Nearly Isostatic Networks and Allosteric Effects

Ian McKenzie

April 2020

Abstract

There has been a lot of recent research interest in tuning the properties of spring networks. This project specifically focuses on the ability to tune the strain response of one part of the network in response to a strain applied to a remote part of the network. In this project I explore the fascinating mathematics of rigidity theory and implement a computational model for working with spring networks in Python. I then attempt to recreate the results of a recent paper, *Designing allostery-inspired response in mechanical networks* [1], with some slight tweaks.

Contents

1 Introduction

A framework in the mathematical sense is effectively a drawing of a graph with physics. That is, we consider a drawing of a graph embedded in a d-dimensional space and suppose that its elements can move in some way. In this project we will consider only $\mathbb{R}^2$ and 'bar-and-joint' frameworks (hereafter simply frameworks). This means that the vertices can move freely in the space under the constraint that the distance between any pair of vertices connected by an edge stays constant (i.e. the edges have fixed length). We say that a framework is *rigid* if the distance between all pairs of vertices stays the same through any allowed movement, not just between those connected by an edge.

Frameworks are not only theoretically interesting objects of study, but are also useful in analysing many physical situations. Classic examples include scaffolding and architecture, where a rigid construction is vital for safety and longevity. Other interesting areas that are related include the structures of proteins, which are essentially molecular frameworks.

It is this link to biology that provides the inspiration for this project. Enzymes are proteins that catalyse reactions. They have an *active site* where the substrate upon which the enzyme acts binds to it. Some also have a regulatory or *allosteric site*, where a different molecule can bind to the enzyme. This changes the shape of the active site – which need not be close to the allosteric site – inhibiting its catalysis. The aim of this project is to understand this "action at a distance" as it occurs in almost-rigid frameworks. In doing so we closely follow a 2017 paper, *Designing allostery-inspired response in mechanical networks* by Rocks et al.[1], hereafter referred to as Rocks et al.

2 Background

In this section we look at the mathematics of rigid frameworks (including a definition for framework) and outline the linear algebra-based formalism we will be using.

2.1 Rigidity theory

We start with definitions for frameworks and motions of frameworks before stating a few preliminary theorems. Much of this theory is drawn from Jack E. Graver's book *Counting on Frameworks* [2]. We then go into the paper on which this project is based.

2.1.1 Frameworks and Rigidity

Definition 2.1. *A* framework $\mathcal{F} = (G, \mathbf{p})$ *is a graph* $G = (V, E)$ *with an embedding* $\mathbf{p}$ *in a d-dimensional Euclidean space, here taken to be* $\mathbb{R}^2$ *unless otherwise specified.* G *is called the* structure graph *of* $\mathcal{F}$*. We call the* N *elements of* $V = \{a_0, a_1, ..., a_{N-1}\}$ *vertices and the* N_b *elements of* $E = \{e_0, e_1, ..., e_{N_b-1}\}$ *edges, bars, or bonds. Then the embedding* $\mathbf{p} \in \mathbb{R}^{dN}$ *is a vector composed of* N $d \times 1$ *vectors stacked, with the* i*th sub-vector* $\mathbf{p}_i$ *corresponding to the position of node* a_i*.*

In keeping with the conception of frameworks as being constructed of inflexible bars and perfect joints, any movement of the framework (a 'motion') will preserve the length of the 'bars' (edges).

Definition 2.2. *A* motion *of a framework* $\mathcal{F} = (G, \mathbf{p})$ *in* $\mathbb{R}^n$ *is a family of embeddings* $\mathbf{p}(t)$*,* $t \in [0, 1]$*, such that*

(i) $\mathbf{p}(0) = \mathbf{p}$, $\mathbf{p}(t)$ *is differentiable* $\forall t$*, and*

(ii) the Euclidean distance $d : \mathbb{R}^n \to \mathbb{R}$ *between the endpoints of any edge is constant. That is,* $\forall e = (a_i, b_j) \in E$*,* $\forall t \in [0, 1]$*,* $d(\mathbf{p}_i(t),\ \mathbf{p}_j(t)) = d(\mathbf{p}_i, \mathbf{p}_j)$*.*

A rigid motion is then a motion where the relative positions of the vertices is preserved, i.e. the distance between all vertices is preserved, not just those joined by an edge. This is contrasted with a deformation, where the distance between some pair of vertices is different and so the shape is changed. A rigid framework is then one that admits no deformations.

Definition 2.3. *A* rigid motion *of a framework* $\mathcal{F} = (G, \mathbf{p})$ *in* $\mathbb{R}^n$ *is a motion where additionally* $\forall u, v \in V$*,* $\forall t \in [0, 1]$*,* $d(p_u(t),\ p_v(t)) = d(p_u, p_v)$*. A motion is a* deformation *if this property does not hold.* $\mathcal{F}$ *is a* rigid framework *if all its allowed motions are rigid.*

Definition 2.4. *A* rigid component *of a framework is a maximal rigid sub-framework. A framework is rigid if it has just one rigid component (the whole framework).*

2.1.2 Infinitesimal Rigidity

There is an almost equivalent concept that turns out to be a lot more useful computationally: infinitesimal rigidity. The idea is to look at the effect that infinitesimal motions have on the lengths of the edges. Instead of defining motions as trajectories of the vertices, here we look at velocities applied to the vertices. We want these infinitesimal velocities to "cancel out" so that the overall effect is that the bars stay the same length. In keeping with the previous pattern of rigidity, infinitesimal rigid motions will maintain this cancelling out for all pairs of vertices, and infinitesimal deformations will change this distance.

Definition 2.5. *An* infinitesimal motion *of a framework $\mathcal{F} = (G = (V, E), \mathbf{p})$ is a function $\mathbf{q} : V \to \mathbb{R}^n$ such that $\forall (a_i, a_j) \in E, (\mathbf{p}_i - \mathbf{p}_j)(\mathbf{q}_i - \mathbf{q}_j) = 0$, where $\mathbf{q}_i = \mathbf{q}(a_i)$ is the velocity of the ith node. An infinitesimal motion is an* infinitesimal rigid motion *if additionally $\forall (a_i, a_j) \in V \times V, (\mathbf{p}_i - \mathbf{p}_j)(\mathbf{q}_i - \mathbf{q}_j) = 0$. An infinitesimal motion is an* infinitesimal deformation *if $\exists (a_i, a_j) \in V \times V$ s.t. $(\mathbf{p}_i - \mathbf{p}_j)(\mathbf{q}_i - \mathbf{q}_j) \neq 0$. A framework $\mathcal{F}$ is* infinitesimally rigid *if it only admits infinitesimal rigid motions.*

These definitions are useful as they allow us to consider linear equations rather than quadratic. The following theorem justifies using infinitesimal rigidity instead.

Theorem 2.6. *Let $\mathcal{F}$ be a framework. If $\mathcal{F}$ is infinitesimally rigid then $\mathcal{F}$ is rigid.*

[I need to add more background about the rigidity matrix, and the relationship to combinatorial rigidity and general position]

The following definition of a rigidity matrix is very useful for working with frameworks computationally. I will define it for two dimensions, but higher-dimensional cases are analogous.

Definition 2.7. *Let $\mathcal{F} = (V, E, \mathbf{p})$ be a 2-dimensional framework. The* rigidity matrix *R of $\mathcal{F}$ is an $N_b \times dN$ matrix. The ith row corresponds to the ith bond, e_i. Suppose that e_i has nodes a_j and a_k. Then $R_{i,2j-1} = (x_j - x_k), R_{i,2j} = (y_j - y_k), R_{i,2k-1} = (x_k - x_j), R_{i,2k} = (y_k - y_j)$, with zeros elsewhere in the row.*

The reason the rigidity matrix is useful is that it encodes information about all the bonds of the network. Two nodes share a bond if and only if a row of the rigidity matrix has non-zero entries in the columns corresponding to their positions. The degree of a node is the number of rows in which the column corresponding to its x-

or y-coordinate is non-zero. Row i, corresponding to $e_i = (a_j, a_k)$, can be thought of as consisting of two row vectors, $(\mathbf{p}_j - \mathbf{p}_k)^\top$ and $(\mathbf{p}_k - \mathbf{p}_j)^\top$, padded by zeros. These row vectors 'point' from a_k to a_j and from a_j to a_k respectively.

2.2 Linear algebra formalism

Rocks et al. uses a substantial amount of linear algebra that is lightly explained, and we provide more details here. We will use this same formalism throughout. This formalism is specifically designed for working with a framework where each bond is a perfect Hookean spring. We shall often call such objects 'spring networks'.

2.2.1 Background concepts

There are some results from linear algebra that will come in handy later on. The first of these is the singular value decomposition.

Theorem 2.8 (Singular Value Decomposition (SVD))**.** *Let A be an $m \times n$ matrix with entries from a field F. Then A can be written as*

$$A = U\Sigma V^*$$

,

where Σ is an $m \times n$ rectangular diagonal matrix (that is, it only has non-zero entries on the main diagonal) with positive real numbers on the diagonal, and U and V are $m \times m$ and $n \times n$ unitary matrices, respectively. Further, if $F = \mathbb{R}$, then $V^* = V^\top$, and U and V are orthonormal matrices.

The second useful concept is the Moore-Penrose inverse, or pseudoinverse [3]:

Definition 2.9 (Moore-Penrose Generalised Inverse)**.** *Let A be an $m \times n$ matrix with entries from a field F. Then $A^\dagger$ is a pseudoinverse for A if:*

1. $AA^\dagger A = A$
2. $A^\dagger AA^\dagger = A^\dagger$
3. $(AA^\dagger)^* = AA^\dagger$
4. $(A^\dagger A)^* = A^\dagger A$

In other words, $AA^\dagger$ is Hermitian and acts like the identity on the column space of A, and $A^\dagger A$ is also Hermitian and acts like the identity on the column space of $A^\dagger$.

Note that if A is an invertible matrix, its inverse A^{-1} satisfies all of these conditions (as $AA^{-1} = A^{-1}A = I$).

Informally, $A^\dagger$ acts like a real inverse on the image of A, and sends everything else to zero, and A does the same for $A^\dagger$.

The SVD gives us a convenient way to calculate the pseudoinverse of a matrix.

Theorem 2.10. *Let A be an $m \times n$ matrix with entries from a field F. Then*

$$A^\dagger = V\Sigma^\dagger U^*$$

,

where $A = U\Sigma V^*$ is the SVD of A. To take the pseudoinverse of Σ, a rectangular diagonal matrix, replace each non-zero element with its reciprocal and transpose.

Proof. See for example [4]. □

We will also need a result about the column-space of matrices multiplied by their transposes.

Lemma 2.11. *Let A be an $m \times n$ matrix with entries from $\mathbf{R}$, and let $row(A)$ and $col(A)$ denote the row- column-spaces of A respectively.*

Then $col(A) = col(AA^\top) = row(AA^\top)$ and $row(A) = col(A^\top A) = row(A^\top A)$.

Proof. First note $AA^\top$ and $A^\top A$ are symmetric, so their row- and column-spaces are identical.

Let $x \in ker(A)$. Then

$$\begin{aligned} Ax &= \mathbf{0} \\ A^\top Ax &= A^\top \mathbf{0} \\ (A^\top A)x &= \mathbf{0} \end{aligned}$$

Thus $null(A^\top A) \subseteq ker(A)$.

Now let $x \in ker(A^\top A)$. Then

$$\begin{aligned} (A^\top A)x &= \mathbf{0} \\ x^\top A^\top Ax &= x^\top \mathbf{0} \\ (Ax)^\top Ax &= 0 \\ \implies \|Ax\| &= 0 \\ \implies Ax &= \mathbf{0} \end{aligned}$$

Thus $ker(A) \subseteq ker(A^\top A)$, and so they are equal.

By the Rank-Nullity Theorem, an $m \times n$ matrix M has $n = rank(M) - null(M)$, where $rank(M)$ is the dimension of its column- or row-space, and $null(M)$ is the dimension of its kernel.

Since $A^\top A$ is an $n \times n$ matrix, and $null(A^\top A) = null(A)$, we get that $rank(A^\top A) = rank(A)$.

Each column of $A^\top A$ is a linear combination of the columns of $A^\top$ (which are the rows of A), so $row(A^\top A) \subseteq row(A)$. If one linear space is a subspace of the other and they both have equal dimensions, then they must be equal. Therefore $row(A^\top A) = row(A)$.

A symmetrical argument gives $col(AA^\top) = col(A)$. □

Here is another result related to the columns of a matrix

Theorem 2.12. *Let A be an $m \times n$ matrix with entries from $\mathbb{R}$ and full column-rank, and let D be an $n \times n$ diagonal matrix with diagonal entries $d_1, d_2, \ldots, d_n \in \mathbb{R}$. If $d_i \neq 0$ then row i of A is in the column-space of $B = ADA^\top$.*

Proof. Taking the columns of A as vectors $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_n$ with entries $\alpha_{i,1}, \ldots, \alpha_{i,m}$, we can write

$$A = \begin{bmatrix} \boldsymbol{\alpha}_1 & \boldsymbol{\alpha}_2 & \ldots & \boldsymbol{\alpha}_n \end{bmatrix} \text{ and } A^\top = \begin{bmatrix} \boldsymbol{\alpha}_1^\top \\ \boldsymbol{\alpha}_2^\top \\ \vdots \\ \boldsymbol{\alpha}_n^\top \end{bmatrix}$$

Then

$$B = ADA^\top = \begin{bmatrix} \boldsymbol{\alpha}_1 & \boldsymbol{\alpha}_2 & \ldots & \boldsymbol{\alpha}_n \end{bmatrix} \begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_1^\top \\ \boldsymbol{\alpha}_2^\top \\ \vdots \\ \boldsymbol{\alpha}_n^\top \end{bmatrix}$$

$$= \begin{bmatrix} \boldsymbol{\alpha}_1 & \boldsymbol{\alpha}_2 & \ldots & \boldsymbol{\alpha}_n \end{bmatrix} \begin{bmatrix} d_1\boldsymbol{\alpha}_1^\top \\ d_2\boldsymbol{\alpha}_2^\top \\ \vdots \\ d_n\boldsymbol{\alpha}_n^\top \end{bmatrix}$$

$$= \begin{bmatrix} \boldsymbol{\beta}_1 & \boldsymbol{\beta}_2 & \ldots & \boldsymbol{\beta}_n \end{bmatrix}$$

,

where $\boldsymbol{\beta}_i = (d_1\alpha_{1,i}\boldsymbol{\alpha}_1 + d_2\alpha_{2,i}\boldsymbol{\alpha}_2 + \ldots + d_n\alpha_{n,i}\boldsymbol{\alpha}_n)$ HHHHHHHHHHHHHHHH(Not sure how to finish this proof?) □

2.2.2 Mechanical formalism

We start by defining how some mechanical terms are used:

Definition 2.13.

> **Extension** *The change in length of a bond due to some applied strain.*
>
> **Tension** *The force along a bond due to an applied strain. In the 1-dimensional case, this is simply a scalar. Positive means the bond is being stretched, negative means compressed.*
>
> **(Engineering) Strain** *Change in length (extension) divided by the equilibrium length.*
>
> **(Net) Force** *Force experienced by the nodes.*
>
> **Displacement** *How much the nodes have been moved from their original positions by some applied strain.*

The goal is to be able to tune the response of a target pair of nodes by manipulating a pair of source nodes. This is formulated as the ratio of the strain on a "ghost bond" of zero stiffness between the target nodes to the strain on a similar bond between the source nodes. We write this ratio as $\eta = \frac{\epsilon_T}{\epsilon_S}$, where ϵ_T is the target strain and ϵ_S is the source strain.

Calculating the full strain would be far more complicated and require a lot more computational power, and so we are only working to linear order. This means we consider the bond lengths and node positions as fixed, while still working with extensions and displacements. Therefore the strains we are working with are only first-order approximations of the true strains. Later, we evaluate the results using the full response.

We define a cost function on this response η based on the ratio of the actual response to the desired response, η^*, with a special case where the desired response is zero:

$$\Delta^2 = \sum_{j=1}^{n} \begin{cases} (\frac{\eta_j}{\eta_j^*} - 1)^2 & \eta_j^* \neq 0 \\ \eta_j^2 & \eta_j^* = 0 \end{cases},$$

where j indexes the source/target pairs. To minimise this cost, we need to calculate how η varies as we remove each bond from the network.

To follow the convention of the paper (and for readability), we define some notation:

Definition 2.14. *Suppose we are in a vector space V, and $a, b \in V$ are vectors. Then $|b\rangle$ is the column vector b, $\langle a|$ is the conjugate transpose of a, and so $\langle a|b\rangle$ is the inner product of a and b and $|a\rangle\langle b|$ is the outer product.*

We define the vectors of bond extensions $|e\rangle$ and of bond tensions $|t\rangle$ in response to the *externally applied strain.* These vectors have length N_b and are defined with respect to the complete orthonormal bond basis $|i\rangle$. Each $|i\rangle$ corresponds to one bond in the network, has length N_b and a 1 in the ith position and 0 everywhere else). We can use these to get the extensions and tensions on any bond:

$$e_i = \langle i|e\rangle \tag{1}$$
$$t_i = \langle i|t\rangle \tag{2}$$

The strain on bond i is $\epsilon_i = \frac{e_i}{l_i}$, where l_i is the equilibrium length of the bond. We model all of the bonds as linear springs, and so they obey Hooke's law (force is proportional to displacement). This is represented by a flexibility matrix F, defined by:

$$\langle i| F |j\rangle = \frac{\delta_{ij}}{k_i}, \tag{3}$$

where $k_i = \frac{\lambda_i}{l_i}$ (with λ_i the stiffness of the bond), and δ_{ij} is the Kronecker delta. This is a diagonal matrix with $\frac{1}{k_i}$ along the diagonal.

This matrix relates the tensions and extensions by

$$|e\rangle = F |t\rangle \tag{4}$$
$$|t\rangle = F^{-1} |e\rangle, \tag{5}$$

which makes sense as the flexibility matrix encodes how easily each bond is stretched (creating an extension) by a force (in this case tension along the bond), according to the linear relationship described by Hooke's law.

Another useful pair of vectors to define are those of node displacements $|u\rangle$ and net forces on the nodes $|f\rangle$. These are related to the bond quantities by the equilibrium matrix Q. Q is the transpose of the rigidity matrix R we defined earlier, with each column divided by the length of the bond to which it corresponds.

Theorem 2.15. *For a framework with equilibrium matrix Q, then the following equations involving tensions, forces, displacements, and extensions hold:*

1. $Q |t\rangle = |f\rangle$

2. $Q^\top |u\rangle = |e\rangle$

Proof.

1. Note that the columns of Q can be thought to each correspond to a bond of our framework. Each pair (in the 2d case) of rows correspond to the position of a node, the first of the pair for the x coordinate, the second for y.

   Consider a node j, and bond $i = (j, k)$. Then row $2j$ contains, in the ith position, the x-coordinate of $\mathbf{p}_j - \mathbf{p}_k$ (call it $x_j - x_k$). If the tension in bond i is t_i, then bond i has a contribution of $(x_j - x_k)t_i$ to the force in the x direction on node j.

2. Note that the rows of $Q^\top$ can be thought of as consisting of a series of row vectors of shape $1 \times d$, where most are $\mathbf{0}$, except for two in each row. For row i, representing bond (j, k), the jth vector is $p_j - p_k$, and similarly the kth vector is $p_k - p_j$.

   The displacement vector $|u\rangle$ can also be thought of as consisting of vectors, this time a stack of column vectors of shape $d \times 1$ where the ith vector is the displacement on node i, $\mathbf{u}_i$. Thus in the product $Q^\top |u\rangle$, the ith element (corresponding to the ith bond) is $(\mathbf{p}_j - \mathbf{p}_k)\mathbf{u}_j + (\mathbf{p}_k - \mathbf{p}_j)\mathbf{u}_k$.

   The first term can be thought of as the projection of $\mathbf{u}_i$ onto the vector pointing from $\mathbf{p}_k$ to $\mathbf{p}_j$. Therefore this gives, to linear order, the change in distance between the nodes (and hence the extension of the bond) due to the displacement of node j.

   The second term is symmetrically the extension due to the displacement of node k, together giving the extension of bond i, as required.

□

HHHHHHHHHHHHHHHH(Not sure about this section, where does half come from???) Because we are considering a system of ideal springs, all the energy in the system is due to the stretching of the bonds. This potential energy stored is equal to the work done in moving each node to its position against the restoring forces of the bonds, which is displacement times force. Because we are working to linear order, we assume that the force is constant throughout (i.e. does not depend on how stretched the spring is). Thus we get:

$$E = \langle u|f\rangle$$

Using the relations described above, we can write this purely in terms of displacement:

$$\begin{aligned} E &= \langle u|f\rangle && (6)\\ &= \langle u| Q |t\rangle && (7)\\ &= \langle u| QF^{-1} |e\rangle && (8)\\ &= \langle u| QF^{-1}Q^{\top} |u\rangle && (9)\\ &= \langle u| H |u\rangle && (10) \end{aligned}$$

where we define the Hessian matrix $H = QF^{-1}Q^{\top}$. HHHHHHHHHHHHHHHH(Explain why H is Hessian?)

According to the principle of least action, when tension is applied to the system, the resulting configuration will be that with the lowest energy. With a set of externally applied tensions $|t^*\rangle$, the minimum energy configuration satisfies (to linear order) HHHHHHHHHHHHHHHH(why?)

$$H |u\rangle = Q |t^*\rangle \tag{11}$$

This can be rearranged to give:

$$|u\rangle = H^{-1}Q |t^*\rangle \tag{12}$$

and recalling Eq. 2, we get the extensions:

$$|e\rangle = Q^{\top} |u\rangle = Q^{\top}H^{-1}Q |t^*\rangle \tag{13}$$

3 Computational model

In order to get an understanding of how the rigidity of a graph changes as you remove edges, I developed a computational model for rigidity in Python to demonstrate applied tensions and resultant stresses, and implemented a pebble game [5] to efficiently detect rigid components in graphs.

3.1 Rigid component experiment

I modified the original pebble game algorithm to give a 'constructive' pebble game. In the original pebble pebble game, the graph is built up by adding edges one at a time until the whole graph is reconstructed. Instead of repeatedly applying this process to the full graph, then for the graph with one edge removed, and so on, the graph is built up edge by edge with the rigid components amended after each edge is added.

In figures 1 and 2 below, rigid components that consist of more than a single edge are shown in bold black, and single-edge components are in grey. Vertices in more than one component are shown in green, and the edge about to be removed is in blue. The removal of the blue edge has a large effect, breaking even some fairly distant rigid components into pieces.

Figure 3 shows how the number of rigid components changes as number of edges changes. Read left to right, it shows rigid components forming and joining together as edges are added. Read right to left, it shows the break-up of the rigid framework into rigid components and then the removal of the remaining individual edges. The edge removal in the two earlier figures corresponds to the sharp jump in 3 just before the 200 mark.

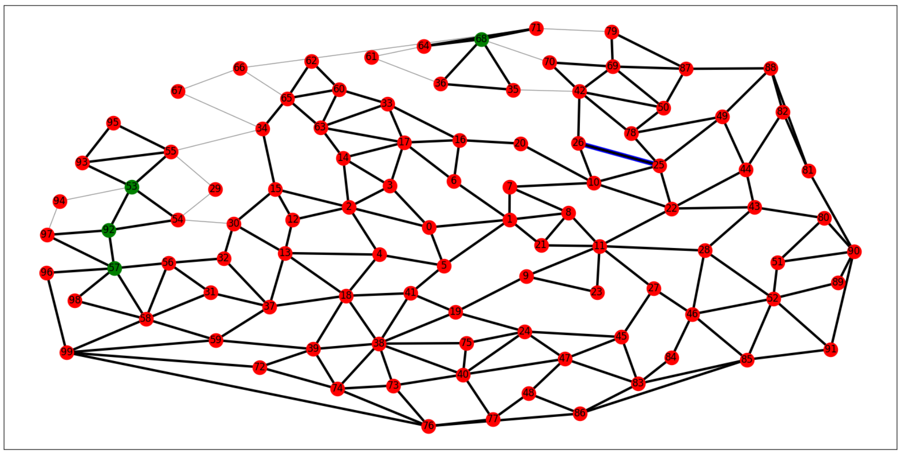

Figure 1: The framework before the removal of the blue edge.

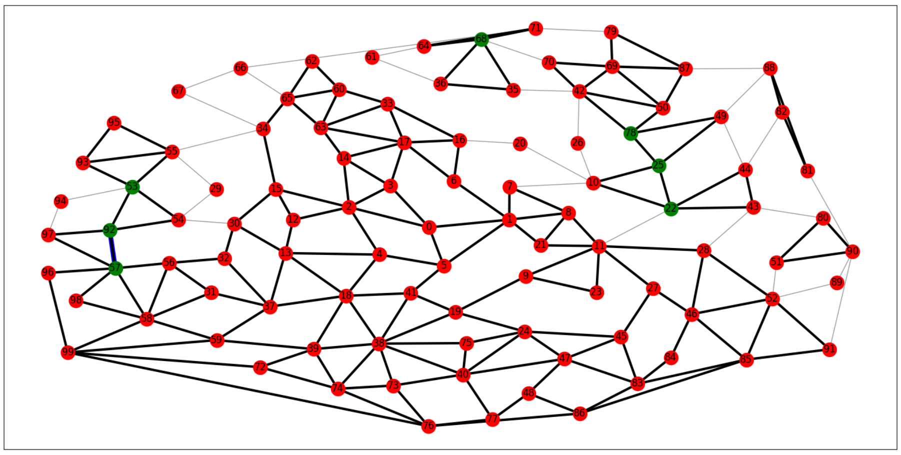
Figure 2: The framework after removal. Note the number of edges that turned grey on the top right.

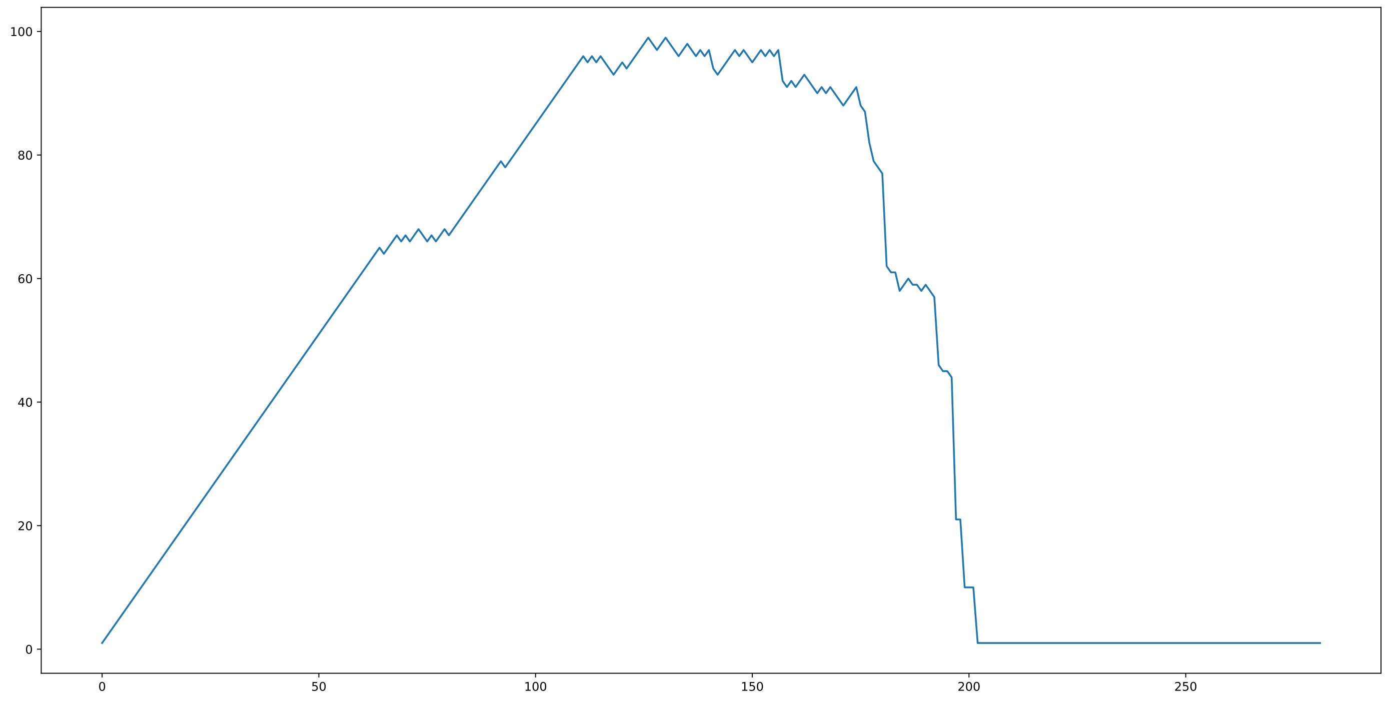

Figure 3: The number of components against the number of edges.

3.2 Applying forces and tensions

One of the main ideas in this project was applying a force or tension to one part of the network, and having it propagate through the whole system. Python code was developed that allowed one to apply forces or tensions to a set of nodes or bonds, respectively, and visualise the stresses that arose in the bonds as a result. In practice we exclusively work with applied tensions, as this allows us to stress a particular bond. Figure 3.2 shows an example of the plots produced.

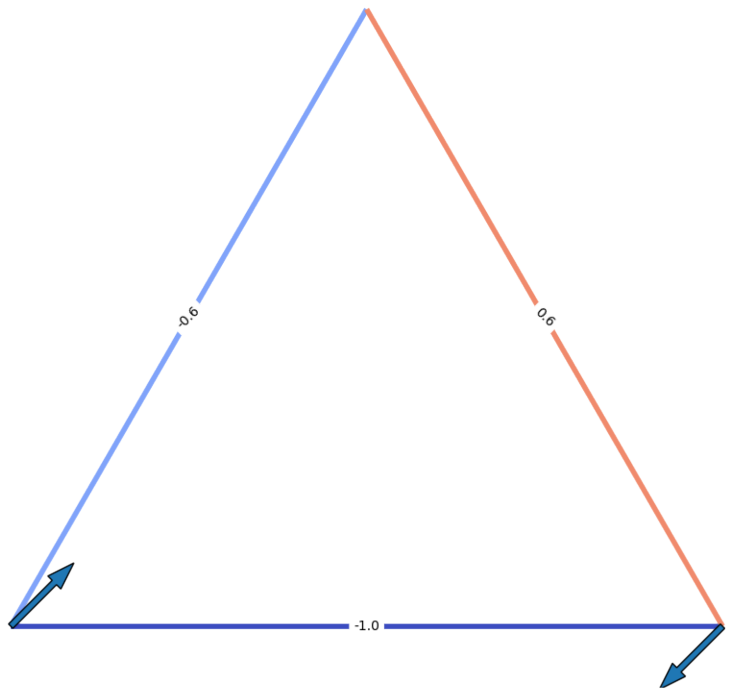

Figure 4: Stresses on a triangular framework. The arrows indicate the direction and relative magnitude of applied force. Negative numbers (blue) correspond to compression of the edge and positive numbers (red) to stretching.

3.3 Animation

To visualise the full non-linear response of the network to an applied strain, we minimise the configurational (potential) energy as described in the *Supporting Information* for Rocks et al. This minimisation is performed subject to the constraint that the strain on the source bond is a specified value, and we vary this value to build up frames of the animation.

However, there appears to be a typographical error in their Eq. S24. Because we are working with a network of springs, the potential energy is proportional to the square of the change in length of the spring. Their equation omits the square, and so should read:

$$E = \sum_{\langle i,j \rangle} k_{ij}(X_{ij} - l_{ij})^2$$

We minimise this corrected energy using a built-in scipy [6] algorithm. This re-

turns a set of node displacements that minimises the energy. We can use these to calculate the non-linear strain on the target bond and hence the non-linear strain ratio. We also use these to draw the deformation of the network under each given strain and compile them into an animation demonstrating the response of the network. An example of such an animation can be found at irmckenzie.co.uk/rigidity/HHHHHHHHHHHHHHHH(upload
an animation). The source bond is indicated in green, and the target in red.

4 Developing a tuning algorithm

4.1 Network generation

In generating their starting random networks, Rocks et al. used the contact graph of jammed packings of soft spheres. Such packings consist of overlapping soft spheres that have been brought to a local energy minimum. They chose this set-up because such networks have material properties that are well-understood. These networks starting at a local energy minimum also justifies taking the response of the network to applied tension to be the minimum energy configuration. HHHHHHHHHHHHHHHH(this feels shaky, which propoerties, and why does it justify?)

We instead make our networks using a three step process:

(i) We use a Poisson disc sampling algorithm [7] tto generate N node positions.

(ii) We add all the edges from a Delaunay triangulation of the node positions.

(iii) We prune edges until we have $2N$ remaining, which is just above $2N - 3$, the minimum number for a rigid graph.

First, long edges are removed, defined as those with lengths greater than twice the node spacing.

Then we remove edges at random, as long as the graph continues to satisfy two heuristics:HHHHHHHHHHHHHHHH(which paper?)

(a) For each node, we do not remove any edges if this would mean all its edges were on the same 'side' (i.e. the node must stay in the convex hull of its neighbours). Note that nodes on the periphery always have all their edges on the same side, so we remove no edges from them.

(b) No edge should be removed that would create a node of degree less than 3.

Our networks do not share the well-understood properties of their sphere packings, and we wanted to see if this would have any effect on the success of the algorithm.

4.2 General approach

The general approach taken to tuning a network for a given strain ratio is as follows:

(i) Remove each edge in turn from the network.

(ii) Compute the extensions, and thus the strains, resulting from removing that edge from the network.

(iii) Calculate the cost Δ^2 associated with these strains.

(iv) Remove the bond that resulted in the lowest cost.

(v) Repeat steps (i) - (iv) until the cost is below some threshold.

This is a greedy algorithm - at each step we look only one step ahead. Therefore we have no guarantees that it will find the best solution (i.e. fewest edge removals), or even that it will terminate at all. The remarkable result of the Rocks et al. paper is this simple approach did work reliably, and with very few edge removals.

The most expensive step, and consequently the one to which most attention is given, is step (ii). Here we have to solve Eq 13, which involves an inverse matrix. Matrix inversion is an expensive operation: it is of the order $\mathcal{O}(n^2 \log n)$, where n in our case is dN.

We are not quite working with an inverse matrix in practice, however. The formalism described in section 2.2.2 involves working with bonds of zero stiffness - so-called "ghost bonds". These allow us to examine the extension and strain between two nodes by having an edge in the graph that does not affect the mechanics of the situation.

This is implemented by expanding the square flexibility matrix, adding a row (and hence column) whose entries are all zero. Note though that $H = QF^{-1}Q^\top$ involves the inverse of F, which we have just made singular. We use instead the Moore-Penrose (MP) pseudoinverse $F^\dagger$, defined in section 2.2.1.

Now consider H. We have no guarantees that H is invertible, and in general it is not, so we must use the pseudoinverse, $H^\dagger$. This is justified as we are multiplying both sides of Eq. 11 by $H^\dagger$, and as $H\left|u\right\rangle$ is by definition in the image of H, we have $H^\dagger H \left|u\right\rangle = \left|u\right\rangle$.

4.3 Implementation

There are various ways to approach step (ii). The methods we use are outlined in the following sections.

4.3.1 Brute force

The most straightforward approach is to simply recalculate $H^\dagger$ after the trial removal of each edge. This comes with the benefit of being (relatively) easy to implement, and so was the first approach tried. It does, however, have the downside of a time complexity of $\mathcal{O}((dN)^2)$ per step, which gets expensive quickly as the size of the graph increases.

This can be reduced by noticing that $H^\top = (QF^\dagger Q^\top)^\top = (Q^\top)^\top (F^\dagger)^\top Q^\top = QF^\dagger Q^\top = H$. Therefore H is symmetric, and because we are working only with reals, equal to its own conjugate transpose, and hence Hermitian. This allows us to use a more efficient method for calculating the pseudoinverse $H^\dagger$.HHHHHHHHHHHHHHH(cite method?)

4.3.2 Sherman-Morrison updating

Rather than recalculating the entire $H^\dagger$ matrix for each edge removal, we can instead update it incrementally using a modified version of the Sherman-Morrison (SM) formula [8]. The SM formula provides a method for computing the inverse of the sum of an invertible matrix A and the outer product of two vectors, $\mathbf{u}\mathbf{v}^* = |u\rangle \langle v|$. We are only working with $\mathbb{R}$, so $\mathbf{v}^* = \mathbf{v}^\top$ Note that the outer product of two vectors is a matrix with rank 1, and that any rank 1 matrix can be written as an outer product of two vectors, so we can say that A has undergone a rank 1 update.

We can frame our problem in terms of a rank 1 update to H.
Suppose we have $H = QF^\dagger Q^\top$ for a framework $\mathcal{F} = (G = (V, E), \mathbf{p})$, and suppose we set the stiffness for a bond e_i to zero (effectively removing it, in our formalism). We wish to calculate the Hessian matrix $\widetilde{H}$ for this new system. Recall that $\langle i| F |j\rangle = \frac{\delta_{ij}}{k_i}$ and so the flexibility matrix for the new system is

$$\widetilde{F} = F - \frac{1}{k_i} |i\rangle \langle i| .$$

Taking the pseudoinverse of F and $\widetilde{F}$ gives

$$\widetilde{F}^\dagger = F^\dagger - k_i |i\rangle \langle i|$$

The equilibrium matrix Q is unchanged, so

$$\begin{aligned}
\widetilde{H} &= Q\widetilde{F}^\dagger Q^\top && (14)\\
&= Q(F^\dagger - k_i |i\rangle \langle i|)Q^\top && (15)\\
&= QF^\dagger Q^\top - k_i Q(|i\rangle \langle i|)Q^\top && (16)\\
&= H + (-k_i Q |i\rangle \langle i| Q^\top) && (17)
\end{aligned}$$

We can then take $\boldsymbol{u} = -k_i Q \left| i \right\rangle$ and $\boldsymbol{v}^\top = \left\langle i \right| Q^\top$. Note that $\boldsymbol{u}$ is a constant times the ith column of Q, and $\boldsymbol{v}^\top$ is the ith row of $Q^\top$, which is again the ith column of Q.

We cannot directly employ the SM formula as it is specifically for invertible matrices. Meyer [9] introduces six generalisations of the SM formula that deal with all possible cases for updating the pseudoinverse.

Meyer sets out three important criteria, which we express in terms of our current example: whether $\mathbf{u}$ is in the column-space of H, whether $\mathbf{v}^\top$ is in the column-space of $H^\top$ (which is the same as whether $\mathbf{v}$ is in the column-space of H), and whether $\beta = 1 + \mathbf{v}^\top H^\dagger \mathbf{u} = 0$.

We show now that $\mathbf{u}$ and $\mathbf{v}$ are in the column-space of H.HHHHHHHHHHHHHHH(how do I finish this?)

5 Tuning algorithm results

I achieved mixed success with these tuning algorithms. The update-based algorithm agrees very closely with the brute-force version, always removing the same edges in sequence and at each step calculating costs that differ with a percentage error of less than 0.1%. Both purported to do suspiciously well, often bringing the cost down to below the threshold by a large margin after removing only two steps, regardless of the network size. The tuned networks really did exhibit the desired linear strain ratio, but the full non-linear response was not as convincing. The networks never achieved the same magnitude of full response as they did in the linear regime, and sometimes the strain ratio even failed to exhibit the same sign.

There are some exceptions to this rule, however. After manually transcribing the graph from Fig. 1 of Rocks et al,

References

[1] Jason W. Rocks, Nidhi Pashine, Irmgard Bischofberger, Carl P. Goodrich, Andrea J. Liu, and Sidney R. Nagel. Designing allostery-inspired response in mechanical networks, Mar 2017.

[2] J. E. Graver. *Counting on Frameworks: Mathematics to Aid the Design of Rigid Structures.* The Mathematical Association of America, 2001.

[3] R. Penrose. A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51(3):406–413, 1955.

[4] A. Ben-Israel and T.N.E. Greville. *Generalized Inverses: Theory and Applications.* CMS Books in Mathematics. Springer New York, 2006.

[5] Audrey Lee and Ileana Streinu. Pebble game algorithms and sparse graphs. *Discrete Mathematics*, 308(8):1425–1437, April 2008.

[6] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[7] Robert Bridson. Fast poisson disk sampling in arbitrary dimensions. In *ACM SIGGRAPH 2007 Sketches*, SIGGRAPH '07, page 22–es, New York, NY, USA, 2007. Association for Computing Machinery.

[8] Jack Sherman and Winifred J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, 1950.

[9] Carl D. Meyer. Generalized inversion of modified matrices. *SIAM Journal on Applied Mathematics*, 24(3):315–323, 1973.