BSc thesis in Applied Physics and Applied Mathematics

Fatigue modelling with a two-dimensional spring network

Ruben ter Meulen
2019

FATIGUE MODELLING WITH A TWO-DIMENSIONAL SPRING NETWORK

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of

Bachelor of Science in Applied Physics and Applied Mathematics

by

Ruben ter Meulen

June 2019

Ruben ter Meulen: *Fatigue modelling with a two-dimensional spring network* (2019)

The image on the front cover and back cover is generated using the Delaunay triangle patter maker on https://msurguy.github.io/triangles/

The work in this thesis was made in the:

Engineering Thermodynamics Group
Process and Energy Department
Faculty of Mechanical, Maritime and Materials Engineering
Delft University of Technology

Supervisors:	Dr. B.P. Tighe
	Prof. dr. ir. A. W. Heemink
Co-readers:	Dr. T. Idema
	Dr. P. Boukany
	Dr. ir. F.J. Vermolen

ABSTRACT

A random periodic two-dimensional spring network us built using a Poisson disk distribution and computing the Delaunay triangulation of the centers of the disks. The edges of the network represent springs that obey Hooke's law with an adjustment on the spring constant. The adjustment allows spring to soften and to break. The two-dimensional spring network can, in combination with the adjustment, be used to model fatigue.

The network softens during a cyclic loading. During each cycle the normal strain is increased until the normal stress equals a specific cyclic stress σ_{cycle}. The loading is continued until the network is broken into two pieces, with a number of N completed cycles. For a range of cyclic stresses the number of cycles before failure N is calculated and fitted with an exponential relation. The fitting parameters for networks with $\#n = 32$ are similar to those of a networks with $\#n = 64$.

Throughout the cyclic loading springs will break and as a result the coördination of the network decreases. The coördination of the network can drop below 4. These networks are hypostatic, while the initial network was hyperstatic. This claim is asserted by the stress-strain curve.

The transition from hyperstatic to hypostatic leads to a conceptual question: "Is the spring network broken if it has teared into two pieces or when it is hypostatic?".

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

This report is the final project for the Double Bachelor Applied Physics and Applied Mathematics given by the Technical University in Delft. The bachelor contains both analytical as numerical mathematics as well as a variety of physics ranging from thermodynamics to quantum mechanics. An algorithm for the field of continuum mechanics will be built and analysed. The research is done in the Engineering Thermodynamics Group of the Faculty of Mechanical, Maritime and Materials Engineering.

Context

Materials are the basis of the mechanical world. The buildings people live in use a variety of materials such as glass, brick, wood and metals. The cars, bikes and planes are made of carbon and steel. A house can be destroyed by a natural disaster. A vehicle can break in an accident. But both houses as vehicles can also break as a result of repetitive use. If the damage is a result of repetitive use, such as opening a door many times or driving a lot, then this damage is categorized as fatigue.
Fatigue is of special importance for the aviation branch. The results of a broken part can be devastating. Consequently all airplanes are checked before every flight. The engineers working on the plane do not check every single part but only check the parts that are used intensively [1] or are of utmost importance. Other parts are only checked weekly, monthly or even only once a year. Empirical research is done to find the how many times a specific pressure can be applied before a object breaks. This information is used to determine how frequently a part should be checked.
Fatigue is first studied by Wilhelm Albert in 1837 [2]. Almost two centuries of research have led to empirical grounded theories as Miner's rule [3] and Paris' law [4]. Frequently appearing curves are S-N or Wöhler curves which are used to determine the life expectancy of a material [5].
The empirical laws can be used to built a numeric model that simulates fatigue in a way that resembles nature. The numeric model in this report uses a two-dimensional spring network.
Stress analysis can already be done using spring networks [6] but this model can be extended such that simulates fatigue.
The spring network can resemble foam [7]. Such foams are used in bicycle helmets. The temperature of the foam increases and decreases throughout the day as a result of its surroundings. The volume of the foam changes with the temperature. The deformations will result in fatigue damage. A spring network can be used to predict the life expectancy of the foam.
A Voronoi tessellation can be used represent three dimensional structures [8]. The edges represent the connection between the center of cells with centers located at the nodesDT. The interaction between cells can also be modelled with a spring network that is constructed with the Delaunay triangulation, the dual of the Voronoi tesselation. The signs of fatigue simulated with the spring network describe tissue damage.
The research done is this report can be expanded such that it can be used for the modelling of fatigue for complex objects. A network for such an object is shown in figure 1.1

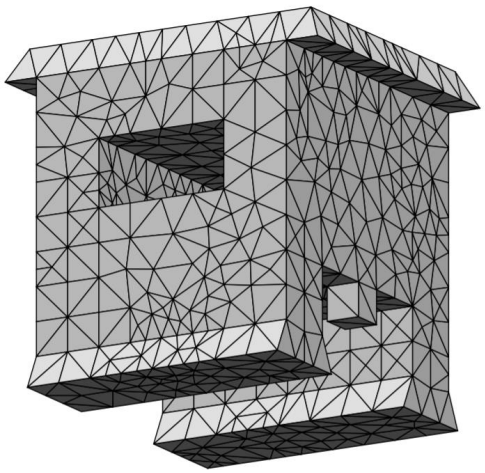

Figure 1.1: A three-dimensional mesh of Delaunay tetrahedra. The image was retrieved from [9]

The mesh can be interpreted as a network of springs with springs replacing the edges. If the springs are modified such that they can soften then the weak spots of this object can be predicted by simulating cyclic loading. The answer to the question: ”Where does the object break if it is used a stand for heavy objects” could be predicted with a expansion of the method built in this report.

Thesis objective

The methods and research presented in this proposal are a basis for further research. The objective is to build a two-dimensional spring network that simulates fatigue. The fatigue is modelled by replacing the spring constant in Hooke's law by a spring function. The characteristics of the model are examined for a specific spring function. The outcome of the measurements gives a framework for further research.

Research questions

In accordance with the thesis objective the following research question is formulated: ”What addition to Hooke's law is needed to model fatigue in a two-dimensional spring network.”. The consequences of a specific Hooke's law is examined. The main topics that are examined are fatigue life and stiffness.

Method

The method starts with a recipe for generating random spring networks. This is followed by a function that replaces the constant in Hooke's law. The implications that this function has for a single spring and a simple two-dimensional network are discussed. Finally, the method used to stretch or compress the network is introduced.
The random networks are stretched repetitively and will soften as a result of the adjusted Hooke's law. When a spring softens it will eventually break. The whole spring network is broken into two pieces after a certain number of cycles, determining the fatigue life for that specific configuration.
This report will commence with the explanation of the main concepts in chapter 2. The algorithm will be discussed in 3. In the same chapter an overview of linear

regression will be given. The chapter is concluded with a overview of the measurements reviewed in the next chapter. Four properties of the spring network are shown in chapter 4 accompanied by a discussion of the properties. In the chapter 5 will be a review of the algorithm constructed in chapter 3 along with a discussion of the results and recommendations for further improvements. After the recommendations there will be several suggestions for further research. Finally, the report is wrapped up in 6.

2 THEORY

The theory contains an brief introduction in random networks, followed by Hooke's law for one-dimensional and two-dimensional springs. The system of equations for the spring network are formulated and two boundary conditions are introduced The theory will conclude with three theoretical concepts, namely strain, stress and fatigue. Two properties that are used in the results are defined in section 2.5.

2.1 RANDOM NETWORKS

A random network is a random graph [10, 11] consisting of edges and vertices that hold attributes. Nodes, endpoints are synonyms for vertices. In the context of this research, an edge represents a spring. The network is random if there is a random element in generating the network.

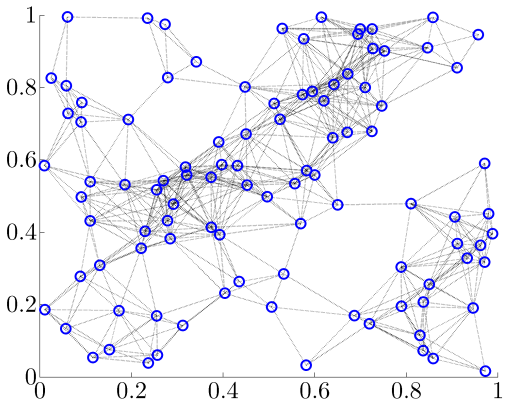

Figure 2.1: An example of a random network. The image is retrieved from [12]

The spring network build in 3 is generated with a Poisson disk distribution the coördinates of each vertex, providing the random element. The Poisson disk distribution is distribution of disks with radius r in a two-dimensional space. The disks are not allowed to overlap. The coördinates of the vertices is chosen to be initially in $[0, 1]^2$, but this space can be any rectangular region.

Attributes

The edges and vertices have attributes. The attributes of a vertex v are its Cartesian coördinates. For an edge e there are also two attributes, namely the spring constant and the rest-length. These attributes are initially set to $k_e = 1$ and $l_e = \sqrt{(x_{v1} - x_{v2})^2 + (y_{v1} - y_{v2})^2}$, where $v1$ and $v2$ are the two vertices that are connected by the edge. The quantity $\sqrt{(x_{v1} - x_{v2})^2 + (y_{v1} - y_{v2})^2}$ is the Euclidean distance between $v1$ and $v2$. The rest-length for a single spring remains constant in this report. The spring constant will be replaced by a function in section 3.3. This function allows the spring to soften and to break.

Coördination

The coördination of a network is defined as the average number of edges per vertex, which is twice the number of edges divided by the number of vertices. The factor of two origins from the two endpoints each edge has. The coördination will be important for the stiffness of the network.

2.2 HOOKE'S LAW

The edges of the random network represent springs. The force that a spring exerts on its endpoints is described by Hooke's law. Hooke's law is a fundamental concept originating from Robert Hooke in 17th century [13]. Hooke's law was published as "ut tensio, sic vis" which translates to "as the extension, so the force".
Hooke's law is an idealization of the potential. Hooke's law is obtained by truncating the potential at leading order. In this section the equation, known as Hooke's law, is represented in its one-dimension and two-dimensions form. The two-dimensional form of Hooke's law will be used to specify a system of equation for the spring network.

Hooke's law in one-dimension

A one-dimensional spring s with spring constant k_s, rest length l_s and endpoints x_i, x_j will exert the following force on endpoint x_i

$$F_{ij} = -k_{ij}(x_i - x_j - l_{ij}) \tag{2.1}$$

This law is often formulated as $F_{ij} = -k_{ij}u$. A positive spring force corresponds to the spring pulling or pushing endpoint x_i to the right. A negative spring force imposes a force on endpoint x_i that points to the left. For the rest-length in one-dimension it holds that $l_{ij} = -l_{ji}$.

Hooke's law in two dimensions

Hooke's law in two-dimensions has two endpoints (x_i, y_i) and (x_j, y_j). The spring force can be written as a vector

$$\begin{bmatrix} F_{ijx} \\ F_{ijy} \end{bmatrix} = \begin{bmatrix} -k_{ij}(x_i - x_j - l_{ij}\frac{x_i - x_j}{(x_i - x_j)^2 + (x_i - x_j)^2}) \\ -k_{ij}(y_i - y_j - l_{ij}\frac{y_i - y_j}{(x_i - x_j)^2 + (x_i - x_j)^2} \end{bmatrix} \tag{2.2}$$

Where the force is the force on endpoint (x_i, y_i). The fraction is nothing more then the projection of the spring length in the x- or y-direction respectively. This can be seen in figure 2.2

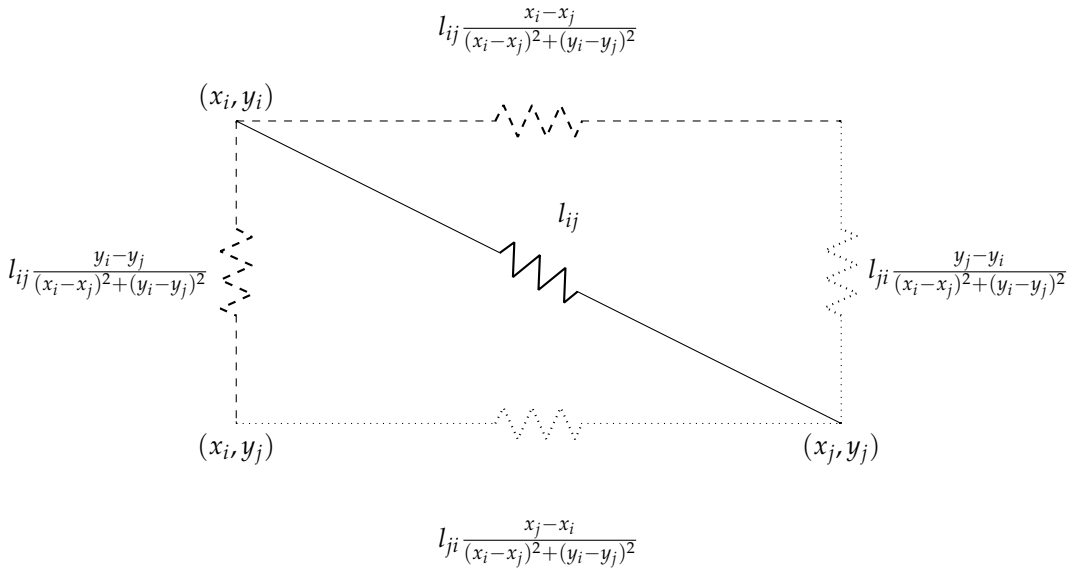

Figure 2.2: A schematic drawing of a two-dimensional spring. The spring is at rest. The dashed lines are the projections of the spring in the x- and y-direction for endpoint (x_i, y_i). The dotted lines are the projections for endpoint (x_j, y_j)

The figure shows five springs. The solid spring is the actual spring. This spring is at rest. The four other springs are projections of the rest-length in the x- and y-direction such that they are connected to endpoint (x_i, y_i) or (x_j, y_j). The projections of the rest-length can be negative.
The two-dimensional spring can be seen ass two separate springs with spring constant k_{ij} and rest-length l_{ij}. It holds that $l_{ij} = l_{ji}$, since this follows from

$$l_{ijx} = l_{ij}\frac{x_j - x_i}{(x_i - x_j)^2 + (y_i - y_j)^2} = l_{ji}\frac{x_j - x_i}{(x_i - x_j)^2 + (y_i - y_j)^2} = l_{jix}$$

Spring network

The spring network contains many springs. If the system is at rest then for each node (x_i, y_i) it must hold that the net force acting on the node is zero. The net force on each node should therefore be zero. For every node (x_i, y_i), it must hold that

$$\begin{aligned}\sum_{j\in S_i} k_{ij}(x_i - x_j - l_{ijx}) = 0\\ \sum_{j\in S_i} k_{ij}(y_i - y_j - l_{ijy}) = 0\end{aligned} \tag{2.3}$$

The set S_i contains the indices j of all endpoints (x_j, y_j) that are connected to (x_i, y_i). Every node has a linear equation for x- and y-direction depending only on the coördinates of the other nodes. A system of equations can therefore be established for the spring network.
The projections of the rest-lengths are approximated as constant and values and can be brought to the right-hand side of the equation. The system of equations can be formulated using a matrix L, position of the nodes $\underline{x}$, $\underline{y}$ and two vectors containing

the projections of the rest-length for each node $\underline{R_x}$, $\underline{R_y}$. The values of the matrix and the vectors are

$$
\begin{aligned}
L_{ij} &= -k_{ij} && \text{if } j \in S_i \\
L_{ii} &= \sum_{j \in S_i} k_{ij} && \text{if } j \in S_i \\
R_{xi} &= \sum_{j \in S_i} k_{ij} l_{ijx} \\
R_{yi} &= \sum_{j \in S_i} k_{ij} l_{ijy}
\end{aligned}
$$

The matrix L is called a Laplacian matrix [14]. The Laplacian is a singular matrix. The vector $\underline{1} = (1, 1, \ldots, 1, 1)^T$ is an eigenvector with eigenvalue 0.

$$L(\underline{x} + a\underline{1}) = L\underline{x}$$

For any $a \in \mathbb{R}$. The eigenvector can be interpreted as the tranlational freedom of the whole network.

2.3 BOUNDARY CONDITIONS

Two different boundary conditions are used. The first boundary condition is needed to remove the traslational freedom of the network. The second boundary condition will provide that the finite system simulates an infinite system [15, 16] and that the boundary of the network will not have an effect on the system.

Dirichlet boundary condition

The first boundary condition holds that a single vertex is pinned down. This is called a Dirichlet boundary condition or fixed boundary condition. When this boundary condition is applied, the translational degree of freedom is lost.
Let vertex m be the vertex that = is pinned down. The coördinates of the vertex are set to (x_m, y_m), but this could be any pair of real numbers. The mth row and mth column of the Laplacian matrix can be removed along with the mth elements of $\underline{R_x}$ and $\underline{R_y}$. Removing the mth column will have its implications on the vectors $\underline{R_x}$ and $\underline{R_y}$. If node i is attached to node m via a spring then ith element of $\underline{R_x}$ and $\underline{R_y}$ are replaced by

$$
\begin{aligned}
R_{xi} &= \sum_{j \in S_i} k_{ij} l_{xij} + k_{im} x_m \\
R_{yi} &= \sum_{j \in S_i} k_{ij} l_{yij} + k_{im} y_m
\end{aligned}
$$

With this boundary condition applied, the vector $\underline{1}$ is no longer an eigenvalue.

Periodic boundary conditions

The next boundary condition will help simulating infinite [15, 16] systems and in theory remove any effect of the boundary of the network. For a network with vertices in $[0, 1]^2$ the periodic boundary conditions will hold that each vertex is has eight copies placed as in figure 2.3

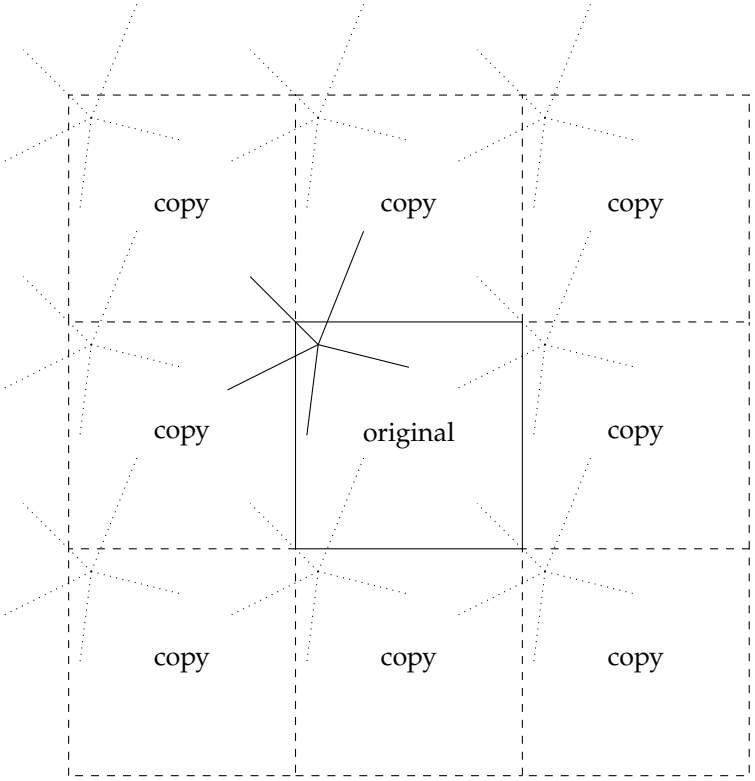

Figure 2.3: A schematic overview of the periodic boundary conditions for the random network. A vertex with five edges is shown with it's copies. Some copied edges feed into the original.

This figure also shows that the edges shows how an edge of the original network that is near the upper boundary flows into the lower boundary. Similarly an edge from the left boundary into the right boundary and one from the upper-left corner into the down-right corner. An example for a periodic graph is shown in figure 2.4

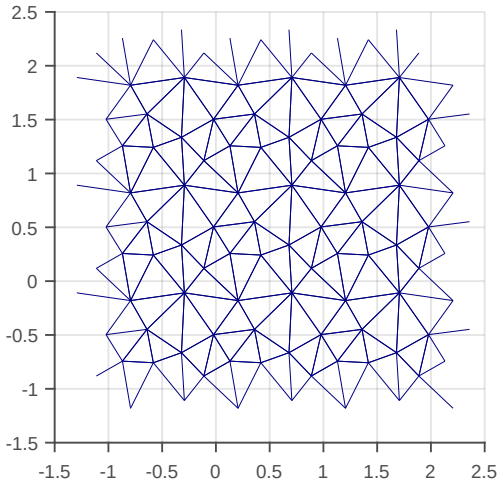

Figure 2.4: An example of a periodic graph. The figure shows an original graph and its eight copies.

2.4 STRAIN AND STRESS

Continuum mechanics is a field of mechanics where materials or substances are assumed to be continuous. With continuous is meant that the substance is described with continuous fields rather then discretized atoms. Normal strain and stress are two concepts that are used in continuum mechanics to describe deformations.

Normal strain

When an object is called deformed, it is meant that it is transformed from a reference configuration into a current configuration [17]. Normal strain ϵ_{xx} captures the deformation of uni-axial extended isotropic object. With isotropic is meant that it is uniform in all directions. Glass and metals are materials that are uniform in all directions. The strain for an isotropic object with reference length L_0 that is extended by $L' - L_0$ is defined as

$$\epsilon_{xx} = \frac{L' - L_0}{L_0} \tag{2.4}$$

The normal strain is the relative extension of an isotropic object. The first subscript x resembles that the extension is done in the direction of the x-axis. The second subscript x shows that relative extension is measured in the x-direction. ϵ_{xy} displays the relative change in width as a result of a change in length, with length and height being oriented in the x-direction and y-direction respectively.
In this report normal strain will be denoted with ϵ from now on.

Stress

Stress is the intensive physical quantity of a continuum that is a generalization of the concept force. The stress is uniquely defined by the proportionality of a force $\vec{f}$ to the surface da it acts on and the orientation of the force in respect to the normal $\hat{n}$ of the surface.
The stress is the most compact way to represent the force. The force and stress are related by $f_\alpha = \sum_\beta \sigma_{\alpha\beta} n_\beta$. With σ the Cauchy stress tensor [18]. The Cauchy stress tensor completely defines the stress of a specific configuration of a continuum. For a two-dimensional system this tensor consists of four elements, namely σ_{xx}, σ_{yy}, σ_{xy} and σ_{yx}.
The element that is of interest for the normal strain of a isotropic object is the normal stress σ_{xx}. For a spring network the normal stress is calculated using the following expression [19, 20]

$$\sigma_{xx} = \frac{1}{2V_0} \sum_i \sum_{j \in S_i} k_{ij}(x_i - x_j - l_{ij} \frac{x_i - x_j}{(x_i - x_j)^2 + (x_i - x_j)^2})(x_i - x_j) \tag{2.5}$$

The first sum iterates over all the different vertices i, while the second sum iterates over the vertices j that are connected to i. The normal stress is denoted as σ in this report.

2.5 FATIGUE

The study of fatigue is about how a continuum weakens as a result of cyclic loading. The stress-strain curve and the Wöhler curve are introduced in this section. Three properties are derived from this curve.

Stress-strain curve

The stress-strain curve is an approach to visualize cyclic loading of a continuum. One full cycle consists of a part where the stress and strain are increased until a specific stress σ_{cycle}, the cyclic stress, is reached and of a part where the stress and strain are then decreased until the strain is zero.
If a continuum softens then the maximum strain during each cycle is increased. It will take a greater strain to achieve the cyclic stress. This can be seen in 2.5

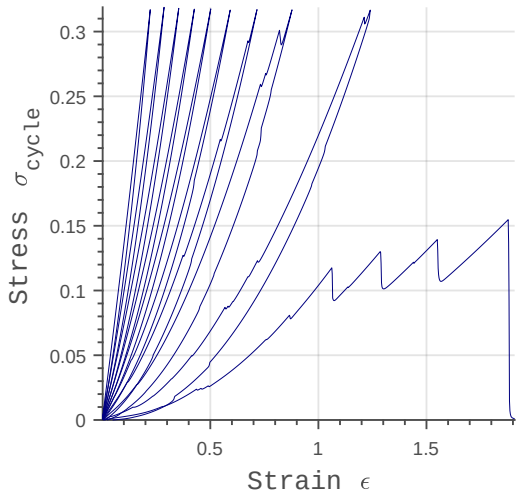

Figure 2.5: A stress-strain curve of a spring network. The network shows signs of fatigue and eventually breaks.

In this figure fatigue can be observed. The strain that is needed to achieve the cyclic stress is increased each cycle. The sudden drops in stress are caused by springs breaking in the network. With a factor dropping out of the sum in equation 2.5. The two properties that are deduced from figure 2.5 are the breaking strain ϵ_b and number of repetitions before failure N.
The breaking strain is the maximum strain of the last completed cycle. The breaking strain and contains information on the rigidity of a material. The breaking strain of a rubber band is expected to be high, showing that the rubber band is non rigid. A rope is rigid and will have a low breaking strain. The example in the figure aboce has breaking strain $\epsilon_b = 1.24$.
The number of repetitions before failure N is defined as the number of completed cycles. This number plays a key role in the life prediction of materials, components and structures [21]. The number of cycles equals the number of peaks for the stress-strain curve. The example in figure 2.5 has $N = 9$.
The breaking strain and number of repetitions before failure are dependent on the spring network and the cyclic stress. The cyclic stress can be plotted versus the number of repetitions, resulting in a Wöhler curve.

Wöhler curve

Wöhler curves, sometimes called S-N curves, graphically show the relation between cyclic stress and repetitions before failure. An example can be seen in figure 2.6 2.5

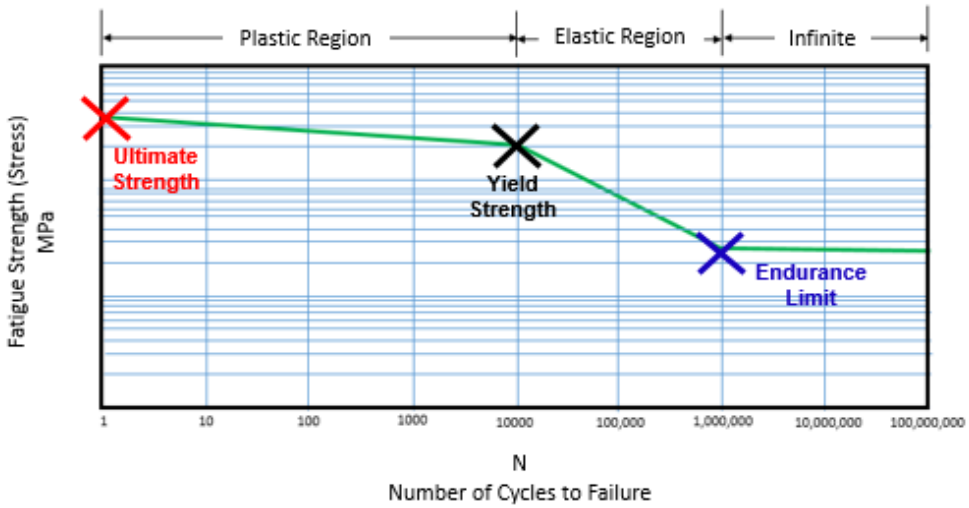

Figure 2.6: A log-log scaled figure showing the three different regions and relations between the number of cycles and the cyclic stress[22]. The three regions signify from left to right: deformation and failure, failure, endurance.

The Wöhler curve is plotted on a log-log scale and shows three different slopes. The rightmost slope is the plastic region. In the plastic region the material is deformed in an irreversible process [23]. The ultimate strength is the smallest stress for which the material breaks in the first cycle. The yield strength is the stress on the boundary lower boundary of the plastic region. The elastic region is the region where the deformation of the material is reversible [24], the material still can break after a specific number of cycles. The endurance limit is the limit where the material has elastic behavior and in addition never breaks.
The straight lines in figure 2.6 correspond to exponential relations between σ_{cycle} and N. More on this exponential relation can be found in section 3.7. To be consistent with the Basquin's exponential law, the Wöhler curves will inverted. The number of repetitions before failure N is shown as a function of the cyclic stress σ_{cycle}. This is the otherway around in the example in figure 2.6.

3 NUMERICAL METHODS

The design layout of the algorithm is discussed in this chapter. The spring network will be created using the Poisson Disc distribution and the Delaunay triangulation. The method that is used to stretching and compressing the system is explained.
The spring constant is replaced by a function, followed by an analysis of a one-dimensional spring model and a simple network. The general outline of the method and compressing is shown. Basquin's exponential law is further explained with an example. The equations for linear regression are also shown. The last property for the results, stiffness, is treated using the concepts hyperstacity, isostacity and hypostacity. The chapter is concluded with an overview of the measurements that are made. An implementation is appended in A.

3.1 POISSON DISC DISTRIBUTION

Generating the random network consists of two parts. The first part is generating the coördinates of $\#n$ vertices. The distribution that is used is called the Poisson disc distribution. The $\#n$ vertices that are generated will be spread out somewhat evenly throughout $[0,1]^2$. The algorithm that is used can be found in appendix A. A variation on the algorithm of [25] can also be used.
The algorithm takes as argument the number $\#n$, a diameter d and returns $\#n$ node points in a box of one by one, distributed by a Poisson disc distribution. Two separate nodes are at least separated by a distance d.
The distance d is set to a specific value. This value is picked such that it is small enough that it allows $\#n$ circles to be placed in $[0,1]^2$. The density of $\#n$ circles with radius r in a box of $[0,1]^2$ is given by

$$\rho = \frac{n\pi r^2}{1^2} \tag{3.1}$$

This density is the fraction of the area covered by the disks. The density of best known circle packings is close too $\rho = 0.8$ [26]. This number depends on $\#n$. The diameter that allows the Poisson disk distribution to place $\#n$ vertices should be close to

$$d = 2r = 2\sqrt{\frac{\rho}{n\pi}} \approx \frac{1}{\sqrt{\#n}} \tag{3.2}$$

The diameter that is used in practice is however $d = \sqrt{.6}/\sqrt{n}$. Diameters with $d > \sqrt{.7}/\sqrt{n}$ almost never could generate a set of $\#n$ nodes, caused by gaps between the disks as can be seen in figure 3.1

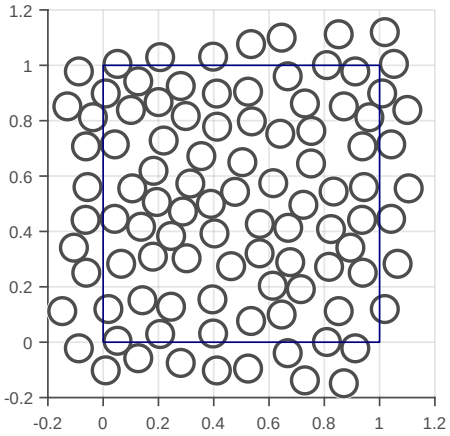

Figure 3.1: The result from the Poisson disk distribution for $\#n = 64$. The diameter of the disks seem to be appropriately chosen, since only a few disks can be added. The disks on or outside the solid line are copied and shown as a result of the periodic boundary condition 2.3.

The set of disks show only room for several other disks showing that the diameter is chosen appropriately. Some copies of disks are shown the figure above. A copy can be recognized by the center of the disk. If the center of a disk lies outside the box then it is a copy.

3.2 DELAUNAY TRIANGULATION

The second step of generating a random network is applying a Delaunay triangulation [27, 28] to the set of vertices. The Delaunay triangulation has the property that no two edges overlap.
The implementation of [29] is used on the set of nodes generated by the Poisson disk distribution. The result can be seen in 3.2

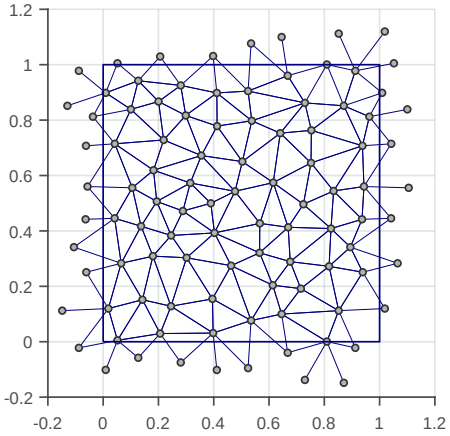

Figure 3.2: The network that is the result of applying the Delaunay triangulation the centers of the disks of figure 3.1.

Figure 3.2 shows an example of a random network that is generated by the Poisson disk distribution and the Delaunay triangulation. Before the method of stretch-

ing and compressing is introduced, the spring function law is presented and the consequences on the one-dimensional spring and a square lattice are analysed.

3.3 ADJUSTMENT TO THE SPRING CONSTANT

The springs in Hooke's Law do not soften or break. The force for a certain deviation is not dependent on how many much the spring is used. The value of the extension is not bounded. The spring can in theory be stretched without limits. To model fatigue Hooke's law needs to be adjusted such that it can soften and that it can break.
The adjustment will simulate damage. This damage is obtained when the spring is used. The spring constant can be replaced by a spring function. The following spring function can be used

$$k_{ij} := \min\left(k_{ij}, \frac{l_{ij}}{\sqrt{((x_i - x_j)^2 + (y_i - y_j)^2}}\right) \tag{3.3}$$

The spring constant is set to $k_s = 1$ initially. The spring softens as a function of the maximal length it has ever reached. When a network of springs this spring function is stretched, then the springs will soften. After the first cycle, the spring functions have a specific value. In the next cycle the cyclic stress is reached for the second time but the springs have not weakened. In order to model fatigue, the network must weaken during each cycle. The stretching or compressing itself should result in softening of the spring function.
The following spring function of the spring constant softens when it is stretched and when it is compressed

$$k_{ij} = \frac{l_{ij}}{d_{ij}} H(\frac{l_{ij}}{d_{ij}} - .1) \tag{3.4}$$

In this function l_{ij} is the rest-length of the spring and H the Heaviside step function. The quantity d_{ij} is the absolute sum over the change in deviation, initialized at $d_{ij} = l_{ij}$. This quantity is called the travelled distance of the spring. The travelled distance is a positive quantity that increases when the spring is stretched or compressed. This spring function weakens when the single spring is extended or compressed. The Heaviside step function allows the spring function to snap when $d_{ij} > 10 l_{ij}$. The spring function is ten percent of its original value when this happens. T

3.4 ONE-DIMENSIONAL MODEL

A single spring that with the spring function of equation 3.4 will show signs of fatigue. The number of repetitions of failure can be approximated for small cyclic stress $\sigma_{cycle} << 1$. Consider a single spring that follows 3.4. The spring has rest-length 1 and volume 1. Note that volume in one-dimension has units of length. For a certain deviation u the cyclic stress is given by

$$\sigma_{cycle} = \frac{1}{d(n-1) + u} u(1 + u) \tag{3.5}$$

Where $d(n-1)$ is the travelled distance at the beginning of the cycle. When in the spring is stretched in cycle n, the travelled distance equals $d(n-1) + u$. After the cyclic stress has been reached the travelled distance is increased by $|-u|$ resulting in $d(n) = d(n-1) + 2u$. Equation 3.5 can be solved for u and substituted giving

$$d(n) = d(n-1) + \sigma_{cycle} - 1 + \sqrt{(\sigma_{cycle} + 1)^2 + 4d(n-1)\sigma_{cycle}} \tag{3.6}$$

As long as the spring has not broken it can be seen that the cyclic stress can always be reached. For $\sigma_{cycle} << 1$ it holds that

$$d(n) \approx d(n-1) + 2\sigma_{cycle} + 2d(n-1)\sigma_{cycle} \tag{3.7}$$

This recurrence equation, with initial condition $d(0) = 1$ is solved to

$$d(n) = 2(2\sigma_{cycle} + 1)^n - 1 \tag{3.8}$$

The spring breaks when $k < 0.1$ or equivalently $d(n) > 10$. The spring will therefore break when

$$n = \frac{\log(5.5)}{\log(2\sigma_{cycle} + 1)} \approx \frac{0.85}{\sigma_{cycle}} \tag{3.9}$$

The factor 0.85 is only depends on the breaking condition. In general the coefficient C for which the repetitions before failure can be approximated by $n = C\sigma_{cycle}^m$, with $m = -1$, is equivalent with a the spring breaking when $d(n) > 2e^{2C} - 1$. No such an expression is found for m.

The numerical results for the single spring will be included in results 4. The repetitions before failure will be calculated by recursively calculating $d(n)$ with equation 3.6 and extracting the number first n for which $d(n) > 10$. This is done for different cyclic stresses σ_{cycle}. The breaking strain for each cyclic stress is set to $\epsilon_b = \frac{d(n-1)-d(n-2)}{2}$.

3.5 FATIGUE FOR A SIMPLE NETWORK

A square grid can be interpreted as a very simple periodic network. This simple network is shown in figure 3.3

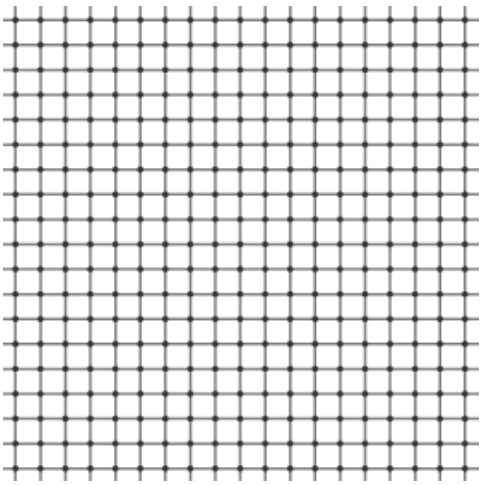

Figure 3.3: A square lattice with. The the four springs are attached in the shape of a cross. The figure is retrieved from the American Mathematical Society [30].

The square grid can be described as a periodic network where all the original nodes are in $[0,1]^2$. If there are n^2 points in this network then the stress for a strain of u is given by

$$\sigma_{cycle} = n^2 \frac{1/n}{1/n + u/n}(u/n)(1/n + u/n) = \frac{1}{1+u}u(1+u) \tag{3.10}$$

The stress of the square lattice is given by the number of horizontal springs n^2 times the stress of a single spring with rest-length $1/n$ and deviation u/n. The stress has the same expression as the single spring 3.5. The the approximation and the results for the single spring will therefore also hold for the square lattice.

3.6 RANDOM NETWORK TRANSFORMATION

The method of stretching and compressing uses the copies of the network introduced in section 2.3. Figure 3.4 shows how six copies can be displaced in order to stretch the network

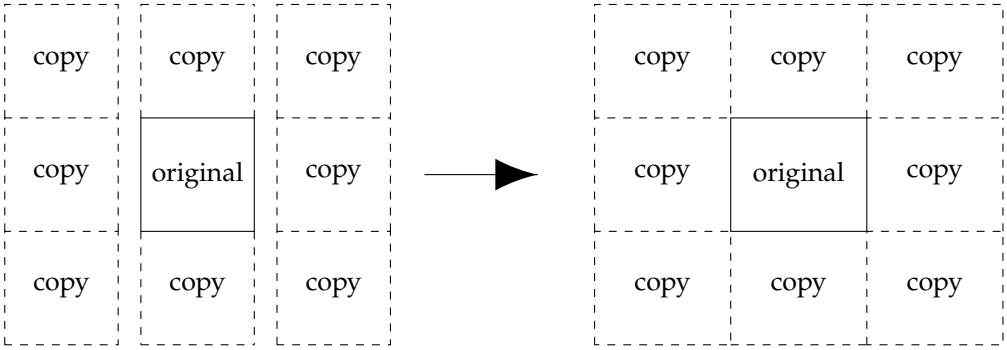

Figure 3.4: A schematic overview of the method of stretching for a random network. This method can be applied to networks similar to those of figure 3.2.

The figure shows how six copies can be displaced creating a gap and how the network relaxes to fill the gap. The gap between the on both sides of the original should be sufficiently small. If the gap is too big then the springs that cross feed into the left three copies or right three copies are stretched significantly more. The springs on the border will break before the other springs if the gap size is too big. A sufficiently small gap size is a gap size for which the spring network does not break at the border of the original network.
Compressing of the network is done by shifting the copies three copies on the left and right over the middle three, followed by a relaxation of the network.

3.7 BASQUIN'S EXPONENTIAL LAW

In 1910 Basquin proposed a exponential relationship between the repetitions before failure and cyclic stress, having used wöhler's data [31]. He proposed that

$$N = C\sigma_{cycle}^{m} \tag{3.11}$$

Where C and m are empirically or numerically determined values. In this chapter only the notation N and σ_{cycle} is denoted as N and σ respectively. This relation can be visualized in Wöhler curves on a log-log scale. The resulting fit will look like a line when it is plotted on a log-log scale. An example of empirical data fitted with a Basquin slope is shown in figure 3.5

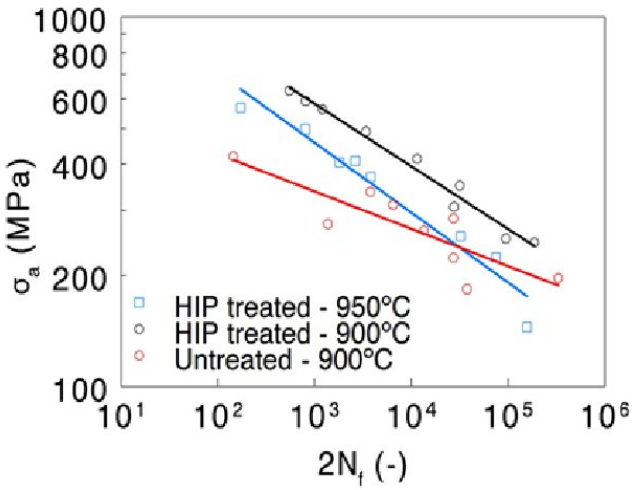

Figure 3.5: A log-log scaled Wöhler curve, showing the Basquin slopes of hot isostatic pressing (HIP) treated and untreated MAR-M247 [32]. Note that twice the number of cycles before failure is shown on the x-axis.

The figure shows that Basquin's exponential is at least a good approximation of the relation between the number of repetitions and the cyclic stress.
Basquin's exponential law can be rewritten as $y_i = ax_i + b$ with $y_i = \log(N_i)$, $x_i = \log(\sigma_i)$, $a = m$ and $b = \log(C)$. The random element in the network will result in an uncertainty in the number of repetitions and therefore also in y_i. A measurement of y_i has a mean μ_i and a standard deviation α_i.
The standard linear regression is replaced with a weighted fit with weights $w_i = \frac{\mu_i^2}{\alpha_i^2}$.
The coëffiecients m and C are given by [33]

$$m = \frac{\sum_i w_i \sum_i w_i x_i y_i - \sum_i w_i x_i \sum_i w_i y_i}{\sum_i w_i \sum_i w_i x_i^2 - (\sum_i w_i x_i)^2} \tag{3.12}$$

$$\alpha_m = \sqrt{\frac{\sum_i w_i}{\sum_i w_i \sum_i w_i x_i^2 - (\sum_i w_i x_i)^2}} \tag{3.13}$$

$$C = \exp \frac{\sum_i w_i x_i^2 \sum_i w_i y_i - \sum_i w_i x_i \sum_i w_i x_i y_i}{\sum_i w_i \sum_i w_i x_i^2 - (\sum_i w_i x_i)^2} \tag{3.14}$$

$$\alpha_C = C\sqrt{\frac{\sum_i w_i x_i^2}{\sum_i w_i \sum_i w_i x_i^2 - (\sum_i w_i x_i)^2}} \tag{3.15}$$

These sets of equation are not only used for calculating the coefficients for Basquin's exponential law. The equations are also used for The relation between the cyclic stress and breaking strain, for which a part of the data shows a possible exponential relation.

The number of repetitions for the single spring can be calculated numerically and has no uncertainty. The parameters of Basquin's exponential law for M measurements is given by [33]

$$m = \frac{M \sum_i x_i y_i - \sum_i x_i \sum_i y_i}{M \sum_i x_i^2 - (\sum_i x_i)^2} \tag{3.16}$$

$$\alpha_m = \alpha_{CU}\sqrt{\frac{M}{M \sum_i x_i^2 - (\sum_i x_i)^2}} \tag{3.17}$$

$$C = \exp \frac{\sum_i x_i^2 \sum_i y_i - \sum_i x_i \sum_i x_i y_i}{M \sum_i x_i^2 - (\sum_i x_i)^2} \tag{3.18}$$

$$\alpha_C = C\alpha_{CU} \sqrt{\frac{\sum_i x_i^2}{M \sum_i x_i^2 - (\sum_i x_i)^2}} \tag{3.19}$$

$$\alpha_{CU} = \sqrt{\frac{1}{M-2} \sum_i (y_i - mx_i - \log(C))^2} \tag{3.20}$$

Basquin's exponential law returns the number of repetitions N as a function of σ_{cycle}

3.8 STIFFNESS

The random network starts out as a structure. If for example the network in figure 3.2 is stretched then the springs are also extended. This extension increases the stress σ of the system. A deformation comes at the cost of work. The network is categorized as hyperstatic [34].
The opposite of hyperstatic is hypostatic. A hypostatic network can be deformed without an increase of stress. The network is a mechanism rather then a structure. Suppose that every node four springs attached to it, like the square lattice. There are four constraints for each node but the node also shares the springs with four other nodes. In two-dimensions the nodes have two degrees of freedom. The position of the vertices has as many constraints as degrees of freedom. For every degree of freedom there is a constraint. A constraint can only be applied to a single node, causing the number of springs needed to be double the number of springs.
Maxwell states that if the number of springs e for a network with v vertices follows

$$e > 2v - 4 \tag{3.21}$$

that the network is hyperstatic [35]. The network is a structure that can bear stress if there are two springs per node. This translates to a coördination of at least 4.
Conversely, if the coördination is below four then the network is hypostatic. The network can be extended as a mechanism. If the strain is increased sufficiently then stress of the network will increase. The network is connected and for a certain strain the tension in the springs will increase.
The final coördination Zof a network is the last property that is used in the results. The final coördination is measured when the stress is increased for the last time. If the network breaks then the Laplacian becomes singular, resulting in theory to infinite coordinates of one half of the system. The other half is pinned down using the Dirichlet boundary condition.

3.9 MEASUREMENTS

The number of repetitions before failure N and the breaking strain ϵ_b are calculated for the single spring. The cyclic stresses σ_{cycle} are chosen to be 41 logarithmically spaced points in $[10^{-2}, 10^{1}]$.
The same cyclic stresses are used to simulate fatigue in the spring network model. The measurement is repeated ten times. Two system size are used, namely $\#n = 32, 64$.
The gap size for measurements for a cyclic stress $\sigma_{cycle} < 0.1$ is 10^{-4}. For $0.1 < \sigma_{cycle} < 0.2$ is 10^{-3} and finally $0.2 < \sigma_{cycle}$ is $2 * 10^{-3}$.

4 RESULTS

This chapter will contain the results of simulations of the two-dimensional spring model with a modified Hooke's law 3.3. The measurements are described in section 3.9. Four different properties are analysed as a function of the cyclic stress, namely the repetitions before failure, breaking strain and final coördination. The definitions of these properties can be found in section 2.5.

4.1 BASQUIN'S EXPONENTIAL LAW

An explicit relation for the repetitions before failure N for the single spring is derived in section 3.4 that holds if $\sigma_{cycle} << 1$. The numerical results of this spring follow and are compared with the results of the spring network.

Single spring

The travelled distance for the single spring is calculated with a recursion relation. The recursion equation is approximated, resulting in $N = \frac{0.85}{\sigma_{cycle}}$ for $\sigma_{cycle} << 1$. The numerical results are shown in figure 4.1

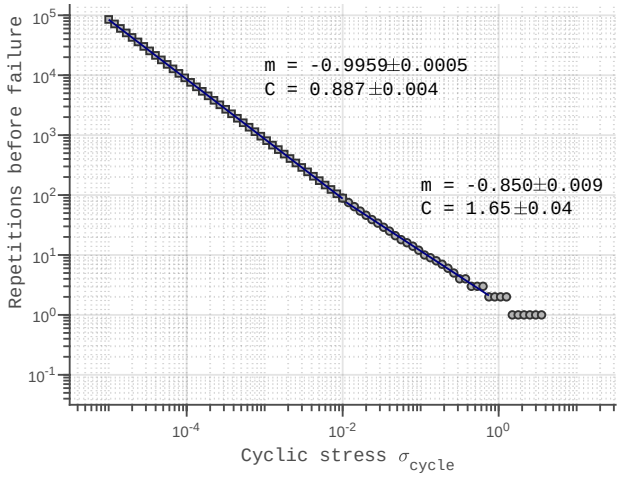

Figure 4.1: The numerical results of the repetitions before failure versus cyclic stress for the single spring model are shown on a log-log scale.The data is separated in two sets, indicated by the square and circular markers. Each data set is fitted by a Basquin slope $N = C\sigma_{cycle}^{m}$.

A data set of $\sigma_{cycle} \in [10^{-2}, 10^{1}]$ is shown with circular markers. The fitted Basquin slope on the smallest 26 cyclic stresses of this data set. This is to be consistent with the later fit of the spring network. The linear regression resulted in coefficients $C = 1.65 \pm 0.04$ and $m = -0.850 \pm 0.009$.
The square markers show the relation between the cyclic stress and repetitions before failure for 41 cyclic stresses $\sigma_{cycle} \in [10^{-5}, 10^{-2}]$. The data set if fitted with $N = C\sigma_{cycle}^{m}$. It is found that $m = -0.9959 \pm 0.0005$ and $C = 0.887 \pm 0.004$.

This is consistent with the approximation. For 41 logarithmically spaced cyclic stresses $\sigma_{cycle} \in [10^{-6}, 10^{-4}]$ a coefficient $C = 0.85 \pm 8 * 10^{-6}$ is found along with $m = -1 \pm 7 * 10^{-7}$. The results for this are shown in

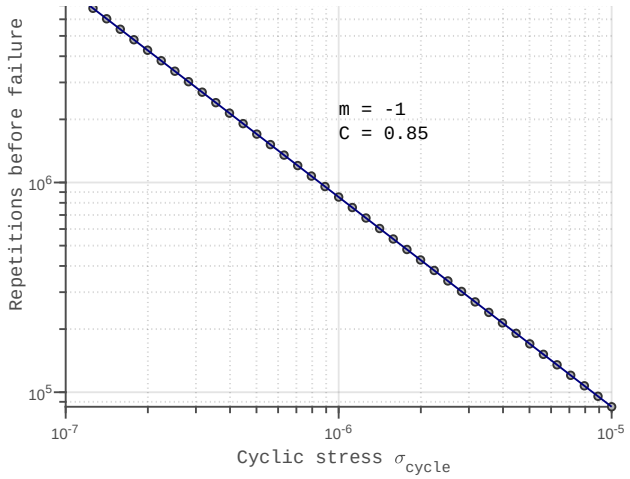

Figure 4.2: The numerical results of the repetitions before failure versus cyclic stress for the single spring model are shown on a log-log scale.The cyclic stresses are in the range $\sigma_{cycle} \in [10^{-7}, 10^{-5}]$ and if fitted with a Basquin slope $N = C\sigma_{cycle}^{m}$.

Spring network

The network consists of many spring that have varying orientations. The rest-lengths of the springs vary as well as the number of springs per node. The outcome of the simulations for the repetitions before failure versus cyclic stress is shown in figure 4.3 and figure 4.4

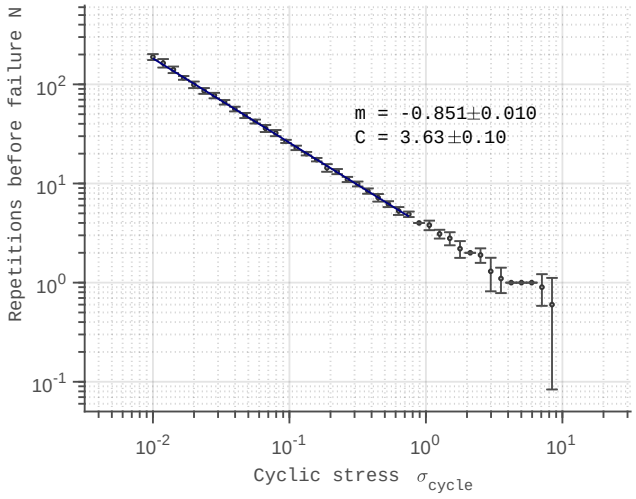

Figure 4.3: The repetitions before failure versus cyclic stress is shown for random networks with $\#n = 32$ vertices. A part of the data is fitted with a Basquin slope $N = C\sigma_{cycle}^{m}$.

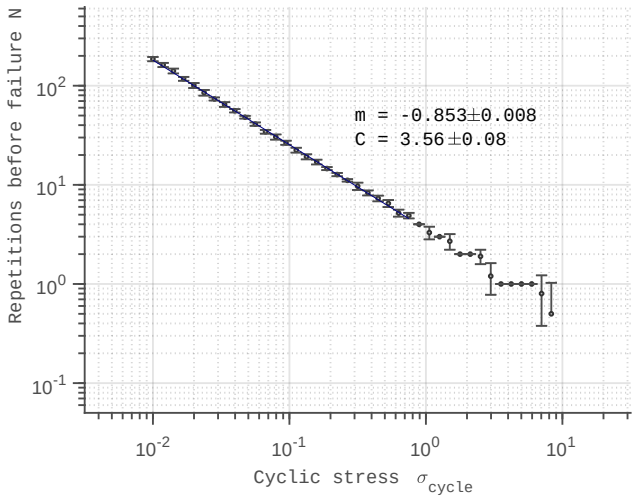

Figure 4.4: The repetitions before failure versus cyclic stress is shown for random networks with $\#n = 64$ vertices. A part of the data is fitted with a Basquin slope $N = C\sigma_{cycle}^{m}$.

The Basquin slope for a network with $\#n = 32$ nodes is similar to that of a network with $\#n = 64$ nodes. The downward half of some error bars of seems stretched, which is due to the log-log scale. For example a cyclic stress that has number of repetitions $N = 0.5 \pm 0.5$ would have the upper half of the error bar from 0.5 to 10^0 and the other stretching to minus infinity. An error bar with infinite length is not shown in figure 4.4.
For some cyclic stresses all ten instances resulted the same number of repetitions before failure. This corresponds with a zero variance and an error bar with zero length. The weights for these specific cyclic stresses are infinite. The data with infinite weights is left out of the weighted linear regression. The data with a higher cyclic stress then the data with infinite weights is also left out of the regression. This leaves the lower 26 cyclic stresses for the weighted fit for both networks.
The fit for the network with 32 nodes resulted in fitting parameters $m = -0.851 \pm 0.010$ and $C = 3.63 \pm 0.10$. The parameters for the fit for the network of 64 nodes are $m = -0.853 \pm 0.008$ and $C = 3.56 \pm 0.08$. These results are in accordance with each other. The small differences are most likely caused by the randomness in the data.
The parameters m of the single spring and the spring network are comparable. The Poisson disk distribution has resulted that the expression 3.10for the cyclicstress of the square lattice holds for the random network

$$\sigma_{cycle} = n^2 \frac{1/n}{1/n + u/n}(u/n)(1/n + u/n) = \frac{1}{1+u}u(1+u)$$

The evenly distributed nodes will have spring lengths close to $1/n$ and a deviation u/n.
The difference for the parameter C can be ascribed to the increased number of springs. With more springs the cyclic stress be achieved at a lower strain, more energy is needed to deform the system. A lower strain corresponds with less damage and as a result the Basquin slope will shift to the right.

4.2 STIFFNESS

The coördination determines the stiffness of the network. The network can be categorized as hyperstatic or hypostatic by looking at the coördination. The final coördination for the networks of 32 and 64 nodes are shown in figure 4.5 and figure 4.6

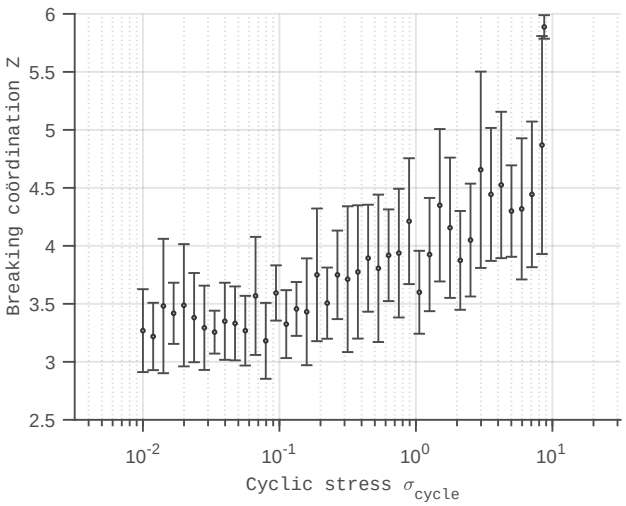

Figure 4.5: The final coördination versus the cyclic stress is shown for a network with $\#n = 32$ vertices.

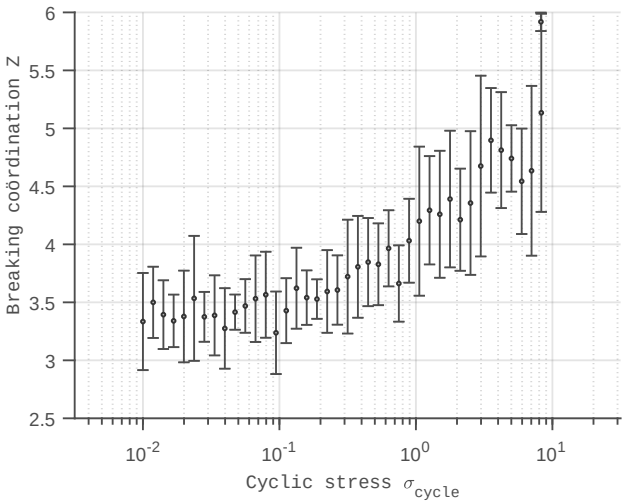

Figure 4.6: The final coördination versus the cyclic stress is shown for a network with $\#n = 64$ vertices.

The uncertainty is relatevily big due to the moment of the measurement.The moment that the final coördination is measured when the stress was increased for the last time. With the spring constant is a function of travelled distance, it is expected that the system breaks half of the time when the strain increases and the other half when the strain decreases. Half of the results of the final coördination are measured at the wrong moment. The data from the last two figures is adjusted by removing the data points of systems that have broken in the part of the cycle where the strain decreases resulting in figures 4.7, 4.8

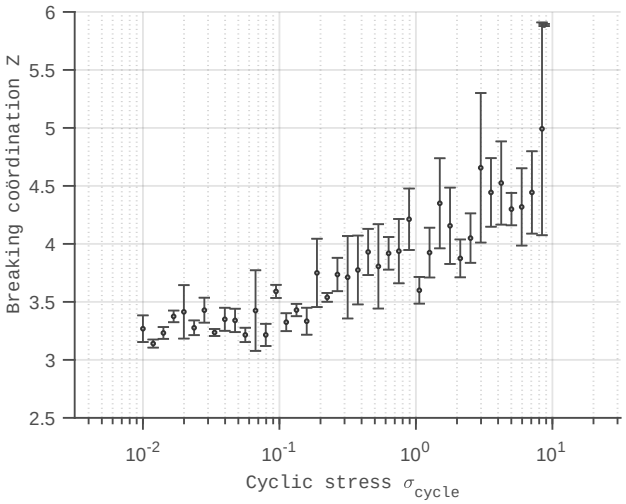

Figure 4.7: The final coördination versus the cyclic stress is shown for a network with $\#n = 32$ vertices. The data of the networks that broke during compression part of the cycle are left out.

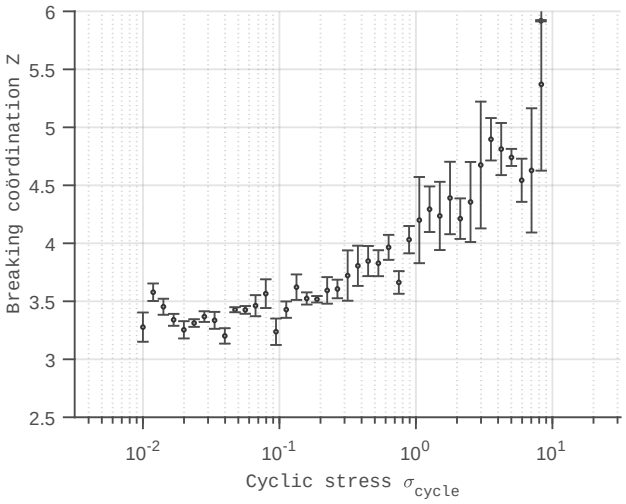

Figure 4.8: The final coördination versus the cyclic stress is shown for a network with $\#n = 64$ vertices. The data of the networks that broke during compression part of the cycle are left out.

The length of the error bars is decreased of almost all data points. When the cyclic stress is bigger most or even all of the ten measurements break on the first cycle, for these measurements the error bars are similar or equal relative to the earlier figures. 4.5, 4.6. The final coördination for a low cyclic stress is well below 4, corresponding with hypostatic networks. For a higher cyclic stress the networks are found to be hyperstatic.
The final coördination can also be visualized with showing the network just before it breaks. Three networks are shown for cyclic stresses $\sigma_{cycle} = 7.0795, 0.0398, 10^{-4}$. The network for a cyclic stress of $\sigma_{cycle} = 7.0795$ is shown in figure4.9

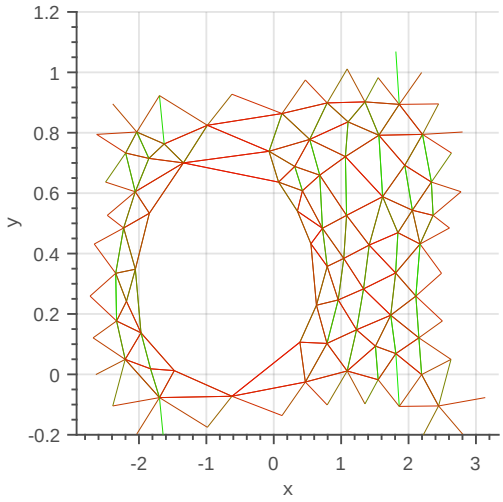

Figure 4.9: A network of $\#n = 64$ vertices is shown. T The cyclic stress is $\sigma_{cycle} = 7.0795$. The network visualizes how a high cyclic stress corresponds with a high final coördination of $Z = 5.75$. The network broke on the first cycle. The breaking strain is undefined for this case.

he color indicates the amount of softening. The color green is used for no or a little decrease in the spring function, red for a lot of softening. The color is change continuously. The color shows that the vertical springs have suffered more damage then the vertical springs.
This network has a coördination of 5.75 which is recorded as the final cördination. The vertical crack is distinctive for the the high final coördination.
There are 368 springs left when the coördination is measured. From figure 4.9 it can be seen that there are eight more springs needed to break before the system is broken. The real final coördination is therefore 5.625 giving a bias of roughly 0.1.
When the cyclic stress is close to 0.1 or lower, the damage is spread out throughout the system. Many springs are broken, this can be seen in figure4.10.

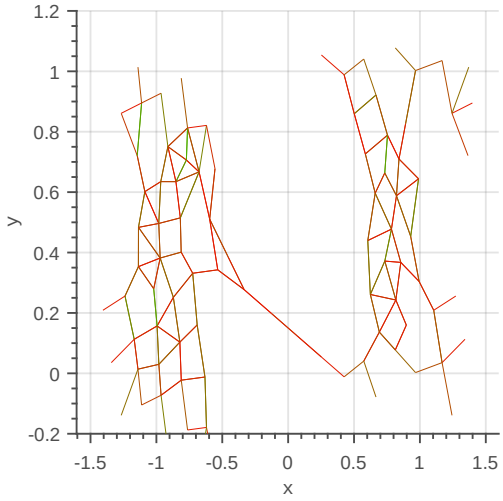

Figure 4.10: A network of $\#n = 64$ vertices is shown. The cyclic stress is $\sigma_{cycle} = 3.98 * 10^{-2}$. A lot of the springs have broken. It seems that there are several small and big cracks resulting in a low final coördination of $Z = 3.66$. For this network $\epsilon_b = 1.93$.

The final coördination measured is 3.6563. Only one more spring will break before the system is broken, resulting in a bias of roughly 0.01. This is very common when a low breaking coördination is measured.
The cyclic stress used for loading of the next network is $\sigma_{cycle} = 10^{-4}$

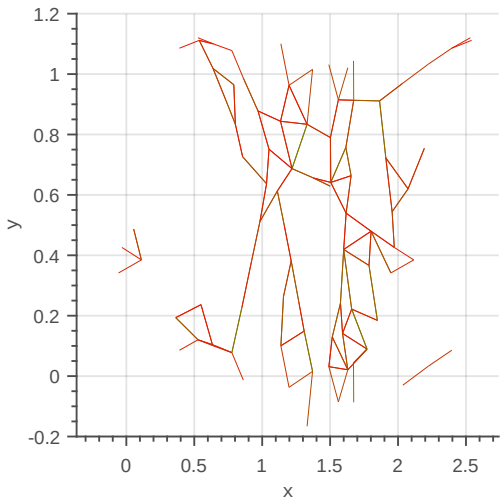

Figure 4.11: A network of $\#n = 64$ vertices is shown. The cyclic stress is $\sigma_{cycle} = 10^{-4}$. A lot of the springs have broken. It seems that there are several small and big cracks resulting in a low final coördination of $Z = 2.12$. For this network $\epsilon_b = 1.12$.

With a final coördination of $Z = 2.12$, the network is hypostatic. The stress-strain curve of the last network is shown in figure 4.12

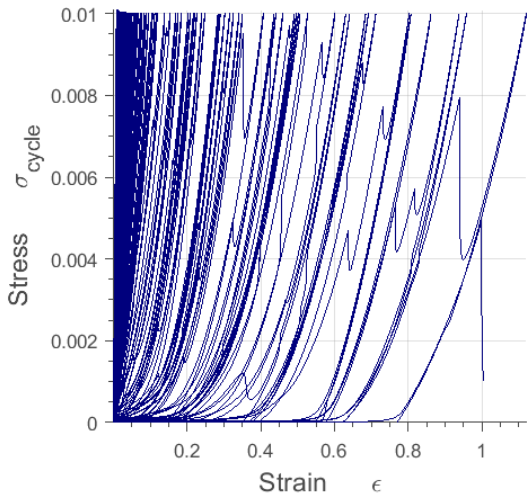

Figure 4.12: The stress-strain curve of the network in figure 4.11.

The strain increases without resulting in an increase of the stress. After a while the stress begins to increase. The first and the last completed cycle of the stress-strain curve are plotted on a log-log scale in figures 4.13, 4.14

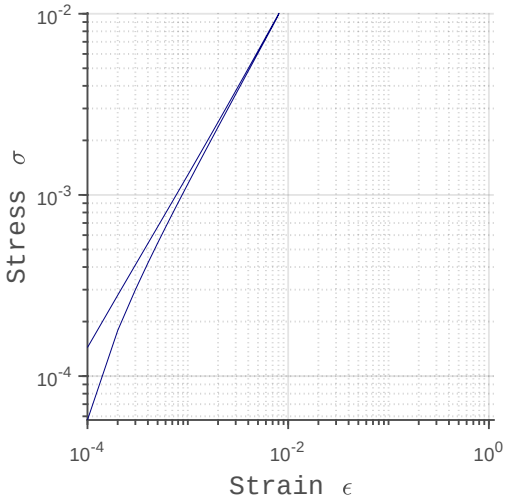

Figure 4.13: A log-log plot of the first cycle of the stress-strain curve of figure 4.12.

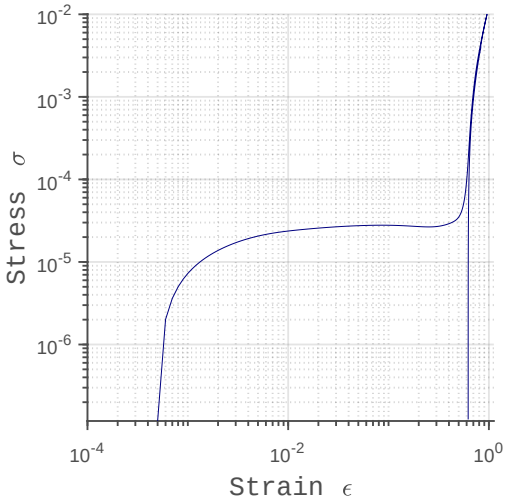

Figure 4.14: A log-log plot of the last cycle of the stress-strain curve of figure 4.12.

Figure 4.13 shows the stress-strain curve of a hyperstatic network. The softening can be seen the stress for a specific strain has two values. The lower value of stress is measured for the compressing part of the cycle.
The stress-strain curve of the last cycles shows that the strain can be increased without without resulting in a positive stress.

4.3 BREAKING STRAIN

The breaking strain is also analysed for the single spring 4.15

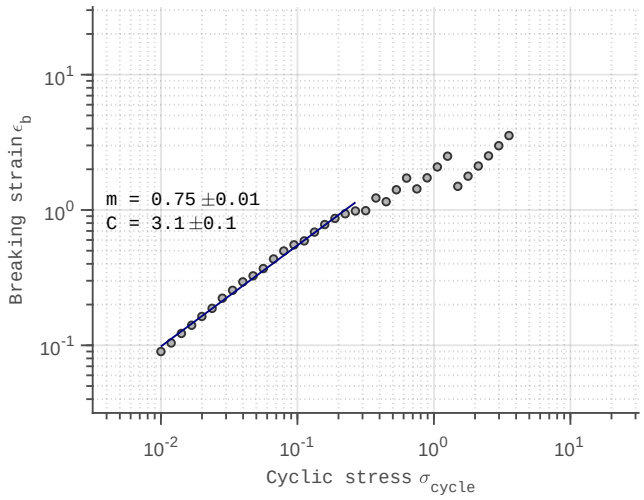

Figure 4.15: The breaking strain versus cyclic stress for the single spring. A part of the data is fitted with a power relation $\epsilon_b = C\sigma_{cycle}^m$

The breaking strain has an upper bound of 9, that corresponds to a single spring that breaks on the first cycle. The number of 9 comes from that the spring breaks if the travelled distance of the spring is increased by 9.
A lower cyclic stress does not always correspond to a lower breaking strain. The softening of the spring will as a result have that the breaking strain is increased every cycle. A lower cyclic stress corresponds with less damage per cycle with as a result can increase the fatigue life by one cycle. The breaking strain is increased in this last cycle and greater then the case where the cyclic stress was slightly bigger and a repetition less. Twenty data points are fitted with the power relation $\epsilon_b = C\sigma_{cycle}^m$. The fitting parameters resulted in $m = 0.75 \pm 0.01$ and $C = 3.1 \pm 0.1$.

Spring network

The results for the breaking strain for networks of sizes $\#n = 32$ and $\#n = 64$ nodes is shown in the following two figures

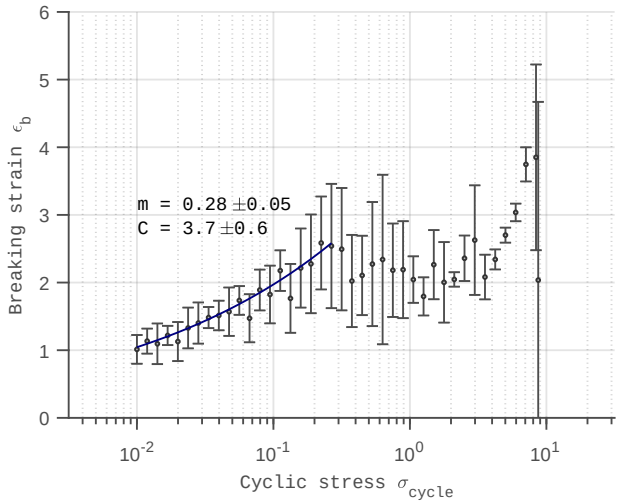

Figure 4.16: Breaking strain versus cyclic stress for a system of $\#n = 32$ nodes. A power fit , $\epsilon_b = C\sigma_{cycle}^m$,of the rightmost 20 data points is shown.

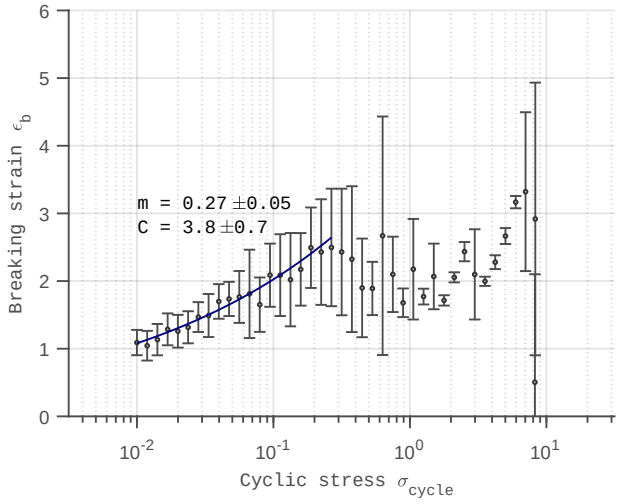

Figure 4.17: Breaking strain versus cyclic stress for a system of $\#n = 64$ nodes. A power fit , $\epsilon_b = C\sigma_{cycle}^m$,of the rightmost 20 data points is shown.

The data shown in figure 4.16 and figure 4.17. is not very precise compared to the data in the Basquin slope. The breaking strain seems to be very dependent on the initial system.
For low cyclic stress, the breaking strain of the spring network is at least ten times as big as the breaking strain for the single spring. This can be seen in figure 4.12. There are a few springs that are connected in a line. The springs have a lower rest-length and for an equal cyclic stress they will have a lower stress than that of a single spring. In section 3.5 it is shown that the expression for the stress of the square lattice with a deviation u is equal to that of a single spring with deviation u. For a square lattice with $\#n = m^2$ points a row will have m node points and therefore m springs. The stress in a row is given by

$$\sigma = n\frac{u/n}{d(n-1)+u/n}(u/n)(1/n+u/n) = \frac{1}{n}\frac{u}{d(n-1)+u}u(1+u)$$

The line of springs in figure 4.12 will, as a result of their smaller individual rest length, not bear as much stress as when the line consisted of one long spring. The network will need a bigger strain to reach the cyclic stress. The breaking strain is therefore also bigger.
When the network is hypostatic, it can be extended up to a certain strain without increasing the stress. If the stress is not increased then the springs are not stretched. This certain strain will also increase the breaking strain.
The slopes fitted to figure 4.16 and 4.17 suggest that the breaking strain will go to zero if the cyclic stress goes to zero. While the fit itself seems good more data is needed to see if the exponential relation can be extrapolated. This is discusses at the end of the next chapter.

5 DISCUSSION

This chapter will begin with several options for improving the C++ algorithm. The results of Basquin's exponential law, stiffness and breaking strain are discussed. The chapter is conluded with some suggestions for further research.

Possible improvements for the C++ algorithm

The data in the results is extracted from a data set generated by the C++ algorithm. The time needed to generate this data set was over three days.
At the end of section 3.9 the size of the gap is selected. For a strain of 1 and cyclic stress of $\sigma_c ycle = 10^{-2}$, a displacements are needed. The system of equations are solved a thousand times. This is the most time consuming step of the algorithm. Ideally an analytical or numerical expression for the gap size will depending on the number of nodes and stress, can be used to minimize the number of displacements. Perhaps Cramer's rule can be used to solve the coördinates [36]. The expression for the coördinates will only depend. spring functions, rest-lengths and the strain of the network. These expressions can be used to limit the damage on the border of the network. Numerical data can otherwise be used to find the relation between cyclic stress σ_{cycle} and maximal gap size. This can be done by finding the minimal gap size for which the network breaks at the edge.
The projections of the rest-length are approximated as constant. A linear approximation will alow the gap size to be bigger. The Gaussion elimination can maybe be replaced by a sparse solver [37].

Basquin's exponential law

The data of the spring networks seem to follow Basquin's exponential law. The exponent of the spring network is similar to that of the single spring and square lattice. The exponent of the single spring changed when the number of repetitions was obtained for a different interval of the cyclic stress, this might also be the case for the spring network.
The fitting parameters for Basquin for the networks of size $\#n = 32$ nodes are similar to those of $\#n = 64$ nodes. The expression for the stress in the square lattice suggests that the parameters are independent of $\#n$. For the random network this needs to be checked for system sizes with very few nodes and with a higher number of nodes.

Stiffness

The moment for the measurement of the final coördination is not chosen according to the definition. As a result the data did not represent the true final coördination of the network. The problem with the disconnect graph can be solved by implementing an algorithm that checks if there is a path from one node to all the nodes. This path can cross the boundary of the network.
For low cyclic stress the initial hyperstatic networks became hypostatic networks. As a result the structure of springs transitioned into a mechanism of springs. The definition of a broken network is under discussion. It can be stated that a stiff object can be classified as broken when it is sloppy. The answer will dependent on the con-

text. If the answer holds that the network is broken when it is sloppy, then the data will be significantly different and better definitions of hyperstatic and hypostatic are needed.

Breaking strain

The breaking strain for the spring network does not show an obvious similarity with that of the single spring and square lattice. If definition of a broken network is that the network is sloppy then the breaking strain for a specific cyclic stress will be much lower then it is with the used definition of broken. This can be seen in figure 4.12.
It can also be expected that the breaking strain is dependent on the system size but more measurements for different network sizes is needed. If a network consists of more nodes then the line of springs connecting the network can be expected to be replaced by a beam of springs.

The definition of the breaking strain is the greatest strain achieved during the completed cycles. It could be seen that the variance of high cyclic stresses $\sigma_{cycle} \approx 5$ is low.

6 CONCLUSION

The periodic random two-dimensional spring network has been adjusted such that it models fatigue. The spring function that replaces the spring constant was a function that decreased when the deviation of the spring is changed. The spring network weakens when it is extended and when it is compressed. When a spring has softened by a specific amount it breaks. The coördination of the network decreases during the cyclic loading. Eventually the spring network breaks into two pieces.
The number of repetitions before failure is determined for random networks with $\#n = 32$ and $\#n = 64$ nodes for varying cyclic stress. This is fitted with Basquin's exponential law and compared to the case of a square lattice. The fitting parameters for the two network sizes are similar. The exponent of the fit of the square lattice and the spring network are found to be similar. It is expected that the exponent depends on the range of the different cyclic stresses that are simulated.
The coördination of the network dropped below 4 for low cyclic stress. These networks are hypostatic and therefore sloppy. The initial network is hyperstatic which corresponds with stiff. As a result the definition of broken is under discussion. Is the network broken when it is sloppy or when it has teared into two pieces. The strain of networks in a hypostatic state can be increased without increasing the stress, after a certain strain the stress of the network will increase.
The results of Basquin's exponential law will be dependent on the definition of broken. The dependence on the definition will have a significant impact on the breaking strain. The maximum strain per cycle increases per cycle. The increase is the greatest during the last few cycles. With a different definition the cyclic process would have already been stopped resulting in a significant lower breaking strain.
The results of this research can be further developed. An option is to expand the model to three dimensions with a Delaunay tetrahedralization. This can then be used to predict where complex object are likely to break. Using a computer model the weak points of 3D printed objects can be predicted.
The method can also be further expanded such that it models tissue damage. The cells can be represented with a Voronoi diagram. The interaction between the cells can be modelled with a Delaunay triangulation. The transition from stiff network to a sloppy network as a result of cyclic loading can be of interest for this particular topic.
The modelling of fatigue can help understanding the fatigue life of materials. The relation between cyclic loading and repetitions before failure as well as the stiffness of a material. The spring network model can be analysed numerically and has various applications. With further expansion, this model can be used to predict the fatigue life of materials and tissues.

BIBLIOGRAPHY

[1] U. G. Goranson, "Fatigue issues in aircraft maintenance and repairs," *International Journal of Fatigue*, vol. 20, no. 6, pp. 413–431, 1998.

[2] W. Schütz, "A history of fatigue," *Engineering fracture mechanics*, vol. 54, no. 2, pp. 263–300, 1996.

[3] M. Miner *et al.*, "Cumulative fatigue damage," *Journal of applied mechanics*, vol. 12, no. 3, pp. A159–A164, 1945.

[4] P. Paris and F. Erdogan, "A critical analysis of crack propagation laws," *Journal of basic engineering*, vol. 85, no. 4, pp. 528–533, 1963.

[5] R. I. Stephens, A. Fatemi, R. R. Stephens, and H. O. Fuchs, *Metal fatigue in engineering*. John Wiley & Sons, 2000.

[6] W. Curtin and H. Scher, "Brittle fracture in disordered materials: A spring network model," *Journal of Materials Research*, vol. 5, no. 3, pp. 535–553, 1990.

[7] D. J. Durian, "Bubble-scale model of foam mechanics: mmelting, nonlinear behavior, and avalanches," *Physical Review E*, vol. 55, no. 2, p. 1739, 1997.

[8] A. Poupon, "Voronoi and voronoi-related tessellations in studies of protein structure and interaction," *Current opinion in structural biology*, vol. 14, no. 2, pp. 233–241, 2004.

[9] J. R. Shewchuk, "Tetrahedral mesh generation by delaunay refinement," in *Symposium on Computational Geometry*, pp. 86–95, 1998.

[10] A. Frieze and M. Karoński, *Introduction to random graphs*. Cambridge University Press, 2015.

[11] B. Bollobás and B. Béla, *Random graphs*. No. 73, Cambridge university press, 2001.

[12] http://people.ece.umn.edu/users/mihailo/software/leaders/random_network.html, 2019. Online; accessed 17 June 2019.

[13] H. Petroski, *Invention by design*. Universities Press, 1996.

[14] B. Mohar, Y. Alavi, G. Chartrand, and O. Oellermann, "The laplacian spectrum of graphs," *Graph theory, combinatorics, and applications*, vol. 2, no. 871-898, p. 5, 1991.

[15] G. Makov and M. Payne, "Periodic boundary conditions in ab initio calculations," *Physical Review B*, vol. 51, no. 7, p. 4014, 1995.

[16] S. H. Simon, *The Oxford solid state basics*. OUP Oxford, 2013.

[17] C. Truesdell and W. Noll, "The non-linear field theories of mechanics," in *The non-linear field theories of mechanics*, p. 48, Springer, 2004.

[18] F. Irgens, *Continuum mechanics*. Springer Science & Business Media, 2008.

[19] J. H. Snoeijer, T. J. Vlugt, M. van Hecke, and W. van Saarloos, "Force network ensemble: a new approach to static granular matter," *Physical review letters*, vol. 92, no. 5, p. 054302, 2004.

[20] B. P. Tighe, J. H. Snoeijer, T. J. Vlugt, and M. van Hecke, "The force network ensemble for granular packings," *Soft Matter*, vol. 6, no. 13, pp. 2908–2917, 2010.

[21] A. Fatemi and L. Yang, "Cumulative fatigue damage and life prediction theories: a survey of the state of the art for homogeneous materials," *International journal of fatigue*, vol. 20, no. 1, pp. 9–34, 1998.

[22] "What is a sn-curve?." https://community.plm.automation.siemens.com/t5/Testing-Knowledge-Base/What-is-a-SN-Curve/ta-p/355935, March 2019. Online; accessed 5 June 2019.

[23] J. Lubliner, *Plasticity theory*. Courier Corporation, 2008.

[24] L. D. Landau, E. M. Lifshitz, V. Berestetskii, and L. Pitaevskii, *Course of Theoretical Physics: Theory of Elasticity*. 1995.

[25] R. Bridson, "Fast poisson disk sampling in arbitrary dimensions.," in *SIGGRAPH sketches*, p. 22, 2007.

[26] E. Specht, "The best known packings of equal circles in a square (up to n= 10000)," 2015.

[27] B. Delaunay *et al.*, "Sur la sphere vide," *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, vol. 7, no. 793-800, pp. 1–2, 1934.

[28] J. Meijering, "Interface area, edge length, and number of vertices in crystal aggregates with random nucleation," *Philips Res. Rep.*, vol. 8, pp. 270–290, 1953.

[29] V. Bilonenko, "2d delaunay triangulation library for c++." https://github.com/delfrrr/delaunator-cpp, 2018.

[30] American Mathematical Society, "Percolation: Slipping through the cracks." http://www.ams.org/publicoutreach/feature-column/fcarc-percolation.

[31] O. Basquin, "The exponential law of endurance tests," in *Proc Am Soc Test Mater*, vol. 10, pp. 625–630, 1910.

[32] I. Šulák, K. Obrtlík, and L. Čelko, "High-temperature low-cycle fatigue behaviour of hip treated and untreated superalloy mar-m247," *Kovove Materialy*, vol. 54, pp. 471–481, 12 2016.

[33] I. Hughes and T. Hase, *Measurements and their uncertainties: a practical guide to modern error analysis*. Oxford University Press, 2010.

[34] A. Carpinteri, *Structural mechanics: a unified approach*. CRC Press, 2017.

[35] J. C. Maxwell, "L. on the calculation of the equilibrium and stiffness of frames," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 27, no. 182, pp. 294–299, 1864.

[36] Z. Gong, M. Aldeen, and L. Elsner, "A note on a generalized cramer's rule," *Linear Algebra and its Applications*, vol. 340, no. 1, pp. 253 – 254, 2002.

[37] M. K. Enns, W. F. Tinney, and F. L. Alvarado, "Sparse matrix inverse factors (power systems)," *IEEE Transactions on Power Systems*, vol. 5, no. 2, pp. 466–473, 1990.

A C++ ALGORITHM

On the next few pages a full implementation of the algorithm can be found. The algorithm can also create an OpenGL window tho visualize the stretching of the system. Some logical operations can be improved, this is discussed in chapter 5. The parts with the greatest complexity is the Gaussian elimination and the while loop in the ”Stretch the system” part. Two possible methods to improve the speed of this part of the algorithm are named and discussed in chapter 5.

```cpp
#include <iostream>     /* cout */
#include <fstream>
#include <time.h>       /* time */
#include <algorithm>    /* min */
#include <math.h>       /* log2 */
#include <vector>
#include <C:\Users\Ruben ter Meulen\source\repos\delaunator-cpp-master\include\delaunator.hpp> //https://github.com/delfrrr/delaunator-cpp
#include <cstdio>
#include <chrono>
#include <random>
#include <GL/glew.h>
#include <GLFW/glfw3.h>

//parameters
unsigned int n;     //it is convention to take a n as a power of 2
double sigma_max;   //maximum stress
double deps ;       //step size, small enough that the sytem does not break at the edge
double scale;       //for the spring constant, always set to one

//openGL options
constexpr int SCREEN_WIDTH = 1280;
constexpr int SCREEN_HEIGHT = 720;
const bool graph = 0;
const double Lx_max = .0;

//variables for setting the vertices and for the Delaunay triangulation
double xtest; double ytest; double xadd; double yadd; double min_d;
std::vector<double> coords;
bool isLeft; bool isRight; bool isUp; bool isDown;
std::vector<int> n_orig; std::vector<double> dx; std::vector<double> dy;
double ninex[9] = { 0,0,1,1, 1, 0,-1,-1,-1 };
double niney[9] = { 0,1,1,0,-1,-1,-1, 0, 1 };

//stretching variables
const double K_crit = 0.1;              //if the spring constant is 10 percent or less of it's original value it breaks
unsigned int kdel; bool suc;
double Lx = 1; bool up = 1; bool down = 0;
unsigned int ncycle = 1; unsigned int jv;
unsigned int kadd; double Kji;
double delta_x; double delta_y;

//variables for using OpenGL
std::vector<GLdouble> eps_sigma; GLdouble sigma_add;
GLdouble* lineVertices = new GLdouble[4];
GLFWwindow *window;
GLdouble lineVertices2[] = {
    0,0,1,0,
    0,0,0,1,
    1,1,0,1,
    1,1,1,0
```

```
51  };
52
53  //variables for pruning of the system
54  unsigned int imax; unsigned int jmax; double test = 0;
55  double maxl = 0; double upperl = 1.1;
56  unsigned int ncon; unsigned int nsprings = 0;
57  unsigned int imax2; unsigned int jmax2;
58
59  FILE *file_read;
60  double eps_b;
61  int main() {
62      auto start = std::chrono::steady_clock::now();
63      //read input variables
64      fopen_s(&file_read, "C:\\Users\\Ruben ter Meulen\\source\\repos\
          \fasteralg\\fasteralg\\inp.txt","r");
65      fscanf_s(file_read, "%d" , &n);
66      fscanf_s(file_read, "%lf", &sigma_max);
67      fscanf_s(file_read, "%lf", &deps);
68      fscanf_s(file_read, "%lf", &scale);
69      fclose(file_read);
70      double eps = (double) .6 / n;    //explained in numerical methods
71      const double delta = 1;          //copy the whole system instead of a
          only some parts
72      unsigned int ntot = n;
73      double* y = new double[n]; double* x = new double[n];
74      double* y_text = new double[n]; double* x_text = new double[n];
75      std::vector < std::vector<double>> spring_length(n);
76      double** K = new double*[n];
77      double** K_text = new double*[n];
78      double** path_length = new double*[n];
79      double** old_spring_length = new double*[n];
80      double* Fx = new double[n]; double* Fy = new double[n];
81      //create node points with a distance more than eps away from each other
82      std::mt19937 randomGen((unsigned int) time(NULL));
83      for (size_t i = 0; i < n; i++){
84          min_d = 0; //0 < eps
85          while (min_d < eps){
86              xadd = (double)randomGen() / (double)randomGen.max();
87              yadd = (double)randomGen() / (double)randomGen.max();
88              min_d = 2; //2 > eps
89              for (size_t j = 0; j < i; j++){
90                  for (size_t k = 0; k < 9; k++){
91                      xtest = xadd + ninex[k];
92                      ytest = yadd + niney[k];
93                      min_d = std::min(min_d, pow(xtest  - x[j], 2) + pow(ytest
                        - y[j], 2));
94                  }
95              }
96          }
97          x[i] = xadd; y[i] = yadd;
98          n_orig.push_back(i); dx.push_back(0); dy.push_back(0);
99          coords.push_back(x[i]); coords.push_back(y[i]);
100     }
```

```
101     //periodic boundary conditions
102     for (size_t i = 0; i < n; i++){
103         isDown = 0; isUp = 0; isLeft = 0; isRight = 0;
104         if (x[i] < delta) {
105             isLeft = 1; ntot++;
106             n_orig.push_back(i); dx.push_back(1); dy.push_back(0);
107             coords.push_back(x[i] + 1); coords.push_back(y[i]);
108         }
109         if (x[i] > 1 - delta) {
110             isRight = 1; ntot++;
111             n_orig.push_back(i); dx.push_back(-1); dy.push_back(0);
112             coords.push_back(x[i] - 1); coords.push_back(y[i]);
113         }
114         if (y[i] < delta) {
115             isDown = 1; ntot++;
116             n_orig.push_back(i); dx.push_back(0); dy.push_back(1);
117             coords.push_back(x[i]); coords.push_back(y[i] + 1);
118         }
119         if (y[i] > 1 - delta) {
120             isUp = 1; ntot++;
121             n_orig.push_back(i); dx.push_back(0); dy.push_back(-1);
122             coords.push_back(x[i]); coords.push_back(y[i] - 1);
123         }
124         if (isUp && isRight) {
125             ntot++;
126             n_orig.push_back(i); dx.push_back(-1); dy.push_back(-1);
127             coords.push_back(x[i] - 1); coords.push_back(y[i] - 1);
128         }
129         if (isDown && isRight) {
130             ntot++;
131             n_orig.push_back(i); dx.push_back(-1); dy.push_back(1);
132             coords.push_back(x[i] - 1); coords.push_back(y[i] + 1);
133         }
134         if (isDown && isLeft) {
135             ntot++;
136             n_orig.push_back(i); dx.push_back(1); dy.push_back(1);
137             coords.push_back(x[i] + 1); coords.push_back(y[i] + 1);
138         }
139         if (isUp && isLeft) {
140             ntot++;
141             n_orig.push_back(i); dx.push_back(1); dy.push_back(-1);
142             coords.push_back(x[i] + 1); coords.push_back(y[i] - 1);
143         }
144     }
145
146     //create adjacency list using the delaunay triangulation O(n log(n)+9*
        (6*2)*n)
147     delaunator::Delaunator dt(coords);
148     std::vector < std::vector<int>> adj_list(ntot);
149     for (size_t i = 0; i < dt.triangles.size(); i += 3){
150         adj_list[dt.triangles[i]].push_back(dt.triangles[i + 1]); adj_list
            [dt.triangles[i]].push_back(dt.triangles[i + 2]);
151         adj_list[dt.triangles[i + 1]].push_back(dt.triangles[i]); adj_list
```

```
            [dt.triangles[i + 1]].push_back(dt.triangles[i + 2]);
152         adj_list[dt.triangles[i + 2]].push_back(dt.triangles[i]); adj_list
            [dt.triangles[i + 2]].push_back(dt.triangles[i + 1]);
153     }
154     for (size_t i = 0; i < n; i++){
155         for (size_t j = 0; j < adj_list[i].size(); j++){
156             suc = false; kdel = j+1;
157             while (!suc && kdel < adj_list[i].size()){
158                 if (adj_list[i][j] == adj_list[i][kdel] ){
159                     adj_list[i].erase(adj_list[i].begin()+kdel);
160                     suc = true;
161                 }
162                 kdel++;
163             }
164         }
165     }
166
167     for (size_t i = 0; i < n; i++) {
168         old_spring_length[i] = new double[adj_list[i].size()];
169         path_length[i] = new double[adj_list[i].size()];
170         for (size_t j = 0; j < adj_list[i].size(); j++) {
171             spring_length[i].push_back(sqrt(pow(x[i] - x[n_orig[adj_list[i]
              [j]]] - dx[adj_list[i][j]], 2) + pow(y[i] - y[n_orig[adj_list
              [i][j]]] - dy[adj_list[i][j]], 2)));
172             nsprings++;
173         }
174     }
175
176     //create opengl window
177     if (graph)
178     {
179         if (!glfwInit())
180         {
181             return -1;
182         }
183         window = glfwCreateWindow(SCREEN_WIDTH, SCREEN_HEIGHT, "OpenGL",
              NULL, NULL);
184         if (!window)
185         {
186             glfwTerminate();
187             return -1;
188         }
189         glfwMakeContextCurrent(window);
190         glViewport((GLint) 0.0f, (GLint) 0.0f, SCREEN_HEIGHT,
              SCREEN_HEIGHT);
191         glMatrixMode(GL_PROJECTION);
192         glLoadIdentity();
193         glOrtho(-.5-Lx_max, 1.5+Lx_max, -.5, 1.5, 0, 1);
194         glMatrixMode(GL_MODELVIEW);
195         glLoadIdentity();
196     }
197
198     //stretch the system
```

```
199     for (size_t i = 0; i < n; i++) {
200         K[i] = new double[n];
201         K_text[i] = new double[n];
202         Fx[i] = 0; Fy[i] = 0;
203         for (size_t j = 0; j < n; j++)
204             K[i][j] = 0;
205         for (size_t j = 0; j < adj_list[i].size(); j++) {
206             path_length[i][j] = spring_length[i][j];
207             old_spring_length[i][j] = spring_length[i][j];
208             jv = adj_list[i][j];
209             K[i][n_orig[jv]] += -spring_length[i][j] / path_length[i][j];
210             K[i][i] -= K[i][n_orig[jv]];
211             Fx[i] += -K[i][n_orig[jv]] * (spring_length[i][j] * ((x[i] - x
                  [n_orig[jv]] - Lx * dx[jv]) / old_spring_length[i][j]) + Lx *
                  dx[jv]);
212             Fy[i] += -K[i][n_orig[jv]] * (spring_length[i][j] * ((y[i] - y
                  [n_orig[jv]] - dy[jv]) / old_spring_length[i][j]) + dy[jv]);
213         }
214     }
215     while (true){
216         //Gaussian elimination O(n^3)
217         for (size_t i = 0; i < n - 1; i++) {
218             for (size_t j = i + 1; j < n; j++) {
219                 if (K[j][i] != 0) {
220                     Kji = K[j][i] / K[i][i];
221                     Fx[j] -= Kji * Fx[i] ;
222                     Fy[j] -= Kji * Fy[i] ;
223                     for (size_t k = 0; k < n; k++)
224                         if (k != i && K[i][k] != 0)
225                             K[j][k] -= Kji * K[i][k];
226                     K[j][i] = 0;
227                 }
228             }
229         }
230         for (int i = n - 2; i > -1; i--) {
231             for (size_t j = i+1; j < n; j++) {
232                 Fx[i] -= K[i][j] * x[j];
233                 Fy[i] -= K[i][j] * y[j];
234             }
235             x[i] = Fx[i] / K[i][i];
236             y[i] = Fy[i] / K[i][i];
237         }
238
239         //update spring constant
240         sigma_add = 0;
241         for (size_t i = 0; i < n; i++) {
242             Fx[i] = 0; Fy[i] = 0;
243             for (size_t j = 0; j < n; j++)
244                 K[i][j] = 0;
245             for (size_t j_it = 0; j_it < adj_list[i].size(); j_it++) {
246                 jv = adj_list[i][j_it];
247                 delta_x = (x[i] - x[n_orig[jv]] - Lx * dx[jv]);
248                 delta_y = y[i] - y[n_orig[jv]] - dy[jv];
```

```
249                 path_length[i][j_it] += scale * abs(old_spring_length[i]
                      [j_it] - sqrt(pow(delta_x, 2) + pow(delta_y, 2)));
250                 old_spring_length[i][j_it] = sqrt(pow(delta_x, 2) + pow
                      (delta_y, 2));
251                 if (spring_length[i][j_it] / path_length[i][j_it] > K_crit)
                      {
252                     K[i][n_orig[jv]] += -spring_length[i][j_it] /
                        path_length[i][j_it];
253                     K[i][i] -= K[i][n_orig[jv]];
254                     Fx[i] += -K[i][n_orig[jv]] * (spring_length[i][j_it] *
                        ((delta_x) / old_spring_length[i][j_it]) + Lx * dx[jv]);
255                     Fy[i] += -K[i][n_orig[jv]] * (spring_length[i][j_it] *
                        ((delta_y) / old_spring_length[i][j_it]) + dy[jv]);
256                     sigma_add += -K[i][n_orig[jv]] * (delta_x) *
257                         (delta_x - spring_length[i][j_it] * (delta_x) /
                        old_spring_length[i][j_it]);
258                 }
259             }
260         }
261         sigma_add = sigma_add / 2; //counted all springs twice
262         eps_sigma.push_back(Lx - 1);
263         eps_sigma.push_back(sigma_add);
264         //plot some things
265         if (graph){
266             glfwMakeContextCurrent(window);
267             glClear(GL_COLOR_BUFFER_BIT);
268             glEnableClientState(GL_VERTEX_ARRAY);
269             for (size_t i = 0; i < n; i++){
270                 for (size_t j_it = 0; j_it < adj_list[i].size(); j_it++){
271                     if (spring_length[i][j_it] / path_length[i][j_it] >
                        K_crit) {
272                         jv = adj_list[i][j_it];
273                         lineVertices[0] = x[n_orig[i]] + Lx * dx[i];
274                         lineVertices[1] = y[n_orig[i]] + dy[i];
275                         lineVertices[2] = x[n_orig[jv]] + Lx * dx[jv];
276                         lineVertices[3] = y[n_orig[jv]] + dy[jv];
277                         glColor3d(1 + K[i][n_orig[jv]], -K[i][n_orig[jv]],
                        0);
278                         glVertexPointer(2, GL_DOUBLE, 0, lineVertices);
279                         glDrawArrays(GL_LINES, 0, 4);
280                     }
281                 }
282             }
283             glColor3f(1, 1, 1);
284             glVertexPointer(2, GL_DOUBLE, 0, lineVertices2);
285             glDrawArrays(GL_LINES, 0, 8);
286             glDisableClientState(GL_VERTEX_ARRAY);
287             glfwSwapBuffers(window);
288             glfwPollEvents();
289         }
290
291         //move the system up and down
292         if (sigma_add < sigma_max && up){
```

```
293             Lx += deps;
294         }
295         else if(sigma_add > sigma_max && up){
296             down = 1; up = 0;
297             eps_b = Lx - 1;
298         }
299         else if (Lx - 1. > 0. && down){
300             Lx -= deps;
301         }
302         else{
303             up = 1; down = 0;
304 
305             ncycle++;
306         }
307         if ((Lx > 2. && sigma_add < 0.001) || isnan(sigma_add))
308             break;
309         if (eps_sigma.size() > 5)
310         {
311             if (eps_sigma[eps_sigma.size() - 1] > eps_sigma[eps_sigma.size()
                - 3])
312             {
313                 for (size_t i = 0; i < n; i++)
314                 {
315                     x_text[i] = x[i]; y_text[i] = y[i];
316                     for (size_t j = 0; j < adj_list[i].size(); j++)
317                         K_text[i][n_orig[adj_list[i][j]]] = K[i][n_orig
                        [adj_list[i][j]]];
318                 }
319             }
320         }
321 
322 
323     }
324     if(graph)
325         glfwTerminate();
326 
327     std::ofstream file_write("C:\\Users\\Ruben ter Meulen\\source\\repos\
        \fasteralg\\fasteralg\\outp.txt", std::ofstream::trunc |
        std::fstream::in | std::fstream::out);
328     file_write << ncycle << '\t' << eps_b << '\t' <<
329         deps << '\t' << sigma_max << '\t' << nsprings << '\t';
330     for (size_t i = 0; i < eps_sigma.size(); i++)
331         file_write << eps_sigma[i] << '\t';
332     file_write << std::flush;
333     file_write.close();
334     std::ofstream file_write2("C:\\Users\\Ruben ter Meulen\\source\\repos\
        \fasteralg\\fasteralg\\graph.txt", std::ofstream::trunc |
        std::fstream::in | std::fstream::out);
335     for (size_t i = 0; i < n; i++) {
336         for (size_t j_it = 0; j_it < adj_list[i].size(); j_it++) {
337             jv = adj_list[i][j_it];
338             if (-K_text[i][n_orig[jv]] > K_crit) {
339                 file_write2 << x_text[n_orig[i]] + Lx*dx[i] << '\t' <<
```

```
                    y_text[n_orig[i]] + dy[i] << '\t' << x_text[n_orig[jv]] +
                    Lx*dx[jv] << '\t' << y_text[n_orig[jv]] + dy[jv] << '\t'
                    <<
340                   1 + K_text[i][n_orig[jv]] << '\t' << -K_text[i][n_orig
                     [jv]] << '\t' << 0 << '\t';
341             }       //all edges are save twice, it is better to replace this
                   by saving edges instead of the vertices
342         }
343     }
344     file_write2 << std::flush;
345     file_write2.close();
346 }
```

COLOPHON

This document was typeset using LaTeX. The document layout was generated using the `arsclassica` package by Lorenzo Pantieri, which is an adaption of the original `classicthesis` package from André Miede.


TU Delft