than zero
static double y1 = 1;          // must be different than zero

// fixed point P2
static double x2 = 1;          // must be different than zero
static double y2 = 4;          // must be different than zero

// middle point P3
static double x3 = 2.8;        // must be different than zero
static double y3 = 2.8;        // must be different than zero
```

4.4.4 Output of the Java Code edgecableApp.java

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\ab52>javac edgecableApp.java
C:\Documents and Settings\ab52>java edgecableApp
=================================================================
Fixed points (supports) of the edge cable:
Point 1:  x1 = 4.0  y1 = 1.0
Point 2:  x2 = 1.0  y2 = 4.0
Middle point of the cable (not fixed!!!):
Middle Point:  xm = 2.8  ym = 2.8
=================================================================
Geometry function y(x) for the edge cable
y(x) = a*x*x + b*x + c
Function variables:
a= -0.27777777777777773  b= 0.3888888888888887  c = 3.888888888888889
ymax = 4.0 ymin = 1.0
=================================================================
Geometry function x(y) for the edge cable
x(y) = a*y*y + b*y + c
Function variables:
a= -0.2777777777777777  b= 0.38888888888888634  c = 3.888888888888891
xmax = 4.0 xmin = 1.0
=================================================================
Intersections of edge cable with horizontal basic grid members
Horizontal Edge Point # 1
xh = 4.0  yh = 1.0
Horizontal Edge Point # 2
xh = 3.3999999999999995  yh = 2.0
Horizontal Edge Point # 3
xh = 2.620937271229854  yh = 3.0
Horizontal Edge Point # 4
xh = 0.9999999999999986  yh = 4.0
=================================================================
Intersections of edge cable with vertical basic grid members
Vertical Edge Point # 1
xv = 1.0  yv = 3.999999999999997
Vertical Edge Point # 2
xv = 2.0  yv = 3.399999999999997
Vertical Edge Point # 3
xv = 3.0  yv = 2.620937271229851
Vertical Edge Point # 4
xv = 4.0  yv = 0.9999999999999982
C:\Documents and Settings\ab52>
```

Interpretation of this computation output

The investigated edge cable has 6 intersection points with the basic grid members. This means that the 2-D grid has the following 6 boundary coordinates at this edge cable:

- Boundary point 1: x = 1 y = 4
- Boundary point 2: x = 2 y = 3.4
- Boundary point 3: x = 2.62 y = 3
- Boundary point 4: x = 3 y = 2.62
- Boundary point 5: x = 3.4 y = 2
- Boundary point 6: x = 4 y = 1

The following variables have been the input variables for this computation:

dx = 1	dy = 1	nx = 6	ny = 6	f = 0.424 (max. displacement)
x1 = 4	y1 = 1	x2 = 4	y2 = 1	x3 = 2.8 y3 = 2.8

4.5 Computation of the Forces in the Boundary Members of the 2-D Grid

The computed coordinates of the boundary nodes (points of edge cable) and the earlier defined basic grid determine together the grid for the fabric structure with edge cable. The forces in the every grid member must be computed in order to use this grid for the grid method. Therefore the initial forces in the prestressed basic grid members must be defined first. The initial forces in the edge cables (boundary members) can be computed as soon as the stresses in the basic grid members are known. The following steps are one method to compute the forces in the boundary members (edge cables):

(1) define the uniform acting loads (σ_x and σ_y) on the edge cables (consider definitions in Figures 4.3 and 4.4)
(2) compute the basic grid member forces F_y and F_x using the formulas $F_y = \sigma_y * d_y$ and $F_x = \sigma_x * d_x$. Consider that basic grid member forces F_y and F_x are often equal. In other word $F = F_y = F_x$.
(3) compute the support forces H and V at the two fixed point of the edge cable by the two following equations:

$$H = \frac{\sigma * L^2}{8 * f} = \frac{\sigma * L^2}{8 * y\left(\frac{L}{2}\right)} \qquad \text{(Equation 4.35)}$$

$$V = \frac{\sigma * L}{2} \qquad \text{(Equation 4.36)}$$

where:

$$\sigma = \sqrt{\left(\sigma_x * \frac{l_x}{l}\right)^2 + \left(\sigma_y * \frac{l_y}{l}\right)^2} \qquad \text{(Equation 4.37)}$$

The equation 4.35 is derived in section 4.3.1 (static analysis of the cable) and the equation 4.36 reflects just the static equilibrium conditions (moment around the second fixed point is equal to zero). The description of equation 4.37 can also be found in section 4.3.1 (loading of the edge cable).

(4) convert the support forces H and V into one force in x-direction (H_x+V_x) and one force in y-direction (H_y+V_y)

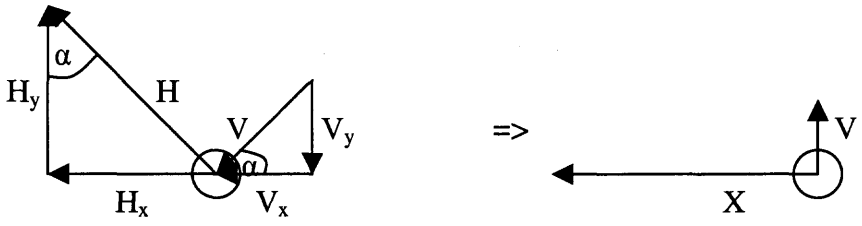

Figure 4.7 Convert support forces into forces in x- and y-direction

$$Y = H_y - V_y = H * \cos\alpha - V * \sin\alpha \qquad \text{(Equation 4.38)}$$

$$X = H_x + V_x = H * \sin\alpha + V * \cos\alpha \qquad \text{(Equation 4.39)}$$

(5) use the law of Pythagoras (a^2+b^2=c^2) and a force triangle to compute the forces in the boundary members (edge cable forces)

$$S_j = \sqrt{\left(\sum X_i\right)^2 + \left(\sum Y_i\right)^2} \qquad \text{(Equation 4.40)}$$

where:

S_j = tension force in boundary member (edge cable) number j

X_i = sum of all forces in x-direction at node i

Y_i = sum of all forces in y-direction at node i

Note: One of the two ends of the boundary member (edge cable number j is the node i

Interpretation of the results of this computation:

The shown method to compute the geometry of the 2D grid and the forces in the boundary members (edge cables) are only an approach to the exact solution. This method is based on a few assumptions to simplify the computation of the analysis. These assumptions are leading to a few small errors and the results of the computation are not the exact results. However, the accuracy of this method good enough for the shape finding process of fabric structures and the errors are therefore negligible small. The computed shape of the fabric structure using the described method are a very good approach to get 3D model of the geometry of the fabric structure, which can be used to start the static and dynamic analysis of the structure. The small errors that are included in the shape of the fabric structure at this stage will be corrected by the following steps of the design analysis of the fabric structure (static and dynamic analysis, patterning).

Example computation for the cable forces of the edge cable in section 4.4.4

Computation of the uniform load:

It is assumed that the fabric is pre-stressed by an uniform tension forces of 100 Kips/foot (line load). Therefore we get the following forces:

σ_x =100 Kips/foot σ_y = 100 Kips/foot

The uniform loading of the edge cable can be computed using equation 4.37:

$$\sigma = \sqrt{\left(\sigma_x * \frac{l_x}{l}\right)^2 + \left(\sigma_y * \frac{l_y}{l}\right)^2} = \sqrt{\left(100 * \frac{3}{4.24}\right)^2 + \left(100 * \frac{3}{4.24}\right)^2} = 100 \frac{Kips}{foot}$$

Computation of the forces in the basic grid members:

The formulas for the computation of the forces in the basic grid members can be found in section 4.5 in step number (2):

$$F_X = \sigma_X * d_X = 100 \frac{Kips}{foot} * 1 foot = 100 Kips$$
$$F_Y = \sigma_Y * d_Y = 100 \frac{Kips}{foot} * 1 foot = 100 Kips$$

Computation of the support forces V and H:

Equation 4.35 and equation 4.36 can be used for the computation of the support forces V and H:

$$H = \frac{\sigma * L^2}{8 * f} = \frac{100 * 4.24^2}{8 * 0.424} = 530 Kips \qquad V = \frac{100 * 4.24}{2} = 212 Kips$$

Convert support forces into one force in x-direction and one force in y-direction:

The support forces can be converted into one force in x-direction and one in y-direction using the equation 4.38 and equation 4.39. We convert the support forces at node 1 of the figure 4.8 (detail A of the sketch of the computed example edge cable):

$$Y_1 = H_y - V_y = H * \cos\alpha - V * \sin\alpha = 530 * \cos 45 - 212 * \sin 45 = 224.9 Kips$$
$$X_1 = H_x + V_x = H * \sin\alpha + V * \cos\alpha = 530 * \sin 45 + 212 * \cos 45 = 524.7 Kips$$

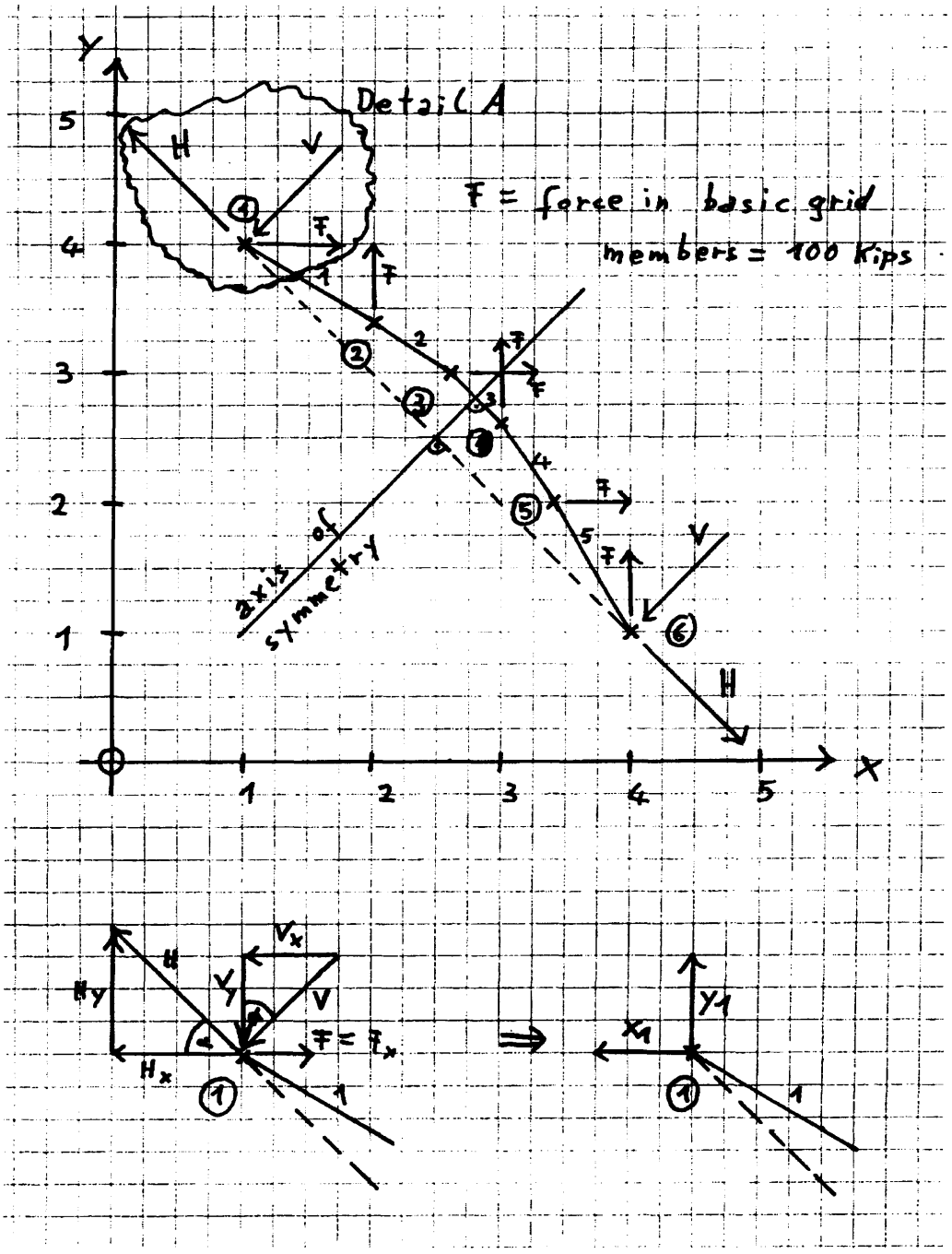

Figure 4.8 Sketch of the boundary members and node of one edge cable

Compute the forces in the single boundary members:

The equation 4.40, which uses the law of Pythagoras and a force triangle, are used to compute the forces in the single boundary members. The sketch of the boundary members in figure 4.8 shows the geometry of the computed edge cable. This loaded and deflected edge cable is symmetric. The displacement of the node 2 is equal to the

displacement of node 5 and the displacement of node 3 is equal to the displacement of node 4. Therefore stresses in the edge cable are symmetric as well, which means that they are on every side (on both halves) the same. In other words, the tension stress in member 1 is equal to the tension stress in member 5 and the tension stress in member 2 is equal to the tension stress in member 4. This means that we only need to compute the forces in one half of the boundary members. The other half of the boundary members have the same stresses, because of the symmetry.

$S_1 = S_5$ => stresses in boundary member 1 and 5 are the same
$S_2 = S_4$ => stresses in boundary member 2 and 4 are the same
S_j = tension force in boundary member (edge cable) number j

Compute tension force in boundary member 1:

$$S_1 = \sqrt{\left(X_1 - \frac{F_x}{2}\right)^2 + (Y_1)^2} = \sqrt{(524.7-50)^2 + (224.9)^2} = 525.3 Kips$$

Compute tension force in boundary member 2:

$$S_2 = \sqrt{\left(X_1 - \frac{F_x}{2}\right)^2 + (Y_1 + F_y)^2} = \sqrt{(524.7-50)^2 + (224.9+100)^2} = 575.2 Kips$$

Compute tension force in boundary member 3:

$$S_3 = \sqrt{\left(X_1 - \frac{3*F_x}{2}\right)^2 + (Y_1 + F_y)^2} = \sqrt{(524.7-150)^2 + (224.9+100)^2} = 495.9 Kips$$

Compute tension force in boundary member 4:

$$S_4 = S_2 = 575.2 Kips$$

Compute tension force in boundary member 5:

$$S_5 = S_1 = 525.3 Kips$$

Note: The computed coordinates of the boundary members and the computed stresses in the boundary members are only an approach. These are no exact results. However, these results are accurate enough for the gird method (shape finding) for fabric structures. The little errors will be corrected in the following design steps.

4.7 Java Code shapeedgeApp.java for Fabric Structures with Edge Cables

```
// Program for shape finding of fabric structures
// edgeshapeApp.java

import java.io.*;
import java.applet.*;
import java.awt.*;

public class edgeshapeApp {

// definition of array sizes
static int an = 36;            // amount of grid nodes
static int am = 61;            // amount of grid members

// create array of Grid
static int as = (an+1);                          // size of array (amount of node + 1)
static double x[] = new double [ as ];           // x-coordinates of nodes
static double y[] = new double [ as ];           // y-coordinates of nodes
static int mb[] = new int [ (am+1) ];            // end 1 of grid members
static int me[] = new int [ (am+1) ];            // end 2 of grid members
static double f[] = new double [ (am+1) ];       // horizontal force in grid members
static double L[] = new double [ (am+1) ];       // length of grid members

// create array of vertical equilibrium
static double v[] = new double [ as ];
static double A[][] = new double [ as ][ as ];
static double z[] = new double [ as ];
static double t[] = new double [ as ];
static double h[] = new double [ as ];


        public static void main(String[] args) {

// input section!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
// coordinates of grid nodes
x[1] = 4;       y[1] = 1
x[2] = 3.45;    y[2] = 2;
x[3] = 4;       y[3] = 2;
x[4] = 4.69;    y[4] = 2;
//    :            :
//    :            :
//   etc.         etc.

// grid members and their pre-stressed forces
mb[1] = 1;            me[1] = 2;            f[1] = 450.2;  // edge cable
```

```
mb[2] = 1;          me[2] = 3;          f[2] = 100;
mb[3] = 1;          me[3] = 4;          f[3] = 631;     // edge cable
mb[4] = 2;          me[4] = 3;          f[4] = 100;
mb[5] = 3;          me[5] = 4;          f[5] = 100;
//      :                   :                   :
//      :                   :                   :
//    etc.                etc.                etc.

        // initialize the other arrays
                for( int a = 1; a <= an; a++ ) {
                        v[a] = 0;
                        z[a] = 0;
                        t[a] = 0;
                        h[a] = 0;
                        for ( int b = 1; b <= an; b++ ) {
                                A[a][b] = 0;
                        }
                }

// input level of the fix points
v[1] = 100;
//      :
//      :
//    etc.


// computation section

        // compute member length
                for( int c = 1; c <= am; c++ ) {
                        int node1 = mb[c];
                        int node2 = me[c];
                        double dy = ( y[node2]-y[node1] );
                        double dx = ( x[node2]-x[node1] );
                        L[c] = ( Math.sqrt(dy*dy + dx*dx) );
                }
        // define matrix A
                for( int d = 1; d <= an; d++ ) {
                        for( int e = 1; e <= am; e++ ) {
                                if( mb[e] == d ) {
                                        int N2 = me[e];
                                        A[d][d] = (A[d][d]+(f[e]/L[e]));
                                        A[d][N2] = (A[d][N2]-(f[e]/L[e]));
                                }
                                if( me[e] == d ) {
                                        int N2 = mb[e];
```

```
                        A[d][d] = (A[d][d]+(f[e]/L[e]));
                        A[d][N2] = (A[d][N2]-(f[e]/L[e]));
                    }
                }
            }
        // define the fixed points in the computation matrix
            for( int g = 1; g <= an; g++ ) {
                if( v[g] != 0 ) {
                    for( int h = 1; h <= an; h++ ) {
                        A[g][h] = 0;
                    }
                    A[g][g] = 1;
                }
            }
        // solve matrix (Gaussian Elimination)
            for ( int k = 2; k <= an; k++ ) {
                for ( int i = k; i <= an; i++ ) {
                    double fa = (A[i][k-1]/A[k-1][k-1]);
                    for ( int j = k; j <= an; j++ ) {
                        A[i][j] = (A[i][j]-(fa*A[k-1][j]));
                    }
                    A[i][k-1] = 0;
                    v[i] = (v[i]-(fa*v[k-1]));
                }
            }
            z[an] = (v[an]/A[an][an]);
            for ( int k = (an-1); k >= 1; k-- ) {
                for ( int i = (an-1); i >= 1; i-- ) {
                    t[i] = (t[i]+(z[k+1]*A[i][k+1]));
                }
                z[k] = ((v[k]-t[k])/A[k][k]);
            }
            for ( int i = 1; i <= an; i++ ) {
                h[i] = (z[i]-100);
            }

            System.out.println( "Generation of Grid was successful" );
            System.out.println( "Output of the Node-Coordinates:" );
            for ( int i = 1; i <= an; i++ ) {
                System.out.println( "Node " + i );
                System.out.println( "X " + x[i] + "  Y " + y[i] + "  Z " + h[i] );
            }
        }
    }
```

4.7 Example Computation of a Fabric Structure with Edge Cable

The goal of this section is to perform an example computation of the shape of a fabric structure with edge cable. The grid method is used for the shape finding for the fabric structure in this example. This example computation performs the shape finding of the below fabric structure with edge cables (figure 4.9) using grid method. The principles that are explained in chapter 4 of this thesis are applied to handle to problem of the edge cable (non-fixed boundaries). The goal of this example computation is to show the practical usage of the grid method for shape finding of fabric structure with edge cable.

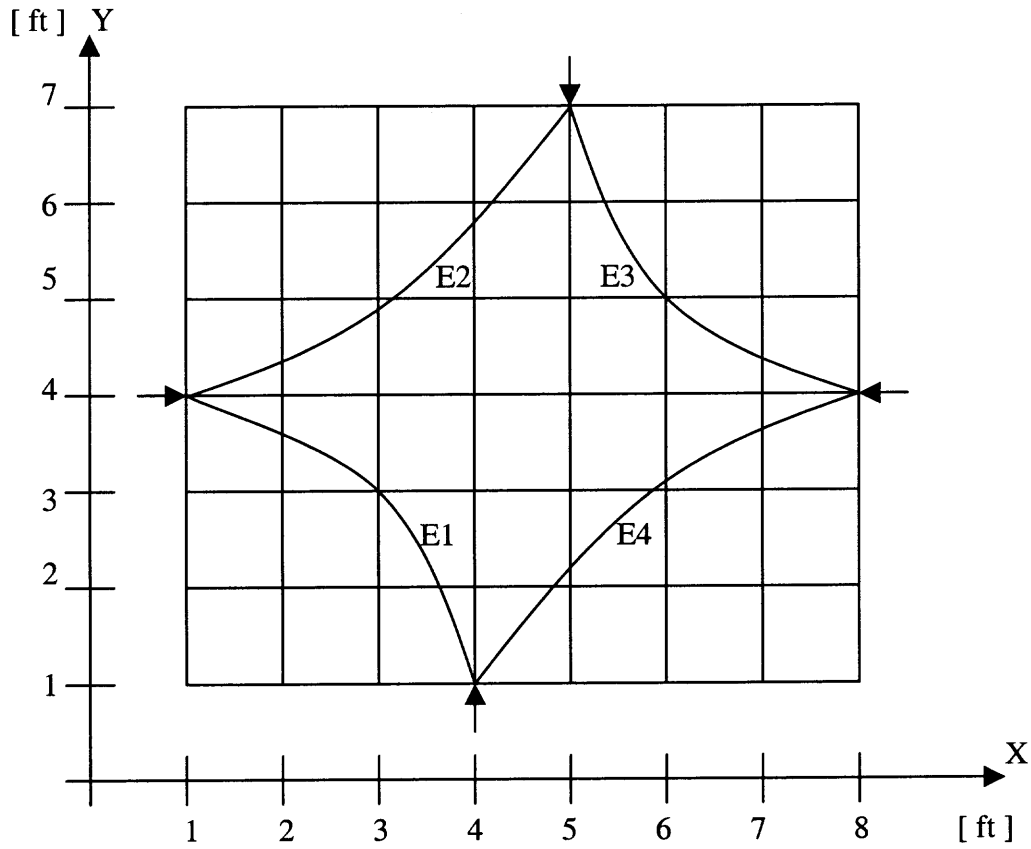

Figure 4.9 Grid of fabric structure with edge cable for example computation

Determine the geometry of the edge cables:

The geometry of the edge cables is defined by the coordinates of the two fixed points (x1, y1, x2 and y2) and the maximum displacement f. The section 4.3.2 shows how the coordinates of a third point (middle point of the edge cable) can be computed. The definition of the maximum displacement f (positive or negative) and the equation for the computation of the coordinates of the third point (x3 and y3) can be found in section

4.3.2 of this thesis. The following list determines the coordinates of the fixed points, the maximum displacement of the edge cable and shows the computed coordinates of the middle point of the edge cable. This list defines the geometry of all four edge cables in this example problem (fabric structure with edge cable).

Edge cable 1 :

Fixed point 1	=>	x1 = 1	y1 = 4
Fixed point 2	=>	x2 = 4	y2 = 1
Max. Displacement	=>	f = 0.5	
Computed 3rd Point	=>	x3 = 2.854	y3 = 2.854

Edge cable 2 :

Fixed point 1	=>	x1 = 1	y1 = 4
Fixed point 2	=>	x2 = 5	y2 = 7
Max. Displacement	=>	f = -0.5	
Computed 3rd Point	=>	x3 = 3.3	y3 = 5.1

Edge cable 3 :

Fixed point 1	=>	x1 = 5	y1 = 7
Fixed point 2	=>	x2 = 8	y2 = 4
Max. Displacement	=>	f = -0.5	
Computed 3rd Point	=>	x3 = 6.146	y3 = 5.146

Edge cable 4 :

Fixed point 1	=>	x1 = 4	y1 = 1
Fixed point 2	=>	x2 = 8	y2 = 4
Max. Displacement	=>	f = 0.5	
Computed 3rd Point	=>	x3 = 5.7	y3 = 2.9

The java program edgecableApp.java and the above listed variables for the edge cables are used for the computation of the node coordinates of the edge cable 1, 2, 3, and 4 (boundary nodes of the grid). The output of the computer code edgecableApp.java and the results (coordinates of the boundary node) are documented in the following section.

4.7.1 Computation of the Boundary Node Coordinates

<u>Computation of node coordinates of edge cable 1</u>

```
C:\Documents and Settings\ab52>java edgecableApp
=============================================================
Fixed points (supports) of the edge cable:
Point 1:  x1 = 1.0  y1 = 4.0
Point 2:  x2 = 4.0  y2 = 1.0
Middle point of the cable (not fixed!!!):
Middle Point:  xm = 2.854  ym = 2.854
=============================================================
Geometry function y(x) for the edge cable
y(x) = a*x*x + b*x + c
```

```
Function variables:
a= -0.33322602325804596  b= 0.6661301162902298  c = 3.667095906967816
ymax = 4.0 ymin = 1.0
==================================================================
Geometry function x(y) for the edge cable
x(y) = a*y*y + b*y + c
Function variables:
a= -0.3332260232580469  b= 0.6661301162902346  c = 3.667095906967812
xmax = 4.0 xmin = 1.0
==================================================================
Intersections of edge cable with horizontal basic grid members
Horizontal Edge Point # 1
xh = 4.0  yh = 1.0
Horizontal Edge Point # 2
xh = 3.449401117175175  yh = 2.0
Horizontal Edge Point # 3
xh = 2.7318466911569423  yh = 3.0
Horizontal Edge Point # 4
ambiguous result:
result 1:  xh = 1.0000000000000049  yh = 4.0
result 2:  xh = 0.9990338983050719  yh = 4.0
==================================================================
Intersections of edge cable with vertical basic grid members
Vertical Edge Point # 1
xv = 1.0  yv = 4.0
Vertical Edge Point # 2
xv = 2.0  yv = 3.449401117175176
Vertical Edge Point # 3
xv = 3.0  yv = 2.731846691156944
Vertical Edge Point # 4
ambiguous result:
result 1:  xv = 4.0  yv = 0.9999999999994917
result 2:  xv = 4.0  yv = 0.9990338983055936
==================================================================
```

Node coordinates of edge cable 1:

- Boundary point 1: x = 4 y = 1
- Boundary point 2: x = 3.45 y = 2
- Boundary point 3: x = 3 y = 2.73
- Boundary point 4: x = 2.73 y = 3
- Boundary point 5: x = 2 y = 3.45
- Boundary point 6: x = 1 y = 4

Computation of node coordinates of edge cable 2

```
C:\Documents and Settings\ab52>java edgecableApp
==================================================================
Fixed points (supports) of the edge cable:
Point 1:  x1 = 1.0  y1 = 4.0
Point 2:  x2 = 5.0  y2 = 7.0
Middle point of the cable (not fixed!!!):
Middle Point:  xm = 3.3  ym = 5.1
==================================================================
Geometry function y(x) for the edge cable
y(x) = a*x*x + b*x + c
Function variables:
```

```
a= 0.1598465473145776  b= -0.20907928388746627  c = 4.049232736572889
ymax = 7.0 ymin = 4.0
==================================================================
Geometry function x(y) for the edge cable
x(y) = a*y*y + b*y + c
Function variables:
a= -0.39872408293461015  b= 5.719298245614046  c = -15.49760765550242
xmax = 5.0 xmin = 1.0
==================================================================
Intersections of edge cable with horizontal basic grid members
No edge cable at y = 1.0
No edge cable at y = 2.0
No edge cable at y = 3.0
Horizontal Edge Point # 4
xh = 0.9999999999999984  yh = 4.0
Horizontal Edge Point # 5
xh = 3.1790180197376805  yh = 5.0
Horizontal Edge Point # 6
xh = 4.308112547458226  yh = 6.0
Horizontal Edge Point # 7
xh = 5.000000000000002  yh = 7.0
==================================================================
Intersections of edge cable with vertical basic grid members
Vertical Edge Point # 1
xv = 1.0  yv = 4.0
Vertical Edge Point # 2
xv = 2.0  yv = 4.423621568997456
Vertical Edge Point # 3
xv = 3.0  yv = 4.925762256572112
Vertical Edge Point # 4
xv = 4.0  yv = 5.679020401888331
Vertical Edge Point # 5
xv = 5.0  yv = 6.99999999999998
==================================================================
```

Node coordinates of edge cable 2:

- Boundary point 6: x = 1 y = 4
- Boundary point 7: x = 2 y = 4.42
- Boundary point 8: x = 3 y = 4.93
- Boundary point 9: x = 3.18 y = 5
- Boundary point 10: x = 4 y = 5.68
- Boundary point 11: x = 4.31 y = 6
- Boundary point 12: x = 5 y = 7

Computation of node coordinates of edge cable 3

```
C:\Documents and Settings\ab52>java edgecableApp
==================================================================
Fixed points (supports) of the edge cable:
Point 1:  x1 = 5.0  y1 = 7.0
Point 2:  x2 = 8.0  y2 = 4.0
Middle point of the cable (not fixed!!!):
Middle Point:  xm = 6.146  ym = 5.146
==================================================================
Geometry function y(x) for the edge cable
y(x) = a*x*x + b*x + c
```

```
Function variables:
a= 0.3332260232580467  b= -5.331938302354607  c = 25.32904093032187
ymax = 7.0 ymin = 4.0
==================================================================
Geometry function x(y) for the edge cable
x(y) = a*y*y + b*y + c
Function variables:
a= 0.33322602325804734  b= -4.66548625583852  c = 21.330328651225322
xmax = 8.0 xmin = 5.0
==================================================================
Intersections of edge cable with horizontal basic grid members
No edge cable at y = 1.0
No edge cable at y = 2.0
No edge cable at y = 3.0
Horizontal Edge Point # 4
ambiguous result:
result 1:  xh = 7.999999999999217  yh = 4.0
result 2:  xh = 8.0009661016957  yh = 4.0
Horizontal Edge Point # 5
xh = 6.268153308843056  yh = 5.0
Horizontal Edge Point # 6
xh = 5.550598882824825  yh = 6.0
Horizontal Edge Point # 7
xh = 5.000000000000001  yh = 7.0
==================================================================
Intersections of edge cable with vertical basic grid members
No edge cable at x = 1.0
No edge cable at x = 2.0
No edge cable at x = 3.0
No edge cable at x = 4.0
Vertical Edge Point # 5
ambiguous result:
result 1:  xv = 5.0  yv = 7.00000000001577
result 2:  xv = 5.0  yv = 7.000966101679138
Vertical Edge Point # 6
xv = 6.0  yv = 5.268153308843057
Vertical Edge Point # 7
xv = 7.0  yv = 4.550598882824824
Vertical Edge Point # 8
xv = 8.0  yv = 4.000000000000001
==================================================================
```

Node coordinates of edge cable 3:

- Boundary point 12: x = 5 y = 7
- Boundary point 13: x = 5.55 y = 6
- Boundary point 14: x = 6 y = 5.27
- Boundary point 15: x = 6.27 y = 5
- Boundary point 16: x = 7 y = 4.55
- Boundary point 17: x = 8 y = 4

Computation of node coordinates of edge cable 4

```
C:\Documents and Settings\ab52>java edgecableApp
==================================================================
Fixed points (supports) of the edge cable:
Point 1:  x1 = 4.0  y1 = 1.0
```

```
Point 2:  x2 = 8.0  y2 = 4.0
Middle point of the cable (not fixed!!!):
Middle Point:  xm = 5.7  ym = 2.9
==================================================================
Geometry function y(x) for the edge cable
y(x) = a*x*x + b*x + c
Function variables:
a= -0.15984654731457798  b= 2.6681585677749355  c = -7.1150895140664945
ymax = 4.0 ymin = 1.0
==================================================================
Geometry function x(y) for the edge cable
x(y) = a*y*y + b*y + c
Function variables:
a= 0.3987240829346095  b= -0.6602870813397134  c = 4.261562998405104
xmax = 8.0 xmin = 4.0
==================================================================
Intersections of edge cable with horizontal basic grid members
Horizontal Edge Point # 1
xh = 3.999999999999999  yh = 1.0
Horizontal Edge Point # 2
xh = 4.691887452541774  yh = 2.0
Horizontal Edge Point # 3
xh = 5.820981980262319  yh = 3.0
Horizontal Edge Point # 4
xh = 8.000000000000009  yh = 4.0
==================================================================
Intersections of edge cable with vertical basic grid members
No edge cable at x = 1.0
No edge cable at x = 2.0
No edge cable at x = 3.0
Vertical Edge Point # 4
xv = 4.0  yv = 0.999999999999999
Vertical Edge Point # 5
xv = 5.0  yv = 2.320979598111664
Vertical Edge Point # 6
xv = 6.0  yv = 3.0742377434278847
Vertical Edge Point # 7
xv = 7.0  yv = 3.576378431002542
Vertical Edge Point # 8
xv = 8.0  yv = 3.9999999999999996
==================================================================
```

Node coordinates of edge cable 4:

- Boundary point 17: x = 8 y = 4
- Boundary point 18: x = 7 y = 3.58
- Boundary point 19: x = 6 y = 3.07
- Boundary point 20: x = 5.82 y = 3
- Boundary point 21: x = 5 y = 2.32
- Boundary point 22: x = 4.69 y = 2
- Boundary point 23: x = 4 y = 1

4.7.1 Definition of Node and Member Numbers for the Grid

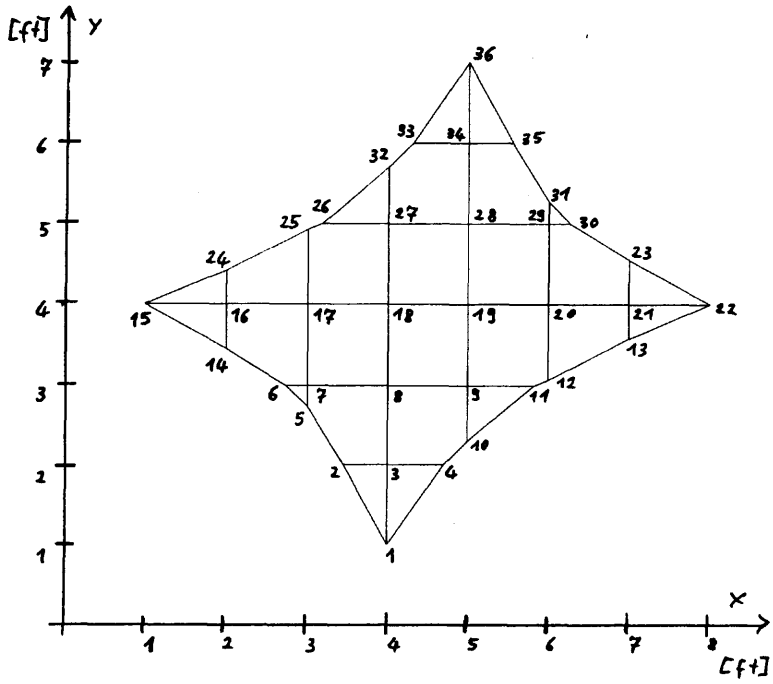

Figure 4.10 Define node numbers for grid of the fabric structure

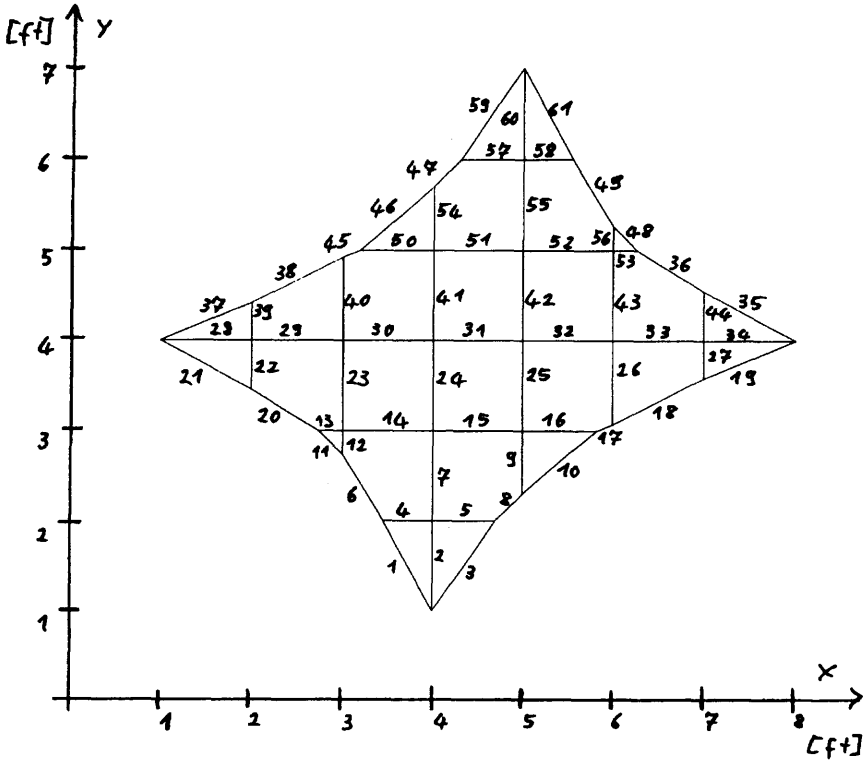

Figure 4.11 Define member numbers for grid of the fabric structure

4.7.3 Computation of the Forces in the Members of the Plane Grid

The forces in the members of the plane (2D) grid are computed using the principles and equations that are explained in section 4.5 (computation of the forces in boundary members of the 2D grid). In this example the geometry and static in edge cable 1 is equal to that in edge cable 3 and the geometry and static in edge cable 2 is equal to that in edge cable 4. Therefore we need only to compute the forces in the members of edge cable 1 and 2 in order to determine the stresses in all boundary members. Note that the results (member forces) are only an approach, which however are accurate enough for the shape finding process of fabric structure in this case.

Edge cable 1 = Edge cable 3
Edge cable 2 = Edge cable 4

Computation of the uniform load:

It is assumed that the fabric is pre-stressed by an uniform tension forces of 100 Kips/foot (line load). Therefore we get the following forces:

σ_x =100 Kips/foot σ_y = 100 Kips/foot

The uniform loading of the edge cable can be computed using equation 4.37:

Edge cable 1: $$\sigma = \sqrt{\left(\sigma_x * \frac{l_x}{l}\right)^2 + \left(\sigma_y * \frac{l_y}{l}\right)^2} = \sqrt{\left(100 * \frac{3}{4.24}\right)^2 + \left(100 * \frac{3}{4.24}\right)^2} = 100 \frac{Kips}{foot}$$

Edge cable 2: $$\sigma = \sqrt{\left(\sigma_x * \frac{l_x}{l}\right)^2 + \left(\sigma_y * \frac{l_y}{l}\right)^2} = \sqrt{\left(100 * \frac{3}{5}\right)^2 + \left(100 * \frac{4}{5}\right)^2} = 100 \frac{Kips}{foot}$$

Computation of the forces in the basic grid members:

The formulas for the computation of the forces in the basic grid members can be found in section 4.5 in step number (2):

$$F_X = \sigma_X * d_X = 100 \frac{Kips}{foot} * 1 foot = 100 Kips$$

$$F_Y = \sigma_Y * d_Y = 100 \frac{Kips}{foot} * 1 foot = 100 Kips$$

Computation of the support forces V and H:

Equation 4.35 and equation 4.36 can be used for the computation of the support forces V and H:

Edge cable 1: $H_1 = \frac{\sigma * L^2}{8 * f} = \frac{100 * 4.24^2}{8 * 0.5} = 449.4 Kips$ $\quad V_1 = \frac{100 * 4.24}{2} = 212 Kips$

Edge cable 2: $H_2 = \frac{\sigma * L^2}{8 * f} = \frac{100 * 5^2}{8 * 0.5} = 625 Kips$ $\quad V_2 = \frac{100 * 5}{2} = 250 Kips$

Convert support forces into one force in x-direction and one force in y-direction:

The support forces can be converted into one force in x-direction and one in y-direction using the equation 4.38 and equation 4.39. We convert the support forces of edge cable 1 at node 15 and the support forces of edge cable 2 at node 36 into forces in x-direction and y-direction:

Edge cable 1 at node 15:

$$Y_1 = H_y - V_y = H * \cos\alpha - V * \sin\alpha = 449.4 * \cos 45 - 212 * \sin 45 = 167.9 Kips$$

$$X_1 = H_x + V_x = H * \sin\alpha + V * \cos\alpha = 449.4 * \sin 45 + 212 * \cos 45 = 467.7 Kips$$

Edge cable 2 at node 36:

$$Y_2 = H_y + V_y = H * \sin\alpha + V * \cos\alpha = 625 * \sin 36.9 + 250 * \cos 36.9 = 575.2 Kips$$

$$X_2 = H_x - V_x = H * \cos\alpha - V * \sin\alpha = 625 * \cos 36.9 - 250 * \sin 36.9 = 349.7 Kips$$

Compute the forces in the members of edge cable 1:

The loaded and deflected edge cable 1 is symmetric. Therefore stresses in the edge cable are symmetric as well. In other words, the tension stress in member 21 is equal to the tension stress in member 1 and the tension stress in member 20 is equal to the tension stress in member 6. Therefore only the half of the members must be analyzed, because the stresses in the members of the other half the edge cable 1 are the same.

$S_{21} = S_1$ => stresses in grid member 21 and 1 are the same
$S_{20} = S_6$ => stresses in grid member 20 and 6 are the same

Compute tension force in member 21:

$$S_{21} = \sqrt{\left(X_1 - \frac{F_x}{2}\right)^2 + (Y_1)^2} = \sqrt{(467.7 - 50)^2 + (167.9)^2} = 450.2 Kips$$

Compute tension force in member 20:

$$S_{20} = \sqrt{\left(X_1 - \frac{F_x}{2}\right)^2 + \left(Y_1 + F_y\right)^2} = \sqrt{(467.7-50)^2 + (167.9+100)^2} = 496.2Kips$$

Compute tension force in member 11:

$$S_{11} = \sqrt{\left(X_1 - \frac{3*F_x}{2}\right)^2 + \left(Y_1 + F_y\right)^2} = \sqrt{(467.7-150)^2 + (167.9+100)^2} = 415.6Kips$$

Compute tension force in member 6:

$$S_6 = S_{20} = 496.2Kips$$

Compute tension force in member 1:

$$S_1 = S_{21} = 450.2Kips$$

<u>Compute the forces in the members of edge cable 2:</u>

The stresses in the edge cable 2 are symmetric. This means that the stress in member 37 is equal to the stress in member 59, the stress in member 38 is equal to the stress in member 47 and the stress in member 45 is equal to the stress in member 46. Therefore only the half of the members must be analyzed, because the stresses in the members of the other half the edge cable 2 are the same.

$S_{59} = S_{37}$	=>	stresses in grid member 37 and 59 are the same
$S_{47} = S_{38}$	=>	stresses in grid member 38 and 47 are the same
$S_{46} = S_{45}$	=>	stresses in grid member 45 and 46 are the same

Compute tension force in member 59:

$$S_{59} = \sqrt{(X_2)^2 + \left(Y_2 - \frac{F_y}{2}\right)^2} = \sqrt{(349.7)^2 + (575.2-50)^2} = 631Kips$$

Compute tension force in member 47:

$$S_{47} = \sqrt{(X_2 + F_x)^2 + \left(Y_2 - \frac{F_y}{2}\right)^2} = \sqrt{(349.7+100)^2 + (575.2-50)^2} = 691.4Kips$$

Compute tension force in member 46:

$$S_{46} = \sqrt{(X_2 + F_x)^2 + \left(Y_2 + \frac{3F_y}{2}\right)^2} = \sqrt{(349.7+100)^2 + (575.2-150)^2} = 618.9Kips$$

Compute tension force in member 45:

$$S_{45} = S_{46} = 618.9Kips$$

Compute tension force in member 38:

$$S_{38} = S_{47} = 691.4Kips$$

Compute tension force in member 37:

$$S_{37} = S_{59} = 631Kips$$

<u>Compute the forces in the members of edge cable 3:</u>

The stresses in the members of edge cable 3 are the same as in edge cable 1. In other words, edge cable 1 = edge cable 3:

$$S_{61} = S_{35} = S_{21} = 450.2Kips$$
$$S_{49} = S_{36} = S_{20} = 496.2Kips$$
$$S_{48} = S_{11} = 415.6Kips$$

<u>Compute the forces in the members of edge cable 4:</u>

The stresses in the members of edge cable 3 are the same as in edge cable 1. In other words, edge cable 1 = edge cable 2:

$$S_{19} = S_3 = S_{59} = 631Kips$$
$$S_{18} = S_8 = S_{47} = 691.4Kips$$
$$S_{17} = S_{10} = S_{46} = 618.9Kips$$

<u>Forces in the grid members that define not edge cables:</u>

The tension force inside the other grid members is 100 Kips in each member.

4.7.4 Input Section of the Example Computation Using Code edgeshapeApp.java

```
// input section!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

// coordinates of grid nodes
x[1] = 4;               y[1] = 1;
x[2] = 3.45;            y[2] = 2;
x[3] = 4;               y[3] = 2;
x[4] = 4.69;            y[4] = 2;
x[5] = 3;               y[5] = 2.73;
x[6] = 2.73;            y[6] = 3;
x[7] = 3;               y[7] = 3;
x[8] = 4;               y[8] = 3;
x[9] = 5;               y[9] = 3;
x[10] = 5;              y[10] = 2.32;
x[11] = 5.82;           y[11] = 3;
x[12] = 6;              y[12] = 3.07;
x[13] = 7;              y[13] = 3.58;
x[14] = 2;              y[14] = 3.45;
x[15] = 1;              y[15] = 4;
x[16] = 2;              y[16] = 4;
x[17] = 3;              y[17] = 4;
x[18] = 4;              y[18] = 4;
x[19] = 5;              y[19] = 4;
x[20] = 6;              y[20] = 4;
x[21] = 7;              y[21] = 4;
x[22] = 8;              y[22] = 4;
x[23] = 7;              y[23] = 4.55;
x[24] = 2;              y[24] = 4.42;
x[25] = 3;              y[25] = 4.93;
x[26] = 3.18;           y[26] = 5;
x[27] = 4;              y[27] = 5;
x[28] = 5;              y[28] = 5;
x[29] = 6;              y[29] = 5;
x[30] = 6.27;           y[30] = 5;
x[31] = 6;              y[31] = 5.27;
x[32] = 4;              y[32] = 5.68;
x[33] = 4.31;           y[33] = 6;
x[34] = 5;              y[34] = 6;
x[35] = 5.55;           y[35] = 6;
x[36] = 5;              y[36] = 7;

// grid members and their pre-stressed forces
mb[1] = 1;              me[1] = 2;              f[1] = 450.2;           // edge cable
mb[2] = 1;              me[2] = 3;              f[2] = 100;
mb[3] = 1;              me[3] = 4;              f[3] = 631;             // edge cable
mb[4] = 2;              me[4] = 3;              f[4] = 100;
mb[5] = 3;              me[5] = 4;              f[5] = 100;
mb[6] = 2;              me[6] = 5;              f[6] = 496.2;           // edge cable
mb[7] = 3;              me[7] = 8;              f[7] = 100;
mb[8] = 4;              me[8] = 10;             f[8] = 691.4;           // edge cable
```

```
mb[9] = 10;     me[9] = 9;      f[9] = 100;
mb[10] = 10;    me[10] = 11;    f[10] = 618.9;    // edge cable
mb[11] = 5;     me[11] = 6;     f[11] = 415.6;    // edge cable
mb[12] = 5;     me[12] = 7;     f[12] = 100;
mb[13] = 6;     me[13] = 7;     f[13] = 100;
mb[14] = 7;     me[14] = 8;     f[14] = 100;
mb[15] = 8;     me[15] = 9;     f[15] = 100;
mb[16] = 9;     me[16] = 11;    f[16] = 100;
mb[17] = 11;    me[17] = 12;    f[17] = 618.9;    // edge cable
mb[18] = 12;    me[18] = 13;    f[18] = 691.4;    // edge cable
mb[19] = 13;    me[19] = 22;    f[19] = 631;      // edge cable
mb[20] = 6;     me[20] = 14;    f[20] = 496.2;    // edge cable
mb[21] = 14;    me[21] = 15;    f[21] = 450.2;    // edge cable
mb[22] = 14;    me[22] = 16;    f[22] = 100;
mb[23] = 7;     me[23] = 17;    f[23] = 100;
mb[24] = 8;     me[24] = 18;    f[24] = 100;
mb[25] = 9;     me[25] = 19;    f[25] = 100;
mb[26] = 12;    me[26] = 20;    f[26] = 100;
mb[27] = 13;    me[27] = 21;    f[27] = 100;
mb[28] = 15;    me[28] = 16;    f[28] = 100;
mb[29] = 16;    me[29] = 17;    f[29] = 100;
mb[30] = 17;    me[30] = 18;    f[30] = 100;
mb[31] = 18;    me[31] = 19;    f[31] = 100;
mb[32] = 19;    me[32] = 20;    f[32] = 100;
mb[33] = 20;    me[33] = 21;    f[33] = 100;
mb[34] = 21;    me[34] = 22;    f[34] = 100;
mb[35] = 22;    me[35] = 23;    f[35] = 450.2;    // edge cable
mb[36] = 23;    me[36] = 30;    f[36] = 496.2;    // edge cable
mb[37] = 15;    me[37] = 24;    f[37] = 631;      // edge cable
mb[38] = 24;    me[38] = 25;    f[38] = 691.4;    // edge cable
mb[39] = 16;    me[39] = 24;    f[39] = 100;
mb[40] = 17;    me[40] = 25;    f[40] = 100;
mb[41] = 18;    me[41] = 27;    f[41] = 100;
mb[42] = 19;    me[42] = 28;    f[42] = 100;
mb[43] = 20;    me[43] = 29;    f[43] = 100;
mb[44] = 21;    me[44] = 23;    f[44] = 100;
mb[45] = 25;    me[45] = 26;    f[45] = 618.9;    // edge cable
mb[46] = 26;    me[46] = 32;    f[46] = 618.9;    // edge cable
mb[47] = 32;    me[47] = 33;    f[47] = 691.4;    // edge cable
mb[48] = 30;    me[48] = 31;    f[48] = 415.6;    // edge cable
mb[49] = 31;    me[49] = 35;    f[49] = 496.2;    // edge cable
mb[50] = 26;    me[50] = 27;    f[50] = 100;
mb[51] = 27;    me[51] = 28;    f[51] = 100;
mb[52] = 28;    me[52] = 29;    f[52] = 100;
mb[53] = 29;    me[53] = 30;    f[53] = 100;
mb[54] = 27;    me[54] = 32;    f[54] = 100;
mb[55] = 28;    me[55] = 34;    f[55] = 100;
mb[56] = 29;    me[56] = 31;    f[56] = 100;
mb[57] = 33;    me[57] = 34;    f[57] = 100;
mb[58] = 34;    me[58] = 35;    f[58] = 100;
mb[59] = 33;    me[59] = 36;    f[59] = 631;      // edge cable
```

```
mb[60] = 34;        me[60] = 36;        f[60] = 100;
mb[61] = 35;        me[61] = 36;        f[61] = 450.2;        // edge cable

        // initialize the other arrays
                for( int a = 1; a <= an; a++ ) {
                        v[a] = 0;
                        z[a] = 0;
                        t[a] = 0;
                        h[a] = 0;
                        for ( int b = 1; b <= an; b++ ) {
                                A[a][b] = 0;
                        }
                }

// input level of the fix points
v[1] = 100;
v[15] = 102;
v[22] = 102;
v[36] = 100;
```

4.7.5 Output of the Example Computation Using Code edgeshapeApp.java

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\ab52>javac edgeshapeApp.java
C:\Documents and Settings\ab52>java edgeshapeApp

Generation of Grid was successful
Output of the Node-Coordinates:

Node 1
X = 4.0         Y = 1.0         Z = 0.0
Node 2
X = 3.45        Y = 2.0         Z = 0.5317488249973934
Node 3
X = 4.0         Y = 2.0         Z = 0.4754092944211834
Node 4
X = 4.69        Y = 2.0         Z = 0.4956327319131333
Node 5
X = 3.0         Y = 2.73        Z = 0.9119690460518228
Node 6
X = 2.73        Y = 3.0         Z = 1.0854443371587053
Node 7
X = 3.0         Y = 3.0         Z = 0.9961812264333219
Node 8
X = 4.0         Y = 3.0         Z = 0.8190737490053692
Node 9
X = 5.0         Y = 3.0         Z = 0.8601145305140818
```

Node 10
X = 5.0 Y = 2.32 Z = 0.6633969370665938
Node 11
X = 5.82 Y = 3.0 Z = 1.0617164062088023
Node 12
X = 6.0 Y = 3.07 Z = 1.141603325481654
Node 13
X = 7.0 Y = 3.58 Z = 1.554975249377108
Node 14
X = 2.0 Y = 3.45 Z = 1.468897610050604
Node 15
X = 1.0 Y = 4.0 Z = 2.0
Node 16
X = 2.0 Y = 4.0 Z = 1.5369285771327554
Node 17
X = 3.0 Y = 4.0 Z = 1.1545815544393463
Node 18
X = 4.0 Y = 4.0 Z = 0.9445899446528898
Node 19
X = 5.0 Y = 4.0 Z = 0.9445899446528756
Node 20
X = 6.0 Y = 4.0 Z = 1.1545815544394031
Node 21
X = 7.0 Y = 4.0 Z = 1.5369285771327839
Node 22
X = 8.0 Y = 4.0 Z = 2.0
Node 23
X = 7.0 Y = 4.55 Z = 1.468897610050604
Node 24
X = 2.0 Y = 4.42 Z = 1.5549752493770512
Node 25
X = 3.0 Y = 4.93 Z = 1.1416033254815119
Node 26
X = 3.18 Y = 5.0 Z = 1.0617164062086601
Node 27
X = 4.0 Y = 5.0 Z = 0.8601145305140108
Node 28
X = 5.0 Y = 5.0 Z = 0.819073749005355
Node 29
X = 6.0 Y = 5.0 Z = 0.9961812264332792
Node 30
X = 6.27 Y = 5.0 Z = 1.0854443371586484
Node 31
X = 6.0 Y = 5.27 Z = 0.9119690460517944
Node 32
X = 4.0 Y = 5.68 Z = 0.6633969370665227
Node 33
X = 4.31 Y = 6.0 Z = 0.49563273191307644
Node 34
X = 5.0 Y = 6.0 Z = 0.475409294421155

```
Node 35
X = 5.55        Y = 6.0        Z = 0.5317488249973934
Node 36
X = 5.0         Y = 7.0        Z = 0.0

C:\Documents and Settings\ab52>
```

The output of the java code edgeshapeApp.java displays the results of the grid method (shape finding process) for the fabric structures. The three dimensional coordinates of the grid nodes form together the shape of the fabric structure (surface of the fabric). The spatial node coordinates and the grid members (which combine the single node) create together the three dimensional grid that reflects the shape of the structure. The result of these computation (grid method using code edgeshape.java) are the spatial shape of the fabric structure with edge cable, which can be used as input for the static and dynamic analysis of this fabric structure with edge cable.

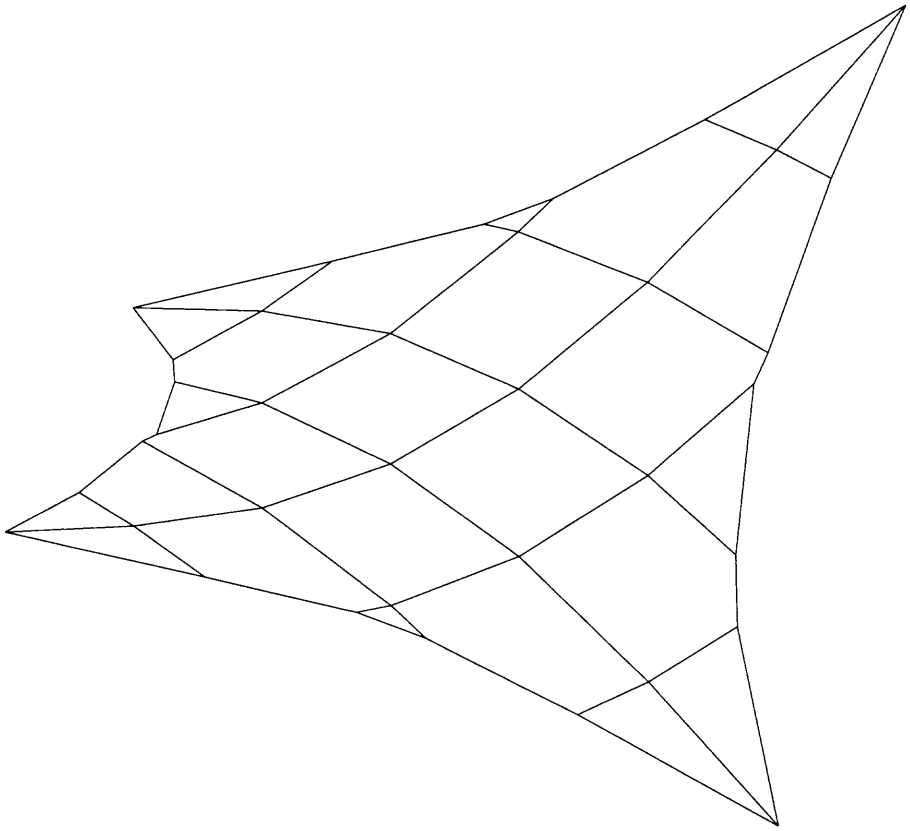

Figure 4.12 Shape of fabric structure with edge cable (result of computation)

CHAPTER 5

NONLINEAR STATIC ANALYSIS OF FABRIC STRUCTURES

5.1 The Mathematical Methods to Perform a Nonlinear Analysis

The nonlinear equilibrium conditions of the fabric structures can be expressed in ordinary differential equations. The nonlinear static (and dynamic) analysis of the fabric structures is nothing else than solving these ordinary differential equations, which define the nonlinear equilibrium conditions. There are several mathematical methods that can be used to solve the system. Three of these methods are:

- Euler's method (incremental solution)
- Newton's method (iterative solution)
- Combined Euler/Newton (combined incremental/iterative solution)

The principles of the mathematical methods Euler's method (incremental solution) and Newton's method (incremental solution) are explained in the following two sections. Later in this chapter are explanations about how these principles are applied in the nonlinear static analysis of the fabric structures. The combined incremental/iterative method just combines these two methods and is therefore not extra explained in this chapter.

5.1.1 The Principles of the Euler's Method (Incremental Solution)

The Euler's method is easy to understand, apply and code up. The solution curve y(x) of the differential equation y' = f(x;y) with the initial value $y(x_0) = y_0$ can be computed for incremental steps. The spacing between every step is equal to h. Therefore the solution curve is computed at the following steps:

$x_1 = x_0 + h$ $\quad$ $x_2 = x_0 + 2h$ $\quad$ $x_3 = x_0 + 3h$ $\quad$ etc.

The y-values at these x-coordinates can be computed by the following approaches. The computed y-values are only approximations and it must be proved, if the computed y-values are accurate enough:

$y(x_0) = y_0$ => initial start value $\qquad$ $y' = f(x;y)$

$$y(x_1) \approx y_1 = y_0 + h * f(x_0; y_0) \qquad \text{(Equation 5.1)}$$
$$y(x_2) \approx y_2 = y_1 + h * f(x_1; y_1) \qquad \text{(Equation 5.2)}$$
$$y(x_3) \approx y_3 = y_2 + h * f(x_2; y_2) \qquad \text{(Equation 5.3)}$$

etc.

Therefore we get the following computation scheme:

Step 0:	x_0	$y(x_0) = y_0$
Step 1:	$x_1 = x_0 + h$	$y(x_1) \approx y_1 = y_0 + h * f(x_0; y_0)$
Step 2:	$x_2 = x_1 + h$	$y(x_2) \approx y_2 = y_1 + h * f(x_1; y_1)$
Step 3:	$x_3 = x_2 + h$	$y(x_3) \approx y_3 = y_2 + h * f(x_2; y_2)$
	etc.	

where: $y' = f(x; y)$

The error of this computation can be estimated by the following equation:

$$\Delta y_i = y(x_i) - y_i \approx y_i - \tilde{y}_i \qquad \text{(Equation 5.4)}$$

where:
$y(x_i)$ = exact solution at the x-coordinate x_i
y_i = approached solution at x_i with step-size = h
$\tilde{y}_i$ = approached solution at x_i with step-size = 2h

It must be proved for every nonlinear analysis using Euler's method, if the maximum error can be neglected. A smaller step size must be used in order to get a more accurate solution, if the error is too large.

5.1.2 The Principles of the Newton's Method (Iterative Solution)

The Newton's method solves a nonlinear equation system by linearizing the system. The Newton's method solves the system by several iterative steps. Every step starts typically at a given point x_i and computes the new point x_{i+1} using a created linearized system. The computation leads to a result as soon as enough iteration steps are performed to get a convergence of the system. It has to be proved, if the solution is accurate enough and the error is small enough to be neglected. The Newton's method shows usually rapid convergence.
Suppose we want to solve the following system of nonlinear equations using Newton's method:

$$f(x, y) = 0 \qquad \text{(Equation 5.5)}$$
$$g(x, y) = 0 \qquad \text{(Equation 5.6)}$$

Perform a derivation similar to that of Newton's method with one equation and variable (recall f(x) = 0 can be solved using $x_{n+1} = x_n - (f(x_n) / f'(x_n))$ => Newton's method for one variable):

$$f(x + \Delta x, y + \Delta y) = f(x, y) + \Delta x * f_x(x, y) + \Delta y * f_y(x, y) + ... \qquad \text{(Equation 5.7)}$$
$$g(x + \Delta x, y + \Delta y) = g(x, y) + \Delta x * g_x(x, y) + \Delta y * g_y(x, y) + ... \qquad \text{(Equation 5.8)}$$

Try to find Δx and Δy such that f(x+Δx, y+Δy) and g(x+Δx, y+Δy) = 0. This can be done by solving the following system:

$$\begin{pmatrix} -f(x,y) \\ -g(x,y) \end{pmatrix} = \begin{pmatrix} f_X(x,y) & f_Y(x,y) \\ g_X(x,y) & g_Y(x,y) \end{pmatrix} * \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \qquad \text{(Equation 5.9)}$$

Where:

$$\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \frac{1}{f_X g_Y - g_X f_Y} * \begin{pmatrix} -g_Y(x,y) & f_Y(x,y) \\ g_X(x,y) & -f_X(x,y) \end{pmatrix} * \begin{pmatrix} f(x,y) \\ g(x,y) \end{pmatrix}$$

A few important points when using the Newton's method:

- A good starting guess is needed. We usually can only find one on the basis of trial and error. It is difficult to graph functions of more than one variable to approximate a solution and obtain a reasonably good starting guess.
- Lots of (partial) derivatives are needed. One variation is to update the iteration system only every few steps or to used numerical derivatives to approximate the iteration system
- Another variation is to consider $f^2 = g^2 = \ldots = 0$ and use methods to minimize a function of many variables, such as steepest decent or conjugate gradient.

5.2 Nonlinear Static Analysis of Fabric Structures

The reason for the need of a nonlinear static analysis when designing fabric structures and the principles of the nonlinear static analysis is already argued in chapter 1.5 of this thesis. The node displacement method is used to define the nonlinear equilibrium and the computation of the fabric structure. The following equations describe the equilibrium conditions using the displacement method for a nonlinear analysis of the fabric structure.

Nonlinear equations for static analysis of fabric structures:

$$dN^T * F + N^T * dF = dP \qquad \text{(Equation 5.10)}$$

$$\Rightarrow \qquad K_E * \delta + K_G * \delta = P \qquad \text{(Equation 5.11)}$$

where:

K_E = elastic stiffness matrix (linear effects)
K_G = geometrical stiffness matrix (nonlinear effects)
P = node forces
F = member forces
N = equilibrium "operator" (unit vector)
δ = node displacement

This nonlinear equation for the static analysis of fabric structures is now interpreted for a typical member of a structure in order to show how the nonlinear analysis works. The following sketch shows the typical member unloaded (A, B) and loaded (Á, B́). It defines the different variables:

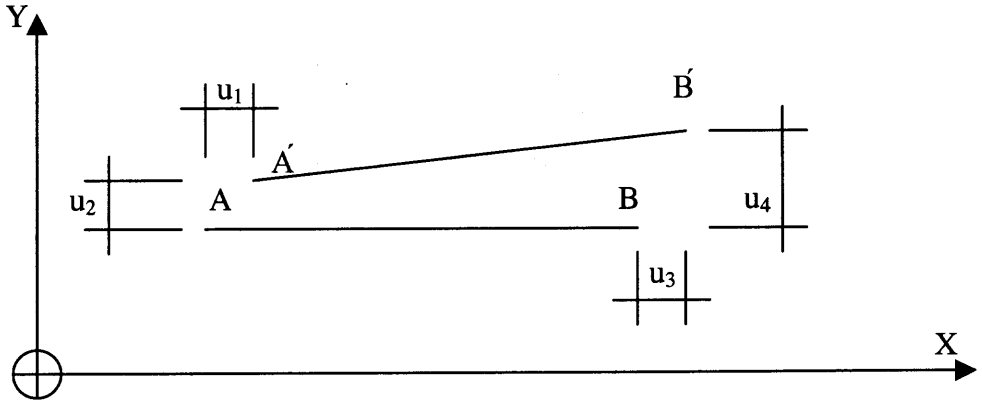

Figure 5.1 Typical member of structure with nonlinear behavior

The member with the end A and B are the ends of the original (unloaded) member while the member with the ends Á and B́ are the ends of the displaced (loaded) member. The variables u_i are defining the length of the node displacements. The equilibrium conditions of the member in figure 5.1 can be expressed by the following equations.

$$P = K_E * u + K_G * u \qquad \text{(Equation 5.12)}$$

$$\begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} = K_E * \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} + K_G * \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \qquad \text{(Equation 5.13)}$$

<u>Derive geometrical stiffness matrix K_G, which defines the nonlinear effects:</u>

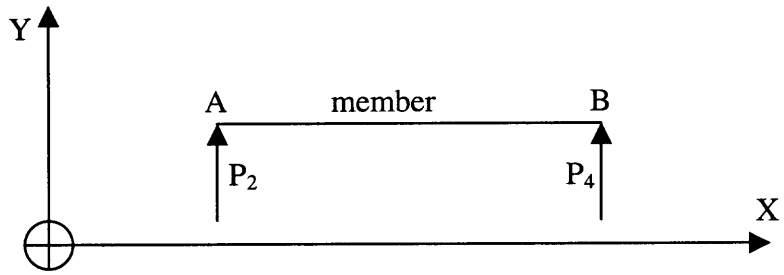

Figure 5.2 Nonlinear effects of the geometrical stiffness matrix

$$P_2 = \frac{F}{l} * (u_2 - u_4) \qquad \text{(Equation 5.14)}$$

$$P_4 = \frac{F}{l} * (u_4 - u_2) \qquad \text{(Equation 5.15)}$$

where:

F = force in the typical member
l = length of the typical member

The geometrical stiffness matrix for a typical member is:

$$K_G = \frac{F}{l} * \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \qquad \text{(Equation 5.16)}$$

<u>Derive elastic stiffness matrix K_E, which defines the linear effects:</u>

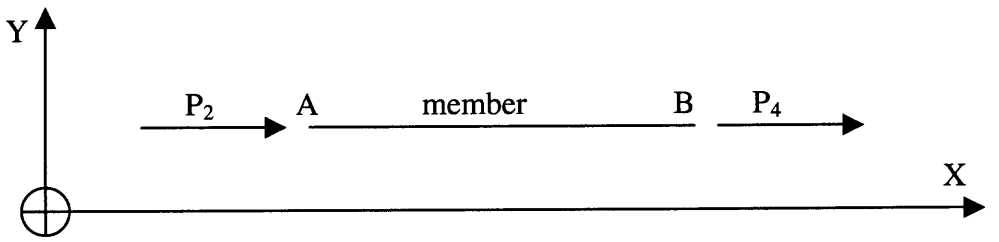

Figure 5.3 Linear effects of the elastic stiffness matrix

$$P_2 = \frac{A * E}{l} * (u_1 - u_3) \qquad \text{(Equation 5.17)}$$

$$P_4 = \frac{A * E}{l} * (u_3 - u_1) \qquad \text{(Equation 5.18)}$$

where:

A = cross section area of the typical member
E = Young's module of the typical member
l = length of the typical member

The elastic stiffness matrix for a typical member is:

$$K_E = \frac{A * E}{l} * \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \qquad \text{(Equation 5.19)}$$

5.3 Implementation of the Nonlinear Static Analysis

This section shows how to create the differential equations for the nonlinear static analysis. These equations must be defined in correspondence to the mathematical method that is used to solve the system of nonlinear equations. In the first part of this section the differential equations are defined for the iterative solution. This system of nonlinear equations can be solved by iterative mathematical methods. In the second part of this section the differential equations are defined for the incremental solution. This system of nonlinear equations can be solved by incremental mathematical methods.

Iterative solution:

$$\left[K_E + K_G\left(F^n\right)\right] * \delta^{(n+1)} = P - N^T * F^n \qquad \text{(Equation 5.20)}$$

$$\delta^{(n+1)} = \left[K_E + K_G\left(F^n\right)\right]^{-1} * \left[P - N^T * F^n\right] \qquad \text{(Equation 5.21)}$$

Incremental solution:

Step 1:

$$\Delta P = \frac{P}{i} = \left(K_{E(1)} + K_{G(1)}\right) * \Delta\delta_{(1)} \qquad \text{(Equation 5.22)}$$

$$\delta_{(1)} = \Delta\delta_{(1)} \qquad \text{(Equation 5.23)}$$

Step 2:

$$\Delta P = \frac{P}{i} = \left(K_{E(2)} + K_{G(2)}\right) * \Delta\delta_{(2)} \qquad \text{(Equation 5.24)}$$

$$\delta_{(2)} = \delta_{(1)} + \Delta\delta_{(2)} \qquad \text{(Equation 5.25)}$$

Step 3:

$$\Delta P = \frac{P}{i} = \left(K_{E(3)} + K_{G(3)}\right) * \Delta\delta_{(3)} \qquad \text{(Equation 5.26)}$$

$$\delta_{(3)} = \delta_{(2)} + \Delta\delta_{(3)} \qquad \text{(Equation 5.27)}$$

5.4 Stiffness Properties of the FEM-Element of the Membrane

The triangular plane stress/strain finite element of Zienkiewicz is used to model the membrane of the fabric structure. However, this plane stress/strain finite element for itself is only a two dimensional result for the problem. This two dimensional result does not include nonlinear effects, therefore a three dimensional finite element must be used. The three dimensional finite element can be constructed using the plane stress/strain finite element in connection with the rotation matrix.

The stiffness properties of the three dimensional finite element that is used to model the membrane (fabric) are defined by two stiffness matrixes. One elastic stiffness matrix represents that the linear effects of the fabric structure and one geometric stiffness matrix represents that the nonlinear effects of the fabric structure.

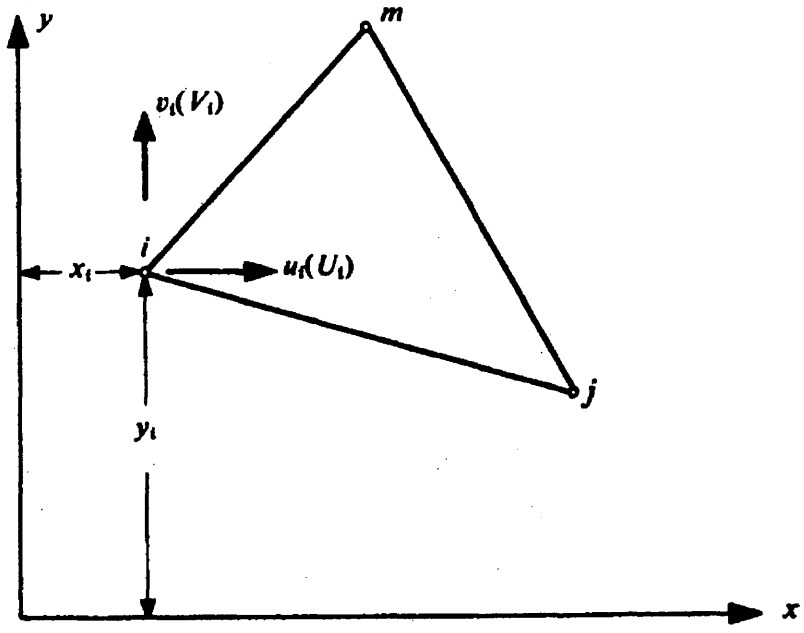

Figure 5.4 Plane stress/strain finite element

Elastic Stiffness Matrix

The following variables of the plane stress/strain finite element (figure 5.4) are defined first and are needed for the following computation of the elastic stiffness matrix:

$$b_i = y_j - y_m \qquad b_j = y_m - y_i \qquad b_k = y_i - y_j$$
$$c_i = x_m - x_j \qquad c_j = x_i - x_m \qquad c_k = x_j - x_i$$

The elastic stiffness matrix is the same for the linear and the nonlinear computations. Therefore the rotations of the three dimensional finite element has no effect to the elastic stiffness matrix of the 3D finite element, which is similar to the elastic stiffness matrix of the plane stress/strain finite element.

Elastic stiffness matrix of the 3D finite element:

$$K_E = \begin{bmatrix} (K_E)_{11} & (K_E)_{12} & (K_E)_{13} \\ (K_E)_{21} & (K_E)_{22} & (K_E)_{23} \\ (K_E)_{31} & (K_E)_{32} & (K_E)_{33} \end{bmatrix} \qquad \text{(Equation 5.28)}$$

where:

$$(K_E)_{RS} = \frac{t}{4*\Gamma} * \frac{E}{(1-\upsilon^2)} \begin{bmatrix} b_R b_S + \frac{1-\upsilon}{2} c_R c_S & b_R b_S \upsilon + \frac{1-\upsilon}{2} c_R b_S & 0 \\ b_R b_S \upsilon + \frac{1-\upsilon}{2} b_R c_S & c_R c_S + \frac{1-\upsilon}{2} b_R b_S & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

(Equation 5.29)

υ = Poisson's ratio
E = Young's modulus (E-modulus)
Γ = surface area of the triangular finite element facet

Geometric Stiffness Matrix

The geometric stiffness matrix of the 3D finite element has a nonlinear out-of-plane stiffness component. Therefore the rotations of the three dimensional finite element must be considered in the nonlinear case. The variables of the out-of-plane rotation of a plane finite element are shown in the sketch below.

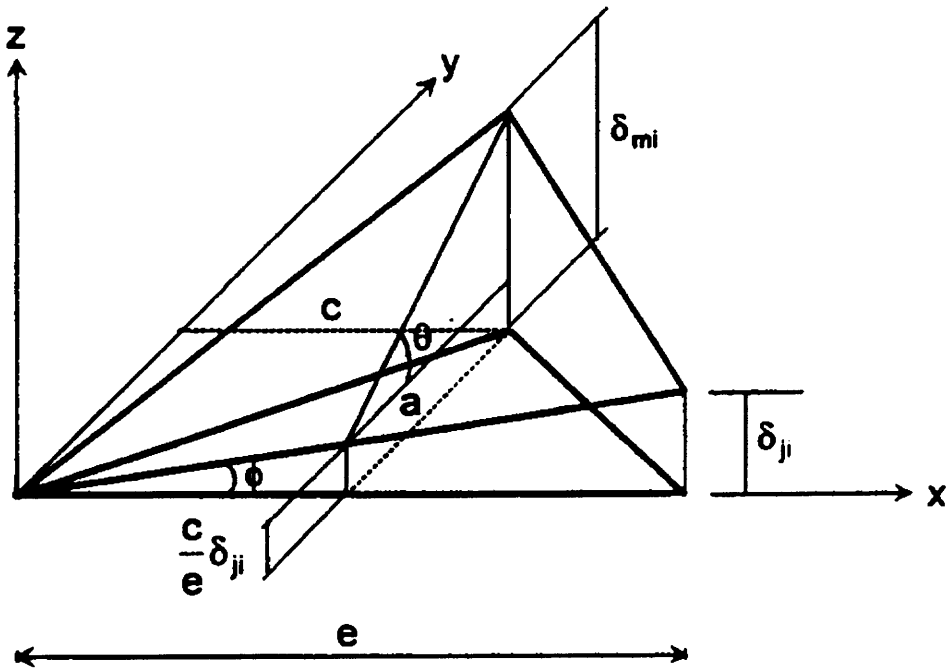

Figure 5.5 Out-of-plane rotations of plane finite element

The geometric stiffness matrix of a plane stress/strain finite element without out-of-plane rotations is the same for linear and nonlinear cases. It should be clear that the geometric stiffness matrix of the three dimensional finite element is nothing else as the geometric stiffness matrix of the plane stress/strain finite element (without rotations) added with a matrix that represents the effects of the out-of-plane rotations.

Geometric stiffness matrix of the 3D finite element:

$$K_G = \frac{t}{2} * \begin{bmatrix} 0 & 0 & 0 & -\tau & \sigma_X & 0 & \tau & -\sigma_X & 0 \\ 0 & 0 & 0 & -\sigma_Y & \tau & 0 & \sigma_Y & -\tau & 0 \\ 0 & 0 & \alpha_i & 0 & 0 & \beta_i & 0 & 0 & \lambda_i \\ \tau & -\sigma_X & 0 & 0 & 0 & 0 & -\tau & \sigma_X & 0 \\ \sigma_Y & -\tau & 0 & 0 & 0 & 0 & -\sigma_Y & \tau & 0 \\ 0 & 0 & \alpha_j & 0 & 0 & \beta_j & 0 & 0 & \lambda_j \\ -\tau & \sigma_X & 0 & \tau & -\sigma_X & 0 & 0 & 0 & 0 \\ -\sigma_Y & \tau & 0 & \sigma_Y & -\tau & 0 & 0 & 0 & 0 \\ 0 & 0 & \alpha_m & 0 & 0 & \beta_m & 0 & 0 & \lambda_m \end{bmatrix}$$

(Equation 5.30)

where:

t = thickness of plane stress/strain finite element

$$\alpha_r = -\frac{e-c}{ae}(F_r)_Y - \frac{1}{e}(F_r)_X \qquad \text{(Equation 5.31)}$$

$$\beta_r = -\frac{c}{ae}(F_r)_Y + \frac{1}{e}(F_r)_X \qquad \text{(Equation 5.32)}$$

$$\lambda_r = \frac{1}{a}(F_r)_Y \qquad \text{(Equation 5.33)}$$

for r = i, j, m, ...

This geometrical stiffness matrix includes the nonlinear geometric effects of the membrane of fabric structures. This geometrical stiffness matrix describes, together with the earlier defined elastic stiffness matrix, the stiffness properties of the 3D finite element of a membrane. The elastic stiffness matrix and the geometrical stiffness matrix, which are defined in chapter 5.4, are the foundation for the nonlinear static analysis of fabric structures.

APPENDIX A

FORTRAN CODE OF PROGRAM HLAY.FOR

```
C     PROGRAM HLAY.FOR
      include 'fgraph.fi'
C     GENERAL MEMBRANE LAYOUT PROGRAM
C     Grid Method
      DIMENSION FORH(999),X(500),Y(500),Z(500),JT(500,8),IFIX(500)
     1 ,NP(999),MI(999),ISYM(500),FOR(999),R(1500),ITYPE(1000)
      READ(50,1)NB,NN,NS,NSTEP,NIT,NF
      WRITE(60,1)NB,NN
    1 FORMAT(6I5)
C
READ(50,42)(X(I),Y(I),Z(I),IFIX(I),ISYM(I),(JT(I,J),J=1,8),I=1,NN)
      READ(50,*)(X(I),Y(I),Z(I),I=1,NN)
      write(60,42)(X(I),Y(I),Z(I),I=1,NN)
   42 FORMAT(5X,3F11.6)
      READ(50,*)(NP(I),MI(I),I=1,NB)
  222 FORMAT(5X,3I5,E20.8)
      WRITE(60,2)(I,NP(I),MI(I),I=1,NB)
    2 FORMAT(4I5,F10.3)
c 7778 WRITE(*,*) 'FIXED POINTS...NODE,ELEV'
c      READ(50,*) NOD,ELEV
c      IF(NOD.EQ.0) GO TO 7777
c      Z(NOD)=ELEV
c      GO TO 7778
 7777 CONTINUE
C      write(*.*) 'y axis symmetry?'
C      read(*.*) isym
C      if(isym.eq.0) go to 6667
       DO 7776 I=1,NB
C      IF(X(NP(I)).NE.0..OR.X(MI(I)).NE.0.) GO TO 7776
C      FORH(I)=FORH(I)/2.
       forh(i)=1.
 7776 CONTINUE
 6667 continue
      DO 7775 I=1,NN
      K=0
      DO 7775 J=1,NB
      IF(NP(J).NE.I) GO TO 7774
      K=K+1
      JT(I,K)=MI(J)
      GO TO 7775
 7774 IF(MI(J).NE.I) GO TO 7775
      K=K+1
      JT(I,K)=NP(J)
 7775 CONTINUE
      DO 44 I=1,NN
      R(3*I)=Z(I)
   44 IF(Z(I).NE.0.) IFIX(I)=1
      WRITE(60,333)(X(I),Y(I),Z(I),IFIX(I),ISYM(I),(JT(I,J),J=1,8),I=1,
     1 NN)
```

```
  333 FORMAT(3F10.3,2I2,8I5)
    3 FORMAT(3F10.3,2I2)
  678 FORMAT(5X,E20.8)
      NIT=200
      DO 6 ITER=1,NIT
      DO 5 I=1,NN
      IF(IFIX(I).NE.0) GO TO 5
      SUM=0.
      SUM1=0.
      DO 52 J=1,8
      IF(JT(I,J).EQ.0) GO TO 51
      K=JT(I,J)
      DX=ABS(X(I)-X(K))
      DY=ABS(Y(I)-Y(K))
      DL=SQRT(DX*DX+DY*DY)
      IF(DL.EQ.0.) GO TO 52
      IBAR=0
      DO 62 L=1,NB
      IF(NP(L).EQ.I.AND.MI(L).EQ.K) IBAR=L
      IF(NP(L).EQ.K.AND.MI(L).EQ.I) IBAR=L
      IF(IBAR.NE.0) GO TO 63
   62 CONTINUE
      WRITE(6,64)
      STOP
   63 CONTINUE
   64 FORMAT('DATA ERROR STOP')
      SUM=SUM+FORH(IBAR)/DL
      SUM1=SUM1+FORH(IBAR)*Z(K)/DL
   52 CONTINUE
   51 Z(I)=SUM1/SUM
C    5 WRITE(6,10)ITER,I,Z(I)
    5 CONTINUE
   10 FORMAT(2I5,F10.2)
    6 CONTINUE
      WRITE(8,24)
   24 FORMAT(///16X,'COORDINATES',24X,'FORCE BALLANCE'/
     1 16X,'X',11X,'Y',11X,'Z',11X,'X',11X,'Y',11X,'Z')
c     Check node equilibrium
c     Set up coords for plot routine
      DO 151 I=1,NN
      R(3*I-2)=X(I)
      R(3*I-1)=Y(I)
      R(3*I)=Z(I)
      SUMX=0.
      SUMY=0.
      SUMZ=0.
      DO 152 J=1,8
      IF(JT(I,J).EQ.0) GO TO 151
      K=JT(I,J)
      DX=X(K)-X(I)
      DY=Y(K)-Y(I)
      DL=SQRT(DX*DX+DY*DY)
      DZ=Z(K)-Z(I)
      DL1=SQRT(DL*DL+DZ*DZ)
      IF(DL.LT..001.OR.DL1.LT..001) GO TO 152
      IBAR=0
      DO 162 L=1,NB
```

```
      IF(NP(L).EQ.I.AND.MI(L).EQ.K) IBAR=L
      IF(NP(L).EQ.K.AND.MI(L).EQ.I) IBAR=L
      IF(IBAR.NE.0) GO TO 163
  162 CONTINUE
      WRITE(6,64)
      STOP
  163 CONTINUE
      FORCE=FORH(IBAR)*DL1/DL
      FOR(IBAR)=FORCE
      SUMX=SUMX+FORCE*DX/DL1
      SUMY=SUMY+FORCE*DY/DL1
      SUMZ=SUMZ+FORCE*DZ/DL1
  152 CONTINUE
  151 WRITE(8,22)I,X(I),Y(I),Z(I),SUMX,SUMY,SUMZ
   22 FORMAT(I5,6E12.5)
      WRITE(150,1) NB,NN
      WRITE(150,73)(NP(I),MI(I),FOR(I),I=1,NB)
   73 FORMAT(2I5,E20.8)
      WRITE(8,23)(I,NP(I),MI(I),ITYPE(I),FOR(I),I=1,NB)

WRITE(150,333)(X(I),Y(I),Z(I),IFIX(I),ISYM(I),(JT(I,J),J=1,8),I=1,
     1 NN)
   23 FORMAT(////7X,'MEMBER OUTPUT'/5X,' +END -END TYPE',15X,'FORCE'/
     1 (4I5,E20.8))
      call splot(np,mi,nn,nb,r,for,2)
      CALL SPLOT(NP,MI,NN,NB,R,FOR,1)
      CALL SPLOT(NP,MI,NN,NB,R,FOR,0)
      STOP
      END
      SUBROUTINE PLOT(NB, NN, X, Y, NP, MI,for,iwrite)
      INCLUDE 'FGRAPH.FD'
      DIMENSION  NP(1), MI(1), X(1), Y(1),for(1)
      INTEGER*2 DUMMY,xk,yk,xm,ym,lx,ly
      RECORD /XYCOORD/ XY
      character*6  text
      character*10 text1
      CHARACTER*64 FONTPATH
      CHARACTER*20 LIST
      FONTPATH='\bill\ilc\modern.fon'
      LIST="t'modern'"//'h6w6b'
      DUMMY = SETVIDEOMODE( $VRES16COLOR)
      DUMMY=REGISTERFONTS(FONTPATH)
      DUMMY=SETFONT(LIST)
      AMAXX=639-20
      AMAYY=479-20
c     find extent of picture window
      XMIN=X(1)
      XMAX=X(1)
      YMIN=Y(1)
      YMAX=Y(1)
      DO 2 I=1,NN
      XI=X(I)
      YI=Y(I)
      IF(XMIN.GT.XI) XMIN=XI
      IF(XMAX.LT.XI) XMAX=XI
      IF(YMIN.GT.YI) YMIN=YI
    2 IF(YMAX.LT.YI) YMAX=YI
```

```
c     scale to center of window
      SCALE = AMAX1((XMAX-XMIN)/AMAXX,(YMAX-YMIN)/AMAYY)
      XSHIFT = (XMAX+XMIN)/2.0 - 639/2*SCALE
      YSHIFT = (YMAX+YMIN)/2.0 - 479/2*SCALE
c     move and draw for each line
      DO 3 I=1,NB
      K=NP(I)
      M=MI(I)
      XK=(X(K)-XSHIFT)/SCALE
      YK=(Y(K)-YSHIFT)/SCALE
      XM=(X(M)-XSHIFT)/SCALE
      YM=(Y(M)-YSHIFT)/SCALE
c     invert picture
      YK = 479-YK
      YM = 479-YM
      LX=((XK+XM)/2)
      LY=((YK+YM)/2)
      CALL MOVETO  ( XK, YK,  XY)
      DUMMY = LINETO  ( XM, YM)
      if(iwrite.ne.2) go to 998
      call  moveto(lx,ly,xy)
      write(text, '(i3)') i
      call outgtext (text)
  998 if(iwrite.eq.0.or.iwrite.eq.2) go to 3
      call moveto(lx,ly,xy)
      write(text1,'(f7.0)') for(i)
      call outgtext (text1)
    3 CONTINUE
      if(iwrite.ne.2) go to 996
      do 997 i=1,nn
      lx=(x(i)-xshift)/scale
      yk=(y(i)-yshift)/scale
      ly=(479-yk)
      call  moveto(lx,ly,xy)
      write(text, '(i3)') i
      call outgtext (text)
  997 continue
  996 continue
      RETURN
      END

      SUBROUTINE SPLOT ( NP,NM,NN,NB,R,for,iwrite)
      INCLUDE 'FGRAPH.FD'
c     iwrite = 0   no text
c              1   writes member forces
c              2   writes node map
      DIMENSION  NP(1),NM(1),RXY(1000),ROT(3,3),for(1)
      DIMENSION ANGL(3),NT(3),A(3,3),R1(3,3,3)
      INTEGER*2 DUMMY
      DIMENSION R(1),X(500),Y(500),RZ(1000)
      WRITE(*,1)
    1 FORMAT(' YOU ARE ABOUT TO ENTER A GRAPHICS '
     1 'DISPLAY MODE'/' THE KEYBOARD COMMANDS ARE'//
     1 '      +1...POSITIVE ROTATION ABOUT X AXIS'/
     1 '      -1...NEGATIVE ROTATION ABOUT X AXIS'/
     1 '      +2...POSITIVE ROTATION ABOUT Y AXIS'/
     1 '      -2...NEGATIVE ROTATION ABOUT Y AXIS'/
```

```
     1 '      +3...POSITIVE ROTATION ABOUT Z AXIS'/
     1 '      -3...NEGATIVE ROTATION ABOUT Z AXIS'/
     1 '       0...EXIT')
c     delay for reading
      READ(*,*)
      DO 616 I=1,3
      DO 617 J=1,3
      DO 617 K=1,3
  617 R1(I,J,K)=0.
  616 R1(I,I,I)=1.
      THX=0.
      THY=00.
      THZ=00.
c     rotate using 10 deg increments
      DTH=10.
  70  PI=3.14159
      DO 604 I=1,3
      DO 603 J=1,3
  603 ROT(J,I)=0.
  604 ROT(I,I)=1.
      ANGL(1)=THX
      ANGL(2)=THY
      ANGL(3)=THZ
      NT(1)=1
        NT(2)=2
      NT(3)=3
      I=0
  302 I=I+1
      IF(ANGL(I))606,605,606
  606 L=NT(I)
      GO TO 612
  618 DO 607 J=1,3
      DO 607 JA=1,3
      A(J,JA)=0.
      DO 607 JB=1,3
  607 A(J,JA)=A(J,JA)+R1(L,J,JB)*ROT(JB,JA)
      DO 608 K=1,3
      DO 608 J=1,3

  608 ROT(K,J)=A(K,J)
  605 IF(I-3) 302,303,303
  303 DO 805 I=1,NN
      RZ(I)=0.
      DO 806 K=1,3
  806 RZ(I)=RZ(I)+ROT(3,K)*R(3*I-3+K)
      DO 805 J=1,2
      RXY(2*I-2+J)=0.
      DO 805 K=1,3
  805 RXY(2*I-2+J)=RXY(2*I-2+J)+ROT(J,K)*R(3*I-3+K)
      GO TO 59
  612 ANG=ANGL(I)*PI/180.
      IF(L-2)613,614,615
  613 R1(1,2,2)=COS(ANG)
      R1(1,2,3)=SIN(ANG)
      R1(1,3,3)=R1(1,2,2)
      R1(1,3,2)=-R1(1,2,3)
      GO TO 618
```

```
 614 R1(2,1,1)=COS(ANG)
     R1(2,1,3)=-SIN(ANG)
     R1(2,3,1)=-R1(2,1,3)
     R1(2,3,3)=R1(2,1,1)
     GO TO 618
 615 R1(3,1,1)=COS(ANG)
     R1(3,1,2)=SIN(ANG)
     R1(3,2,1)=-R1(3,1,2)
     R1(3,2,2)=R1(3,1,1)
     GO TO 618
  59 DO 24 I=1,NN
     X(I)=RXY(2*I-1)
  24 Y(I)=RXY(2*I)
     CALL PLOT(NB,NN,X,Y,NP,NM,for,iwrite)
     READ(*,*) IVAL
     IF(IVAL.EQ.+1) GO TO 2000
     IF(IVAL.EQ.-1) GO TO 3000
     IF(IVAL.EQ. 2) GO TO 4000
     IF(IVAL.EQ.-2) GO TO 5000
     IF(IVAL.EQ. 3) GO TO 6000
     IF(IVAL.EQ.-3) GO TO 7000
     IF(IVAL.EQ. 0) GO TO 8000
2000 THX=THX+DTH
     GO TO 70
3000 THX=THX-DTH
     GO TO 70
4000 THY=THY+DTH
     GO TO 70
5000 THY=THY-DTH
     GO TO 70
6000 THZ=THZ+DTH
     GO TO 70
7000 THZ=THZ-DTH
     GO TO 70
8000 CALL UNREGISTERFONTS()
     DUMMY = SETVIDEOMODE( $DEFAULTMODE )
     RETURN
     END
```

APPENDIX B

JAVA CODE OF PROGRAM edgecableApp.java

```
// Code: edgecableApp.java
// Program for computing of intersection points
// between edge cable and basic grid
// edgecableApp.java

import java.io.*;
import java.applet.*;
import java.awt.*;

public class edgecableApp {

// definition of the basic grid variables

static double dx = 1;  // spacing between vertical members in feet
static double dy = 1;  // spacing between horizontal members in feet
static int nx = 6;     // amount of vertical members
static int ny = 6;     // amount of horizontal members

// definition of the edge cable points

// fixed point P1
static double x1 = 4;        // must be different than zero
static double y1 = 1;        // must be different than zero

// fixed point P2
static double x2 = 1;        // must be different than zero
static double y2 = 4;        // must be different than zero

// middle point P3
static double x3 = 2.8;          // must be different than zero
static double y3 = 2.8;          // must be different than zero


// array to define the basic grid
static double xv[] = new double[nx+1];     // x-coordinate of vertical members
static double yv[] = new double[nx+1];     // computed y-coordinate of vertical member
static double yh[] = new double[ny+1];     // y-coordinate of horizontal members
static double xh[] = new double[ny+1];   // computed x-coordinate of horizontal member

// array to find function of edge cable geometry
static double A[][] = new double[4][4];
static double v[] = new double[4];
```

```
// variables for Gaussian Elimination
static double z[] = new double[4];
static double t1[] = new double[4];
static double t2[] = new double[4];


public static void main(String[] args) {

        // definition of the basic grid
        // computation of the basic grid coordinates
                for( int i = 1; i <= nx; i++ ) {
                        xv[i] = (i*dx);
                }
                for( int i = 0; i <= ny; i++ ) {
                        yh[i] = (i*dy);
                }

        // find xmax and xmin of edge cable
                double xmax;
                double xmin;
                if( x1 <= x2 ) {
                        xmax = x2;
                        xmin = x1;
                }
                else {
                        xmax = x1;
                        xmin = x2;
                }
                if( xmax <= x3 ) {
                        xmax = x3;
                }
                if( xmin >= x3 ) {
                        xmin = x3;
                }
        // find ymax and ymin of edge cable
                double ymax;
                double ymin;
                if( y1 <= y2 ) {
                        ymax = y2;
                        ymin = y1;
                }
                else {
                        ymax = y1;
                        ymin = y2;
                }
```

```
            if( ymax <= y3 ) {
                    ymax = y3;
            }
            if( ymin >= y3 ) {
                    ymin = y3;
            }

            System.out.println("=====================================" );
            System.out.println ( "Fixed points (supports) of the edge cable:" );
            System.out.println ( "Point 1:  x1 = " + x1 + "  y1 = " + y1 );
            System.out.println ( "Point 2:  x2 = " + x2 + "  y2 = " + y2 );
            System.out.println ( "Middle point of the cable (not fixed!!!):" );
            System.out.println ( "Middle Point:  xm = " + x3 + "  ym = " + y3 );

// compute the function y(x) for the edge cable geometry
        // define the matrix for gaussian elimination
            A[1][1] = x1*x1;
            A[2][1] = x2*x2;
            A[3][1] = x3*x3;
            A[1][2] = x1;
            A[2][2] = x2;
            A[3][2] = x3;
            A[1][3] = 1;
            A[2][3] = 1;
            A[3][3] = 1;
            v[1] = y1;
            v[2] = y2;
            v[3] = y3;
        // solve matrix (Gaussian Elimination)
            for ( int k = 2; k <= 3; k++ ) {
                    for ( int i = k; i <= 3; i++ ) {
                            double fa = (A[i][k-1]/A[k-1][k-1]);
                            for ( int j = k; j <= 3; j++ ) {
                                    A[i][j] = (A[i][j]-(fa*A[k-1][j]));
                            }
                            A[i][k-1] = 0;
                            v[i] = (v[i]-(fa*v[k-1]));
                    }
            }
            z[3] = (v[3]/A[3][3]);
            for ( int k = (3-1); k >= 1; k-- ) {
                    for ( int i = (3-1); i >= 1; i-- ) {
                            t1[i] = (t1[i]+(z[k+1]*A[i][k+1]));
                    }
                    z[k] = ((v[k]-t1[k])/A[k][k]);
            }
```

```
// edge cable geometrie function y(x) (variables)
        double ay = z[1];
        double by = z[2];
        double cy = z[3];
// output of the function
        System.out.println( "=====================================" );
        System.out.println( "Geometry function y(x) for the edge cable" );
        System.out.println( "y(x) = a*x*x + b*x + c" );
        System.out.println( "Function variables:");
        System.out.println( "a= " + ay + "  b= " + by + "  c = " + cy );
        System.out.println( "ymax = " + ymax + " ymin = " + ymin );

// compute the function x(y) for the edge cable geometry
    // define the matrix for gaussian elimination
        A[1][1] = y1*y1;
        A[2][1] = y2*y2;
        A[3][1] = y3*y3;
        A[1][2] = y1;
        A[2][2] = y2;
        A[3][2] = y3;
        A[1][3] = 1;
        A[2][3] = 1;
        A[3][3] = 1;
        v[1] = x1;
        v[2] = x2;
        v[3] = x3;
    // solve matrix (Gaussian Elimination)
        for ( int k = 2; k <= 3; k++ ) {
            for ( int i = k; i <= 3; i++ ) {
                double fa = (A[i][k-1]/A[k-1][k-1]);
                for ( int j = k; j <= 3; j++ ) {
                    A[i][j] = (A[i][j]-(fa*A[k-1][j]));
                }
                A[i][k-1] = 0;
                v[i] = (v[i]-(fa*v[k-1]));
            }
        }
        z[3] = (v[3]/A[3][3]);
        for ( int k = (3-1); k >= 1; k-- ) {
            for ( int i = (3-1); i >= 1; i-- ) {
                t2[i] = (t2[i]+(z[k+1]*A[i][k+1]));
            }
            z[k] = ((v[k]-t2[k])/A[k][k]);
        }
```

```
// edge cable geometrie function x(y) (variables)
        double ax = z[1];
        double bx = z[2];
        double cx = z[3];
// output of the function
        System.out.println( "=====================================" );
        System.out.println( "Geometry function x(y) for the edge cable" );
        System.out.println( "x(y) = a*y*y + b*y + c" );
        System.out.println( "Function variables:");
        System.out.println( "a= " + ax + "  b= " + bx + "  c = " + cx );
        System.out.println( "xmax = " + xmax + " xmin = " + xmin );

// compute intersection points between the horizontal basic grid members and the edge
cable
        // solve quadratic equation (find intersection cable/grid)
        // horizontal grid members
        System.out.println( "=====================================" );
        System.out.println( "Intersections of edge cable with horizontal basic grid
members" );
        for( int i = 1; i <= ny; i++ ) {
                if( yh[i] < (ymin-0.01) ) {
                System.out.println("No edge cable at y = " + yh[i] );
                }
                else {
                        if( yh[i] <= (ymax+0.01) ) {
                                double s1;
                                double s2;
                                int check = 0;
                                double rt = Math.sqrt( by*by - 4*ay*( cy-yh[i] ));
                                s1 = (( -by - rt ) / (2*ay) );
                                s2 = (( -by + rt ) / (2*ay) );
                                if( (xmin-0.01) <= s1 & s1 <= (xmax+0.01) ) {
                                        xh[i] = s1;
                                        check = ( check + 1 );
                                }
                                if( (xmin-0.01) <= s2 & s2 <= (xmax+0.01) ) {
                                        xh[i] = s2;
                                        check = ( check + 1 );
                                }
                                System.out.println( "Horizontal Edge Point # " + i );
                                if( check == 1 ) {
                System.out.println( "xh = " + xh[i] + "  yh = " + yh[i] );
                                }
                                if( check == 2 ) {
                System.out.println( "ambiguous result:" );
                System.out.println( "result 1:  xh = " + s1 + "  yh = " + yh[i] );
```

```
                        System.out.println( "result 2:  xh = " + s2 + "  yh = " + yh[i] );
                                }
                        }
                }
        }

// compute intersection points between the vertical basic grid members and the edge cable
    // solve quadratic equation (find intersection cable/grid)
    // vertical grid members
        System.out.println( "=====================================" );
        System.out.println( "Intersections of edge cable with vertical basic grid
members" );
        for( int i = 1; i <= nx; i++ ) {
                if( xv[i] < (xmin-0.01) ) {
                System.out.println("No edge cable at x = " + xv[i] );
                }
                else {
                        if( xv[i] <= (xmax+0.01) ) {
                                double s1;
                                double s2;
                                int check = 0;
                                double rt = Math.sqrt( bx*bx - 4*ax*( cx-xv[i] ));
                                s1 = (( -bx - rt ) / (2*ax) );
                                s2 = (( -bx + rt ) / (2*ax) );
                                if( (ymin-0.01) <= s1 & s1 <= (ymax+0.01) ) {
                                        yv[i] = s1;
                                        check = ( check + 1 );
                                }
                                if( (xmin-0.01) <= s2 & s2 <= (xmax+0.01) ) {
                                        yv[i] = s2;
                                        check = ( check + 1 );
                                }
                                System.out.println( "Vertical Edge Point # " + i );
                                if( check == 1 ) {
                System.out.println( "xv = " + xv[i] + "  yv = " + yv[i] );
                                }
                                if( check == 2 ) {
                System.out.println( "ambiguous result:" );
                System.out.println( "result 1:  xv = " + xv[i] + "  yv = " + s1 );
                System.out.println( "result 2:  xv = " + xv[i] + "  yv = " + s2 );
                                }
                        }
                }
        }
    }
}
```

REFERENCES

1. H.M. Deitel and P.J. Deitel, Java – How to Program, Prentice Hall, Upper Saddle River, NJ, 1997

2. G. Cornell and C.S. Horstmann, Core Java – Second Edition, Prentice Hall, Mountain View, California, 1997

3. S.J. Chapman, Fortran 90/95 for Scientist and Engineers, McGraw-Hill, Boston, Massachusetts, 1998

4. J.W. Leonhard, Tension Structures – Behavior and Analysis, McGraw-Hill, New York, NY, 1988

5. F. Otto, Tensile Structures – Volume one – Pneumatic Structures, The M.I.T. Press, Cambridge, Massachusetts, 1967

6. F. Otto, Tensile Structures – Volume two – Cable Structures, The M.I.T. Press, Cambridge, Massachusetts, 1969

7. T. Herzog, Pneumatic Structures – A Handbook of Inflatable Architecture, Oxford University Press, New York, NY, 1976

8. R. Levy and W. R. Spillers, Geometrically Nonlinear Structures, Chapman & Hall, New York, NY, 1995

9. H. Berger, Light Structures – Structures of Light, Birkhaeuser Verlag, Boston, Massachusetts, 1996

10. M.A. Crisfield, Non-linear Finite Element Analysis of Solids and Structures (Volume 1), John Wiley & Sons, New York, NY, 1994

11. P. Ladeveze, Non-linear Computational Structure Mechanics, Springer, New York, NY, 1999

12. J.S. Przemieniecki, Theory of Matrix Structural Analysis, Dover Publications, New York, NY, 1985

13. O.C. Zienkiewicz, The Finite Element Method (3rd Edition), Mc-Graw Hill, New York, NY, 1977

14. W.R. Spillers, Automated Structural Analysis: An Introduction, Pergamon Press, New York, NY, 1972

15. J.F. Epperson, An Introduction to Numerical Methods and Analysis, John Wiley & Sons, New York, NY, 2002

16. L. Papula, Mathematische Formalsammlung fuer Ingeniere, Vieweg, Wiesbaden, Germany, 1998

17. B. Bukiet, Notes for Math 611 Numerical Methods for Computation – Fall 2002, http://math.njit.edu/~bukiet/M611/M611.html, 12/08/2002

18. W.R. Spillers, Notes on the Calculation of Cable Nets (including Fortran Codes), http://www-ec.njit.edu/civil/faculty/spillers.html, 03/17/2003

19. W.R. Spillers, Paperless Design of Fabric Structures at NJIT – Fabric Structures, http://www-ec.njit.edu/civil/gateway.html, 04/10/2003

20. University Stuttgart, Lehrunterlagen zur Vorlesung: Leichtbau - Entwerfen in Leichtbau, http://www.uni-stuttgart.de/ilek/Lehre/Lehre.html, 01/19/2003