edmcr - Euclidean Distance Matrix Completion in R

Adam Rahman and R. Wayne Oldford
University of Waterloo

Abstract

Through motivating examples, we introduce the Euclidean distance matrix completion problem, and explore its use in various fields in the literature. We then introduce `edmc`, a function available in the `R` package `edmcr`, which solves the Euclidean distance matrix completion problem, and the related sensor network localization problem, using a variety of algorithms.

Working through examples using both real and simulated data, we show how each of the algorithms implemented in `edmc` can be used to effectively complete a Euclidean distance matrix, the pros and cons of each algorithm, and their use in various types of completion problems. We end with a brief introduction to the theoretical background of the algorithms, outlining some of the more technical details implemented in each.

Keywords: *R, matrix completion problems, point configurations, high dimensional data analysis, dimension reduction, guided random search*

1 Introduction

Consider a set of n points in arbitrary dimension p, $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^p$. Let $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_n]^T$ denote the matrix such that $\mathbf{X}[i,] = \mathbf{x}_i^T$. Without loss of generality, assume the $\mathbf{x}_i$ are centered so that $\sum_{i=1}^n \mathbf{x}_i = 0$. The *Euclidean distance* between $\mathbf{x}_i$ and $\mathbf{x}_j$ is denoted

$$d_{ij} = ||\mathbf{x}_i - \mathbf{x}_j|| \tag{1}$$

and let $\mathbf{D} = [d_{ij}]$ be the *Euclidean distance matrix*. Next, let $\mathbf{D}^* = [d_{ij}^2]$ be the *square* Euclidean distance matrix, which can be used to derive the *Gram matrix*, $\mathbf{G} = \mathbf{X}\mathbf{X}^T$. Note that the Gram matrix can be written as

$$\begin{aligned} \mathbf{G} &= \mathbf{X}\mathbf{X}^T \\ &= \mathbf{X}\mathbf{O}\mathbf{O}^T\mathbf{X}^T \end{aligned}$$

for any orthogonal matrix $\mathbf{O}$. The Gram matrix $\mathbf{G}$ provides a point configuration from its eigendecomposition that is unique up to an orthogonal rotation.

In practice, situations may arise where only some of the d_{ij}s are unknown - for instance, in the case of sensor network localization (see for example Alfakih et al. [2011] and Ding et al. [2010]) where distances between sensors are only known if they are within a distance R, known as the radio range, of one another. This would result in what is known as a *partial distance matrix*. Another such example can be found in molecular biology, and more specifically the problem of molecular conformation (see for example Hendrickson [1995]), where given a set of unknown distances and other restrictions, the goal is to determine the underlying configuration of a (protein) molecule.

Given a partial distance matrix, the goal is *completing* it, a problem known as the Euclidean distance matrix completion problem (edmcp). There are many methods for solving the edmcp, see for example Alfakih et al. [1999], Fang and O'Leary [2012], & Trosset [2000] for an introduction to the problem area and an in depth look into the methods we will consider in later sections. None of these tools, however, are available in the programming language `R` (R Core Team [2013]).

To bridge this gap, we introduce the edmcr package, publicly available on the Comprehensive `R` Archive Network (CRAN). We introduce several useful algorithms included in edmcr which allow the user to complete a partial distance matrix, and solve related problems. Each of these algorithms have their own advantages and disadvantages, which we will explore.

This paper is structured as follows. In Section 2, a number motivating examples are introduced that are closely related to the Euclidean distance matrix completion problem. These examples include problems in distance-based dimension reduction, sensor tracking, and molecular conformation.

In Section 3, the function `edmc`, the main function for Euclidean distance matrix completion in edmcr, is introduced. The algorithms which can be called from `edmc` include Alfakih et al. [1999], Fang and O'Leary [2012], Trosset [2000], Krislock and Wolkowicz [2010], and Rahman and Oldford [2016] will also be introduced. The function `sprosr`, used for solving the problem of molecular conformation, is also introduced. `sprosr` is an implementation of the *semidefinite programming-based protein structure determination algorithm* of Ramandi [2011].

In Section 4, a number of examples are introduced using both real and simulated data to demonstrate the syntax and functionality of the algorithms in edmcr. In Section 5, the mathematical details of the Euclidean distance matrix completion aglorithms are presented, providing the background necessary to better understand the more advanced functionality available in the algorithms. Section 6 offers some closing remarks.

2 Motivating examples

Three related, yet distinctly different problems illustrate the Euclidean distance matrix completion problem. Their ordering ranges from cases with little inherent structure to cases involving significant structure in the underlying configuration.

2.1 Distance-based dimension reduction

We begin with the simplest example of distance matrix completion by considering the link between dimension reduction techniques and Euclidean distance matrix completion. In many cases, dimension reduction is done by first considering a partial distance matrix (such as one created by a neighbourhood graph), typically resulting in a sparse matrix containing a small number of known distances. For instance, the Isomap algorithm of Tenenbaum et al. [2000] first creates a partial distance matrix using the distances in a *k-nearest neighbour graph*, requiring only that the resulting graph be connected. The remaining distances are then filled using the shortest paths between nodes along the graph. A point configuration is then constructed using multidimensional scaling. Other examples of distance-based dimension reduction algorithms include Laplacian Eigenmaps (Belkin and Niyogi [2001]) and Locally Linear Embedding (Roweis and Saul [2000]).

Consider instead the following approach. Given a high dimensional point configuration, and the corresponding k-nearest neighbour graph, suppose we treated these as *known* distances in a Euclidean distance matrix. Then, instead of using the shortest path distances to fill in the unknown distances, we instead make use of a Euclidean distance matrix completion algorithm to complete the distance matrix, and embed it in a lower dimension. The lower dimensional point configuration could then be extracted via simple eigendecomposition.

A similar idea would be to use the *minimum spanning tree* of the high dimensional point configuration instead of the k-nearest neighbour graph. The minimum spanning tree is a graph in which every node is visited once, with no cycles, and minimum edge weight of any such graph, and is known to carry important clustering information (Gower and Ross [1969], Friedman and Rafsky [1981], Friedman and Rafsky [1979]). We explore this idea using the statue manifold data set used in Tenenbaum et al. [2000].

In the case of either a k-nearest neighbour graph or a minimum spanning tree graph, we end up with partial distance matrix $\mathbf{D}$ that is relatively sparse. This allows for alternative methods (such as that in Rahman and Oldford [2016]) to be used to complete the matrix.

2.2 Sensor network localization

In sensor network localization, the goal is to determine the positions of a set of *sensors*. Unlike typical edmc problems however, additional information in the form of a set of *anchors* with known positions are available for reference. Distances between sensors, as well as between sensors and anchors, are known only if they are within some (Euclidean) distance of one another, known as the *radio range*, denoted by R.

An example of such a problem is described in Rangarajan et al. [2008], where access points (i.e. anchors) use received signal strength from wireless users (emitters, i.e. sensors) to attempt to locate the positions of access points with *unknown* locations. Figure 1 illustrates the problem.

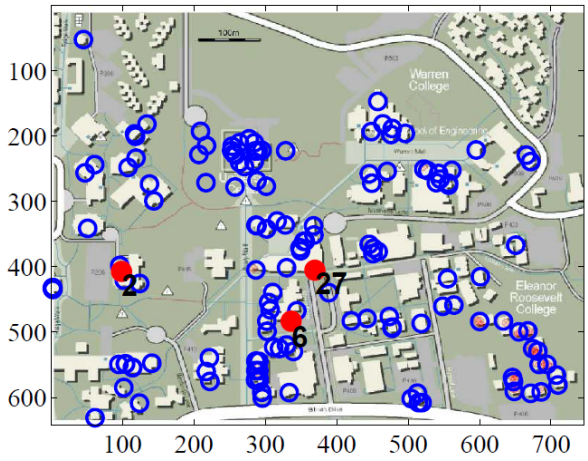


Figure 1: *A map of the 200 known access point locations (in blue). Also shown are access points sampled by a single user (in red) at a certain moment in time, along with the strength of the received signal strength in black. Rangarajan et al. [2008]*

To model this problem as a sensor network localization problem, the received signal strength is first converted to a distance. For emitter i and access point j, let s_{ij} represent the received signal strength at the access point. Then,

$$d_{ij} = d_0 * 10^{(s_0 - s_{ij})/(10n_p)}$$

where s_0 is the received signal strength at the access point at a reference distance of d_0, and σ_0^2 is the standard deviation of received signal strength. n_p is known as the "path-loss exponent", and is environment dependent. It can be viewed as a noise term, as it is generally unknown.

With these individual distances, the distance matrix $\mathbf{D}$ has the following form

$$\mathbf{D} = \begin{bmatrix} \mathbf{D}_{kk} & \mathbf{D}_{ke} & \mathbf{D}_{ku} \\ \mathbf{D}_{ek} & \mathbf{D}_{ee} & \mathbf{D}_{eu} \\ \mathbf{D}_{uk} & \mathbf{D}_{ue} & \mathbf{D}_{uu} \end{bmatrix}$$

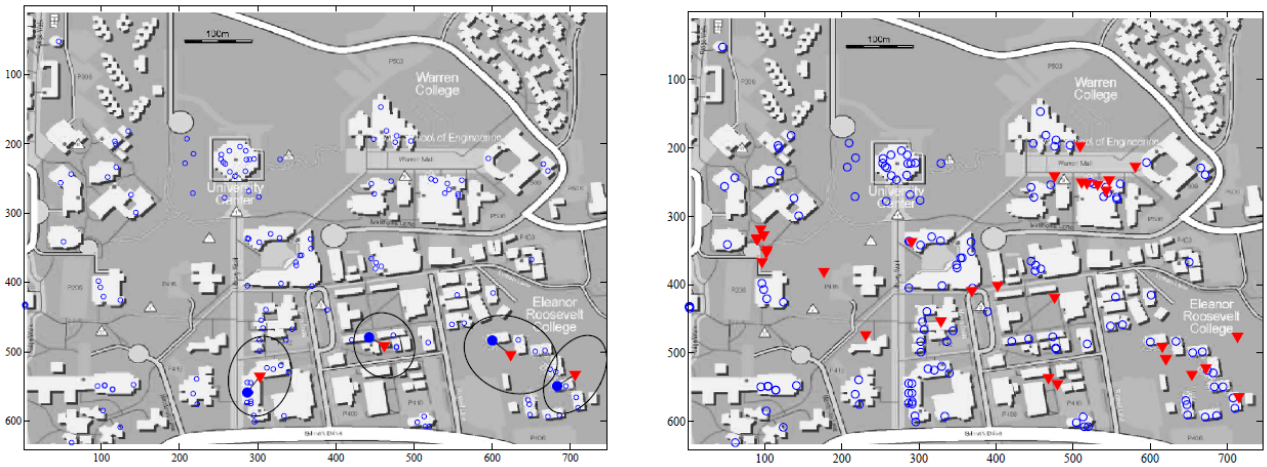
where subscript k denotes the known access points, u the unknown access points, and e the emitters. $\mathbf{D}_{kk}$ represents the distance matrix containing the distances between the known access points, $\mathbf{D}_{ke}$ the distance matrix between the known access points and the emitters, etc.

Since each emitter will only be within range of a subset of the known and unknown access points, both $\mathbf{D}_{ek}$ and $\mathbf{D}_{uk}$ will be a partial distance matrix (denoted with a superscript $\star$). Also, since there are access points with unknown locations, the distance matrices $\mathbf{D}_{uu}$ and $\mathbf{D}_{uk}$ are both completely unknown. The result is the following partial distance matrix $\mathbf{D}^\star$ to be completed, with the end goal being to reconstruct the complete matrix $\mathbf{D}$ from $\mathbf{D}^\star$.

$$\mathbf{D}^\star = \begin{bmatrix} \mathbf{D}_{kk} & \mathbf{D}^\star_{ke} & \star \\ \mathbf{D}^\star_{ek} & \star & \mathbf{D}^\star_{eu} \\ \star & \mathbf{D}^\star_{ue} & \star \end{bmatrix}$$

As a practical application, Rangarajan et al. [2008] consider data collected at the University of California, San Diego (McNett and Voelker [2005]). The data represents received signal strength data for personal data assistants (PDAs) given to 275 freshman students. Over the course of the experiment, 300 unique access points were sensed, however only 200 had known locations.

The goal of the analysis was to locate the approximately 100 access points with unknown locations. Figure 2(a) demonstrates the accuracy of the algorithm by first finding the position of three access points whose positions were hidden from the algorithm. Figure 2(b) shows the 200 known access points in blue, as well as the estimated positions of the 100 unknown access points in red.



(a) (b)

Figure 2: *(a): The proposed positions (in red) of three access points (of known location) whose locations were hidden from the algorithm. Actual locations are show in blue. (b) The proposed positions of the 100 unknown access points (in red), and the positions of the 200 known access points (in blue). Rangarajan et al. [2008]*

Figure 3 illustrates the general problem. Here, blue circles represent access points with known positions, red triangles access points with unknown positions, and grey squares emitters with unknown

positions. Solid lines represent known distances, while dotted lines represent unknown distances, forming a partial distance matrix. The goal of the analysis is to find the positions of the emitters and access points with unknown locations.

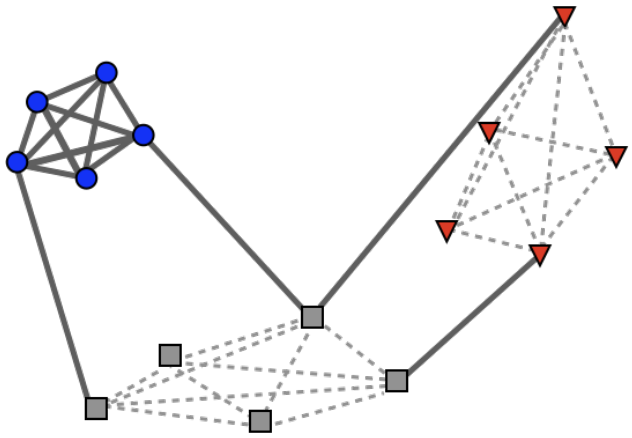


Figure 3: *The general problem addressed by Rangarajan et al. [2008].*

In addition to geo-tracking problems such as this, sensor networks have a large range of applications, such as military/security applications (Lawlor [2005]), environmental habitat monitoring (Mainwaring et al. [2002]), and wildlife monitoring (Szewczyk et al. [2004]), amongst others. There are many proposed methods for solving these problems, ranging from relaxations of semi-definite programming algorithms to iterative multi-dimensional scaling (see for example Krislock and Wolkowicz [2011], Ding et al. [2010], Rangarajan et al. [2008]).

2.3 Molecular conformation

The molecule problem looks to reconstruct a three-dimensional molecule (most often a protein structure) using the incomplete interatomic distances measured via nuclear magnetic resonance (nmr) spectroscopy. While some interatomic distances may be unknown, there is a vast amount of structure that exists in a molecule that can be exploited, as nature gives insights into the angles and distances at which certain atoms must bond. Having *a priori* knowledge of the atomic structure of a molecule will allow for the determination of bounds on the possible distances that can theoretically exist between atoms. Other phenomena such as natural repulsion/attraction forces, and known atomic structure enforce natural bounds on the possible distances that can exist between pairs of atoms.

Given all these restrictions, the problem is to then (as accurately as possible) propose a series of plausible configurations for the molecule - as a set of bounds rarely can completely specify a distance matrix, there are likely many valid solutions to a single conformation problem. There is a large volume of research done in this area, including Havel and Wüthrich [1985], Wüthrich [1990], Glunt et al. [1993], Hendrickson [1995], Trosset [1998], and Moré and Wu [1999].

One of the first attempts to solve this problem was presented in Havel and Wüthrich [1985]. Using nuclear magnetic resonance imaging, they extracted intramolecular, interproton distances from the molecular backbone of various protein structures. As they had the exact measurements (in Ångströms, $\mathring{A}$), the data were modified such that each known distance was placed in an appropriate range (i.e. distances shorter than $2\mathring{A}$ were instead treated as an unknown distance in the range $[2,2.5)\mathring{A}$). In total, 10 data sets were considered, each with varying levels of restrictions on

Figure 4: *A stereographic image of three attempts to replicate the crystal structure of a BPTI protein. The actual crystal structure is superimposed in darker black. This example shows a protein reconstruction with the least known structure of all considered in Havel and Wüthrich [1985].*

bonding angles and distances placed on them. Figures 4 and 5 illustrate the results from two of the experiments.

Figure 5: *A stereographic image of three attempts to replicate the crystal structure of a BPTI protein. The actual crystal structure is superimposed in darker black. This example shows a protein reconstruction with the most known structure of all considered in Havel and Wüthrich [1985].*

3 Solving edmc problems with R

Currently, there is no way to solve Euclidean distance matrix completion problems using R. To fill this void, we introduce edmcr, an R package containing many algorithms to perform Euclidean distance matrix completion, and its various applications, including molecular reconstruction and sensor network localization.

3.1 edmc

To streamline the implementation of several functions which can be used to complete Euclidean distance matrices, we introduce `edmc`, the main function through which all of the completion algorithms can be accessed. There are currently five different methods implemented in `edmc`:

1. Semidefinite Programming (Alfakih et al. [1999])
2. Non-convex Position Formulation (Fang and O'Leary [2012])
3. Dissimilarity Parameterization Formulation (Trosset [2000])
4. Guided Random Search (Rahman and Oldford [2016])
5. Sensor Network Localization (Krislock and Wolkowicz [2010])

The `R` implementation of each is discussed in the remainder of this section, and the mathematical details and formulation for each can be found in Section 5.

The `edmc` function takes the following input variables

`D`	An $n \times n$ Euclidean distance matrix to be completed, with unknown entries set to NA
`method`	The completion algorithm to be used. One of "sdp", "npf", "dpf", "snl", "grs"
`...`	Additional input variables specific to the `method` used

Of practical importance in the implementation of `edmc` is to note that only two inputs are specified - a partial Euclidean distance matrix `D`, and the desired completion algorithm `method`. All other input variables are specific to the algorithm specified in `method`, and are not specified in the `edmc` input list. The user should specify the required input variables (outlined for each `method` below) by naming them directly during the call to `edmc`.

The output of `edmc` is also specific to the method used. The one output common to all is the completed distance matrix `D`. The specific output of each `method` will be discussed below.

3.1.1 method = "sdp"

The Semidefinite Programming algorithm of Alfakih et al. [1999], denoted *sdp*, and coded in the subroutine `sdp`, requires the following input variables to be specified in `edmc`

`D`	An $n \times n$ Euclidean distance matrix to be completed, with unknown entries set to NA
`A`	A weight matrix, with $a_{ij} = 0$ implying d_{ij} is unknown. Generally, if d_{ij} is known, $a_{ij} = 1$ although any non-negative weight is allowed
`toler`	convergence tolerance for the algorithm. Default is set to $1e-8$.

resulting in the following list of output variables

`D`	The completed $n \times n$ Euclidean distance matrix
`optval`	The minimum value of the target function achieved during minimization

The `sdp` algorithm is appropriate for smaller completion problems, as it is computationally quite expensive ($\mathcal{O}(n^2)$). However, the convexity of the search space generally results in a global solution.

3.1.2 method = "npf"

The non-convex position formulation algorithm of Fang and O'Leary [2012], denoted *npf*, and coded in the subroutine `npf`, requires the following input variables to be specified in `edmc`

`D`	An $n \times n$ Euclidean distance matrix to be completed, with unknown entries set to NA.
`A`	A weight matrix, with $a_{ij} = 0$ implying d_{ij} is unknown. Generally, if d_{ij} is known, $a_{ij} = 1$, although any non-negative weight is allowed
`d`	The dimension of the resulting completion
`dmax`	The maximum dimension to consider during dimension relaxation
`decreaseDim`	During dimension reduction, the number of dimensions to decrease at each step
`stretch`	The stretching factor applied to the distance matrix (see Section 5). The default is set to 1 (i.e. no stretching is applied).
`dimMethod`	One of "Linear" or "NLP" corresponding to the desired method of dimension reduction (see Section 5).
`toler`	convergence tolerance for the algorithm. Default is set to $1e-8$.

Many of these input variables have default values (illustrated in the examples provided in Section 4) that are sufficient for most minimization problems. As such, these input values need not be specified during the call to `edmc` - only `D`, `A`, and `d` are required. The advanced user may be interested in changing these defaults in the event that the algorithm fails to converge on a global solution. Section 5 outlines the technical background required to understand how these optional input variables can be used.

In the case of utilizing the `npf` subroutine, `edmc` produces the following list of outputs:

`D`	The completed $n \times n$ Euclidean distance matrix
`optval`	The minimum value of the target function achieved during minimization

The `npf` algorithm is appropriate for any completion problem, as it is substantially faster than `sdp`. This increase in speed unfortunately comes at the cost of convexity, so care must be taken to ensure that algorithm arrives on a global solution, and not a local solution.

3.1.3 method = "dpf"

The dissimilarity parameterization formulation algorithm of Trosset [2000], denoted *dpf*, and coded in the subroutine `dpf`, requires the following input variables to be specified in `edmc`

`D`	An $n \times n$ Euclidean distance matrix to be completed, with unknown entries set to NA
`d`	The dimension for the resulting completion
`lower`	An $n \times n$ matrix containing the lower bounds for the unknown entries in `D`. If NULL (default), `lower` is set to be a matrix of 0s
`upper`	An $n \times n$ matrix containing the upper bounds of the unknown entries in `D`. If NULL, `upper`[i,j] is set to be the shortest path between node i and node j.
`retainMST`	A logical input indicating if the current minimum spanning tree structure in `D` should be retained. If `TRUE`, an mst-preserving lower bound is calculated.

For `dpf`, both `D` and `d` must be specified by the user. The remaining inputs are optional and have

set defaults. Their use will be described via example in Section 4. The `dpf` subroutine produces the following list of outputs

`D`	An $n \times n$ Euclidean distance matrix with embedding dimension d
`optval`	The minimum function value achieved during minimization

The default values of both `lower` and `upper` are set to NULL, internally setting the lower bounds for all unknown values to 0, and setting the upper bounds using the triangle inequality. However, there may be situations where tighter bounds may be imposed. For example, in the case of molecular conformation, physical properties, such as the angle of the formed bond between elements, may force the unknown distance to lie within some boundary, $d_L \leq d_{ij} \leq d_U$. In this case, `upper` can be used to specify these tighter bounds.

The minimum spanning tree is known to contain important information about a point configuration, especially clustering information (Gower and Ross [1969]). The minimum spanning tree of a partial distance matrix can be preserved by imposing an appropriate lower bound on the unknown distances. This can be done simply using the `retainMST` variable in `edmc`; setting it to `TRUE` will preserve the minimum spanning tree in the partial distance matrix by calculating the appropriate lower bound in `lower` for each unknown distance.

3.1.4 method = "grs"

When the `method` variable is set to `"grs"`, `edmc` executes the guided random search algorithm of Rahman and Oldford [2016] by calling the subroutine `grs`. The `grs` algorithm is used to solve Euclidean distance matrix completion problems where only the minimum spanning tree is known.

When `grs` is the specified method, two input variables are required

`D`	An $n \times n$ partial Euclidean distance matrix to be completed containing only the distances in the minimum spanning tree
`d`	The desired embedding dimension

Completion of the matrix $\mathbf{D}$ results in two output variables

`P`	An $n \times d$ matrix containing the positions of the n nodes in d dimensions.
`D`	An $n \times n$ Euclidean distance matrix with embedding dimension `d`

In Section 4, we will address the ability of this algorithm to create point configurations in a lower embedding dimension than the original partial Euclidean distance matrix.

3.1.5 method = "snl"

The sensor network localization algorithm of Krislock and Wolkowicz [2010], denoted *snl*, and coded in the subroutine `snl`, requires the following input variables to be specified in `edmc`

`D`	The partial distance matrix specifying the known distances between nodes. If `anchors` is specified (and is a $p \times d$ matrix), the p final columns and p final rows of `D` specify the distances between the anchors specified in `anchors`.
`d`	The dimension for the resulting completion
`anchors`	A $p \times d$ matrix specifying the d dimensional locations of the p anchors. If the anchorless problem (i.e. a general edmcp) is to be solved, `anchors` = NULL

The `snl` algorithm requires that both `D` and `d` be specified as input variables. Unlike the other algorithms the main purpose of the `snl` algorithm is to solve the sensor network localization problem, however, it can also solve the Euclidean distance matrix completion problem by specifying the `anchors` input as `NULL`. As such, it takes an additional argument `anchors`, which specifies `d`-dimensional positions of the anchors. The main target is to then locate (in `d`-dimensional space) as many of the sensors as possible, resulting in the following output

`X` the d-dimensional positions of the localized sensors. Note that it may be the case that not all sensors could be localized, in which case `X` contains the positions of only the localized sensors.

The $m \times d$ matrix `X` contains the positions of the localized sensors. It is not necessarily the case that all sensors can be localized, in which case they are not included in `X`, and therefore we have $m \leq n$, where n is the total number of sensors.

3.2 sprosr

Determining the structure of a molecule requires a specialized algorithm that can handle the additional angle constraints often found in molecular conformation problems. In edmcr, the protein problem can be solved using the `sprosr` function, an `R` implementation of the *semidefinite programming-based protein structure determination algorithm* of Ramandi [2011].

`sprosr` has three required and four optional input variables

`seq`	A table containing the amino acid sequence of the protein in CYANA .seq format
`aco`	A table containing the angle constraint information in CYANA .aco format
`upl`	A table containing the distance constraint information in CYANA .upl format
`hydrogen_omission`	Should side-chain hydrogen atoms be omitted? TRUE/FALSE. Default is FALSE
`in_min_res`	User overwrite of the minimum residue number.
`in_max_res`	User overwrite of the maximum residue number.

Each of `seq`, `aco`, and `upl` require a table following the form set out by CYANA (Güntert [2004]). For `seq`, the sequence file (.seq) is a table of two columns - column one is the abbreviated name of the amino acid, and column two is the corresponding residue number. Table 3.2 provides an example.

MET	1
GLN	2
ILE	3
PHE	4
VAL	5

Table 1: *The first 5 rows of a sequence (seq) file.*

The `aco` input table contains angular constraints on the phi and psi angles for the amino acid residues, and follows the format of the .aco CYANA file. It contains five columns - the residue

number of the amino acid corresponding to the `seq` input variable, the name of the amino acid residue, the name of the angle being constrained (either PHI or PSI), the lower limit of the angle, and the upper limit of the angle, in degrees. Table 3.2 provides an example.

2	GLN	PHI	-95.0	-85.0
3	ILE	PHI	-149.0	-139.0
4	PHE	PHI	-129.0	-119.0
6	LYS	PHI	-117.0	-107.0
7	THR	PHI	-110.0	-100.0

Table 2: *The first 5 rows of an angle constraint (aco) file.*

Finally, the `upl` input table contains the distance constraints (in Ångströms) between amino acid residues. It contains seven columns - the residue number, amino acid name, and atom name of the first (columns 1-3) and second (columns 4-6) residues, and the constrained distance between them. Table 3.2 provides an example.

1	MET	HA	2	GLN	H	2.93
1	MET	HB2	2	GLN	H	3.86
1	MET	HB2	63	LYS	HA	5.50
1	MET	HB3	2	GLN	H	3.86
1	MET	HB3	63	LYS	HA	5.50

Table 3: *The first 5 rows of an distant constraint (upl) file.*

`sprosr` provides two output variables upon completion

`X`	The three dimensional point configuration of the completed protein molecule
`report`	A list detailing the total violations in the protein molecule

The `X` output is straight forward, it contains the three dimensional positions of the amino acid residues in the completed protein molecule. The `report` output is a list detailing the total violations in the protein, including the number (and total size) of the violations in the equality constraints (`$eq_err`), the number of violations in both the lower (`$lo_err`) and upper bounds (`$up_err`), the number of chiral violations (`$chiral`), the number of angle violations in both phi (`$phi`) and psi (`$psi`), and the number of dihedral (`$dihed`) and hydrogen bonds (`$hbond`) that violate their upper bounds.

3.3 getConfig

Each of the completion algorithms above return a Euclidean distance matrix, where in many applications a point configuration is ultimately desired. Given a Euclidean distance matrix $\mathbf{D}$, we seek the corresponding point configuration matrix $\mathbf{X}$ such that $d_{ij} = ||x_i - x_j||$.

In Section 1, the squared Euclidean distance matrix $\mathbf{D}^*$ and the *Gram* matrix were defined. To obtain a Gram matrix $\mathbf{G}$ from a squared Euclidean distance matrix $\mathbf{D}^*$, first note that d^*_{ij} can be written as:

$$\begin{aligned} d^*_{ij} &= ||\mathbf{x}_i - \mathbf{x}_j||^2 \\ &= \mathbf{x}_i^T\mathbf{x}_i + \mathbf{x}_j^T\mathbf{x}_j - 2\mathbf{x}_i^T\mathbf{x}_j \\ &= g_{ii} + g_{jj} - 2g_{ij} \end{aligned}$$

Thus, given a Gram matrix $\mathbf{G}$, the corresponding squared Euclidean distance matrix is

$$\mathbf{D}^* = K(\mathbf{G}) = \mathbf{1}\mathbf{g}^T + \mathbf{g}\mathbf{1}^T - 2\mathbf{G} \tag{2}$$

where $\mathbf{1}$ is the column matrix of ones, and $\mathbf{g}$ is the column vector containing the diagonal elements of $\mathbf{G}$, i.e. $\mathbf{g} = [g_{11}, g_{22}, ..., g_{nn}]^T$. This function has been implemented in `edmcr` as `gram2edm`, and takes a Gram matrix and returns the corresponding squared Euclidean distance matrix.

Inverting this equation, we arrive at a function which takes a squared Euclidean distance matrix $\mathbf{D}^*$ and returns the corresponding Gram matrix $\mathbf{G}$

$$\mathbf{G} = \mathcal{T}(\mathbf{D}^*) = -\frac{1}{2}\mathbf{P}\mathbf{D}^*\mathbf{P} \tag{3}$$

where $\mathbf{P} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T$ is an orthogonal projection matrix. This function is available in `edmcr` as the function `edm2gram`.

From the Gram matrix, a simple eigendecomposition can be used to obtain a point configuration matrix $\mathbf{X}$. Recall that this point configuration is unique only up to an orthogonal rotation. Note also that a $p \leq n$ dimensional point configuration $\mathbf{X}_p$ is obtained by keeping only the p largest eigenvalues, and setting all others to 0.

This entire process has been implemented in `edmcr` in the function `getConfig`, which takes the input variables

`D`	An $n \times n$ Euclidean distance matrix
`d`	The desired embedding dimension

and returns

`X`	An $n \times d$ point configuration matrix
`Accuracy`	The accuracy of the eigendecomposition, measured as $\frac{\sum_{i=1}^{d}\lambda_i}{\sum_{i=1}^{n}\lambda_i}$

4 Examples

In this section, we show some simple examples to demonstrate the syntax of each of the methods described in Section 3.

4.1 A simple Euclidean distance matrix completion problem

Consider the following partial Euclidean distance matrix (from Trosset [2000])

$$\begin{bmatrix} 0 & 3 & 4 & 3 & 4 & 3 \\ 3 & 0 & 1 & ? & 5 & ? \\ 4 & 1 & 0 & 5 & ? & 5 \\ 3 & ? & 5 & 0 & 1 & ? \\ 4 & 5 & ? & 1 & 0 & 5 \\ 3 & ? & 5 & ? & 5 & 0 \end{bmatrix}$$

with the following known completion in three dimensional space:

$$\begin{bmatrix} 0 & 3 & 4 & 3 & 4 & 3 \\ 3 & 0 & 1 & \sqrt{18} & 5 & \sqrt{18} \\ 4 & 1 & 0 & 5 & \sqrt{32} & 5 \\ 3 & \sqrt{18} & 5 & 0 & 1 & \sqrt{18} \\ 4 & 5 & \sqrt{32} & 1 & 0 & 5 \\ 3 & \sqrt{18} & 5 & \sqrt{18} & 5 & 0 \end{bmatrix}$$

For comparative purposes, note that $\sqrt{18} \approx 4.243$ and $\sqrt{32} \approx 5.657$.

To solve the problem using edmcr, first define the Euclidean distance matrix in R as follows:

```
R> #Define the partial distance matrix
R> D <- matrix(c(0,3,4,3,4,3,
                 3,0,1,NA,5,NA,
                 4,1,0,5,NA,5,
                 3,NA,5,0,1,NA,
                 4,5,NA,1,0,5,
                 3,NA,5,NA,5,0), byrow=TRUE, nrow=6)
```

For solving standard Euclidean distance matrix completion problems such as this, the sdp, npf, and dpf algorithms should be considered. Beginning with the sdp algorithm

```
R> #Define the adjacency matrix A
R> A <- matrix(c(1,1,1,1,1,1,
                 1,1,1,0,1,0,
                 1,1,1,1,0,1,
                 1,0,1,1,1,0,
                 1,1,0,1,1,1,
                 1,0,1,0,1,1), byrow=TRUE, nrow=6)

R> edmc(D=D, method = "sdp", A=A, toler=1e-8)


$D
         [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
[1,] 0.000000 3.000020 3.999993 3.000020 3.999993 3.000011
[2,] 3.000020 0.000000 1.000051 4.321144 5.000003 4.453067
[3,] 3.999993 1.000051 0.000000 5.000003 5.656642 5.000002
[4,] 3.000020 4.321144 5.000003 0.000000 1.000051 4.453067
[5,] 3.999993 5.000003 5.656642 1.000051 0.000000 5.000002
```

```
[6,] 3.000011 4.453067 5.000002 4.453067 5.000002 0.000000

$optval
[1] 5.406998e-10
```

Note that while a solution to the problem was found (since the optimal function value is near zero), this matrix differs slightly from the known completion. The main reason for this difference is the rank of the completed distance matrix. The corresponding Gram matrix has five non-zero eigenvalues, meaning the completed squared distance matrix has rank five, where the original solution had an embedding dimension of three. The inability to specify an embedding dimension is one of the major drawbacks of the `sdp` algorithm, a problem that is remedied by the other algorithms we consider, at the cost of convexity.

Next, consider the solution given by the `npf` algorithm

```
R> set.seed(48)

R> #Define the adjacency matrix A
R> A <- matrix(c(1,1,1,1,1,1,
                 1,1,1,0,1,0,
                 1,1,1,1,0,1,
                 1,0,1,1,1,0,
                 1,1,0,1,1,1,
                 1,0,1,0,1,1), byrow=TRUE, nrow=6)

R> #Default settings for some inputs (not required as input to edmc())
R> dmax = n - 1
R> decreaseDim = 1
R> stretch = 1
R> method = "Linear"
R> toler = 1e-8

R> edmc(D=D, method="npf", A=A, d=3)


$D
     [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
[1,]    0 3.000000 4.000000 3.000000 4.000000 3.000000
[2,]    3 0.000000 1.000001 4.242615 5.000000 4.241516
[3,]    4 1.000001 0.000000 5.000000 5.656888 5.000000
[4,]    3 4.242615 5.000000 0.000000 1.000001 4.241561
[5,]    4 5.000000 5.656888 1.000001 0.000000 5.000000
[6,]    3 4.241516 5.000000 4.241561 5.000000 0.000000

$optval
[1] 3.894878e-11
```

The resulting solution, unlike in the `sdp` subroutine, converges very nearly to the known solution. This is due to the ability to specify the embedding dimension. This, however, comes with its own

set of difficulties. Suppose the embedding dimension of the example matrix is unknown, and is instead specified as `d = 2`, resulting in the following completion

```
R> set.seed(48)
R> edmc(D=D, method="npf", A=A, d=2)


$D
          [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
[1,] 0.000000 2.957321 3.711975 2.957320 3.711976 2.104172
[2,] 2.957321 0.000000 1.423176 3.660281 5.036896 4.790540
[3,] 3.711975 1.423176 0.000000 5.036896 6.377897 5.118735
[4,] 2.957320 3.660281 5.036896 0.000000 1.423176 4.790539
[5,] 3.711976 5.036896 6.377897 1.423176 0.000000 5.118735
[6,] 2.104172 4.790540 5.118735 4.790539 5.118735 0.000000

$optval
[1] 72.33753
```

Clearly, the algorithm is forced to settle on a local solution, reaching an optimal value of 72.34 - recall a solution exists only if a global minimum of zero is achieved. If it is believed a solution actually exists in two dimensions, it may be worthwhile to attempt stretching, or possibly increasing the number of dimensions considered during relaxation. However, a more likely solution is to change the completion dimension.

Finally, consider the solution provided using the `dpf` algorithm

```
R> set.seed(98)

R> edmc(D=D, method="dpf", d=3)


$D
     [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
[1,]    0 3.000000 4.000000 3.000000 4.000000 3.000000
[2,]    3 0.000000 1.000000 4.240979 5.000000 4.245210
[3,]    4 1.000000 0.000000 5.000000 5.659058 5.000000
[4,]    3 4.240979 5.000000 0.000000 1.000000 4.248187
[5,]    4 5.000000 5.659058 1.000000 0.000000 5.000000
[6,]    3 4.245210 5.000000 4.248187 5.000000 0.000000

$optval
[1] 1.878904e-09
```

Again, with the ability to specify a an embedding dimension, the `dpf` algorithm is able to converge very nearly to the global minimum. However, similar to the `npf` algorithm, consider what happens if the embedding dimension is unknown, and attempting to complete the matrix with an embedding dimension of $d = 2$:

```
R> set.seed(98)

R> edmc(D=D, method="dpf", d=2)


$D
     [,1]     [,2] [,3]     [,4] [,5]     [,6]
[1,]    0 3.000000    4 3.000000    4 3.000000
[2,]    3 0.000000    1 3.967597    5 4.349922
[3,]    4 1.000000    0 5.000000    6 5.000000
[4,]    3 3.967597    5 0.000000    1 4.349923
[5,]    4 5.000000    6 1.000000    0 5.000000
[6,]    3 4.349922    5 4.349923    5 0.000000

$optval
[1] 8.456663
```

With an achieved minimum value of 8.46, the algorithm has again failed to converge near the required global minimum of 0, indicating that the proposed solution is not a distance matrix.

4.2 Analyzing the Anderson/Fisher Iris Data

Consider completing the famous Iris data set, originally collected in Anderson [1935] and analyzed in Fisher [1936]. To investigate the relative performance of the Euclidean distance matrix completion methods considered so far, take $\mathbf{D}$ to be constructed from the distances between flowers in the Iris data. Each completion method produces $\widehat{\mathbf{D}}$ for the case of knowing only the distances in the minimum spanning tree.

4.2.1 Completion accuracy

To measure the accuracy of each completion, the relative difference in dissimilarities

$$RDD = \frac{||\mathbf{D} - \widehat{\mathbf{D}}||_F^2}{||\mathbf{D}||_F^2} \tag{4}$$

is calculated.

The top plot of Figure 6 shows the effect of percentage missing has on the computational time. For each percentage, every method but `SDP` was applied to the same five different incomplete missing at random matrices; the `SDP` method took so long that it was applied to only the first of the five matrices. Not surprisingly computational time increases with the percentage missing. Comparing methods, we see that the `DPF` method of Trosset [2000] is consistently faster than the `NPF` method of Fang and O'Leary [2012], which in turn is consistently faster than the `SDP` method of Alfakih et al. [1999]. Computational times are given on a logarithmic scale so these differences are substantial. Some variation in the times taken can also be seen, especially for the larger percentages.

The minimal spanning tree case has the greatest percentage missing possible and appears at the far right of each plot. All times to completion here appear to have dropped, with `NPF` and `DPF` switching positions to make it fastest of the three methods. Since this is the minimal spanning tree case, the two methods of Section **??** can be added. Not surprisingly, `DPFLB` which differs from

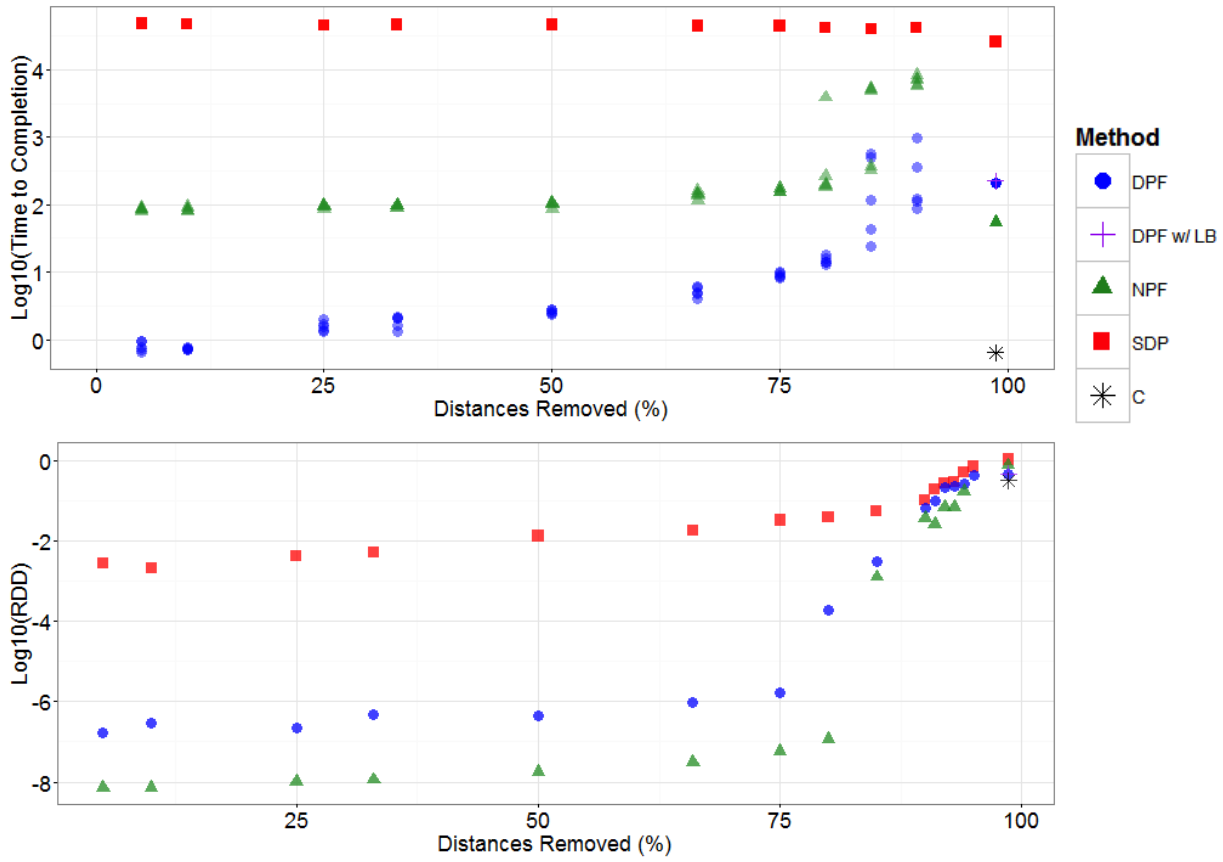


Figure 6: *Effect of increasing the percentage of missing distances on each method. Methods are coded by colour and symbol shape. For each percentage, several different matrices with different patterns of randomly selected missing data were used. In the top plot, each point symbol represents the result of one such matrix with one method; alpha blending of colour is used so that places where the values are essentially the same will appear more saturated due to over-plotting and the blending of the colours. In the bottom plot, only the average values are shown.*

`DPF` only in having precomputed nonzero lower bounds for every missing distance takes essentially the same time to complete as does `DPF`. More interestingly, the constructive method `C` is orders of magnitude faster than all other methods.

The lower plot of Figure 6 shows the average accuracy with which the various methods reconstruct $\mathbf{D}$. On this logarithmic scale lower values indicate greater accuracy so the accuracy of every method decreases as the percentage missing increases – largely because the numerator in (4) has more non-zero entries while the denominator remains unchanged. For every percentage up to and including 85% missing, each method was applied to 50 different missing at random matrices, the exception being `SDP` which, because of the time required, was only applied to a single matrix each time. Beyond 85% only 10 different random matrices were used for each of `NPF` and `DPF`. Again the methods can be ordered: `NPF` is consistently most accurate and `SDP` consistently least accurate across all percentages missing.

4.2.2 Distances as a function of percentage missing

Here we compare the reconstructed distances $\widehat{\mathbf{D}} = [\widehat{d}_{ij}]$ with the actual distances $\mathbf{D} = [d_{ij}]$. Figure 7 plots the pairs $(d_{ij}, \widehat{d}_{ij})$ for all $i < j$ for a few of the missing percentages. Perfect reconstruction would be all points on the $y = x$ line; the box in each plot is the range of the original distances and is identical in size across all plots; scales are identical for the same percentage of missing distances.

For each case, `NPF` outperforms the other two – all distances appear within the box, appear on either side of the $y = x$ line and is nearer to this line in all cases. As the percentage missing increases, the reconstruction of all three methods degrades. Both `DPF` and `SDP` tend to produce ever larger distances in their reconstructions as the percentage increases. `DPF` does produce some distances that are smaller than the original too. In contrast, `SDP` has a tendency to consistently produce distances that are too large for every percentage, and produces much larger distances than does `DPF`. Overall `NPF` provides the best reconstructed distances and `SDP` the worst.

Turning to minimal spanning tree completions, Figure 8 shows the pairs $(d_{ij}, \widehat{d}_{ij})$ for all $i < j$ for a single reconstruction for all five methods. The shapes of the first four (`SDP`, `NPF`, `DPF`, and `DPFLB`) are surprisingly similar, each showing three different branches. All four have almost all distances $\widehat{d}_{ij} > d_{ij}$ and many larger than $\max_{ij} d_{ij}$, with `SDP` producing the largest distances, followed by `NPF`, then `DPFLB`. The last of these has larger distances than those of `DPF` likely because of the lower bounds which ensure that `DPFLB` is MST-preserving.

In marked contrast, the constructive method `C` produces a completion whose distances $\widehat{d}_{ij}$ are all within the range of the true distances d_{ij} and are much more nearly concentrated around the $y = x$ line. If anything, `C` seems more inclined to produce smaller distances than are necessary. This might be corrected by having the vectors generated in Algorithm 2 not be generated on a sphere uniformly but to favour directions that would increase distances to other points already in the tree.

4.2.3 Point reconstruction

Next, consider reconstructing the Iris point configuration using only the distances known in the minimum spanning tree. To standardize the comparisons between algorithms, the Iris data is first transformed to its principal directions from a singular value decomposition of $\mathbf{X} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^{\mathsf{T}}$ as $\mathbf{XV}$. This projects each point $\mathbf{x}_i$ onto a new coordinate system given by the new variates $V1$, $V2$, $V3$, and $V4$. This is done by using the `getConfig` function on the completed distance matrices.

First, create the partial distance matrix containing only the minimum spanning tree of the Anderson Iris dataset, as well as the related adjacency matrix (for those algorithms who require it)

```
R> set.seed(99)

R> #Read in the Data
R> Iris <- iris[,c(1:4)]
R> Iris <- Iris[-143,]     #Iris row 143 and 102 are identical

R> #Compute a Distance Matrix
R> IrisDistance <- as.matrix(dist(Iris))

R> #Compute the minimum spanning tree
R> IrisMST <- mst(IrisDistance)

R> #Create a new Distance Matrix Containing only the mst distances
R> n <- nrow(Iris)
```

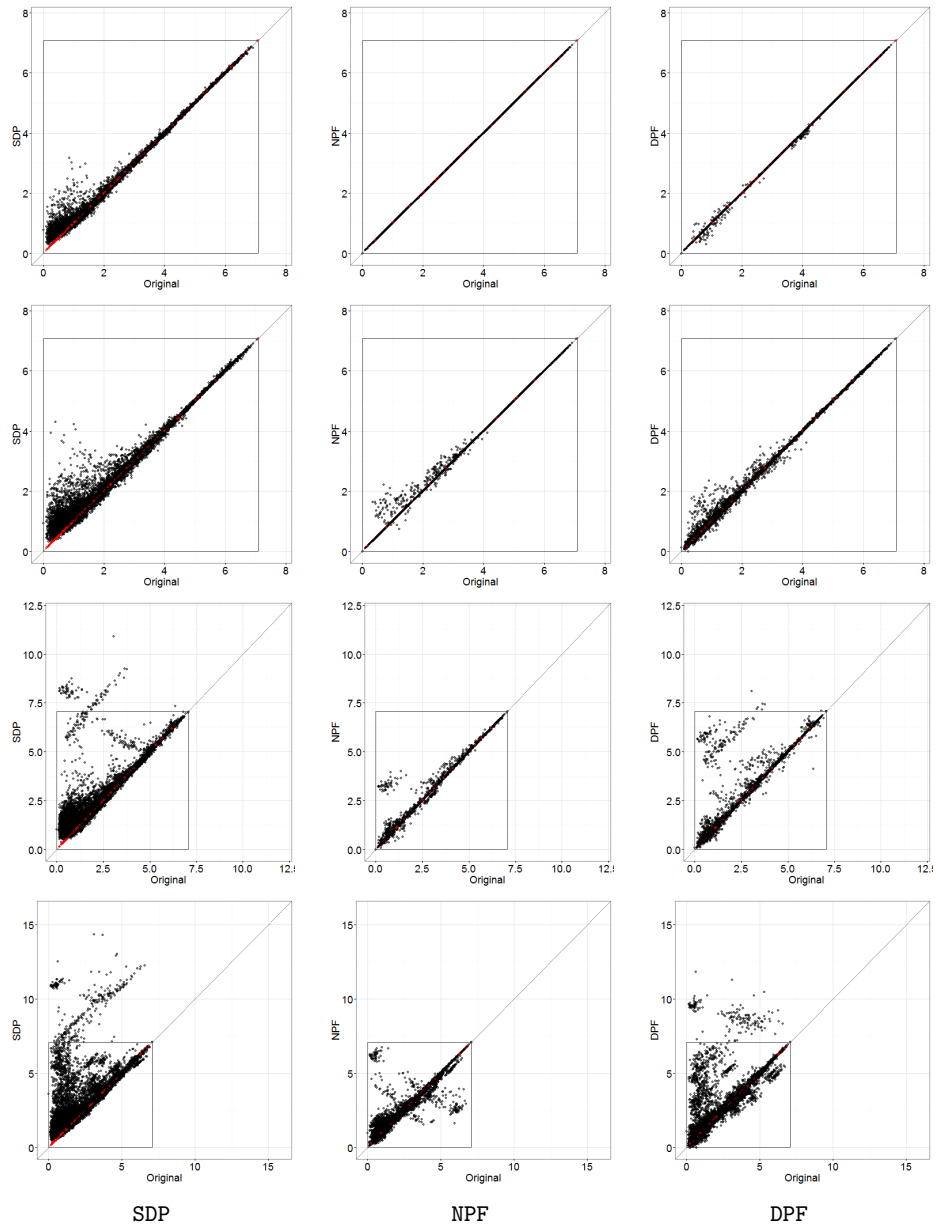


Figure 7: *Plots of the pairs of* $(d_{ij}, \widehat{d}_{ij})$ *for all* $i < j$ *for a single reconstruction of the Iris distance matrix; red values are the original minimal spanning tree distances. Perfect reconstruction would be all points on the* $y = x$ *line; the box in each plot is the range of the original distances and is identical in size across all plots; scales are identical for the same percentage of missing distances.*

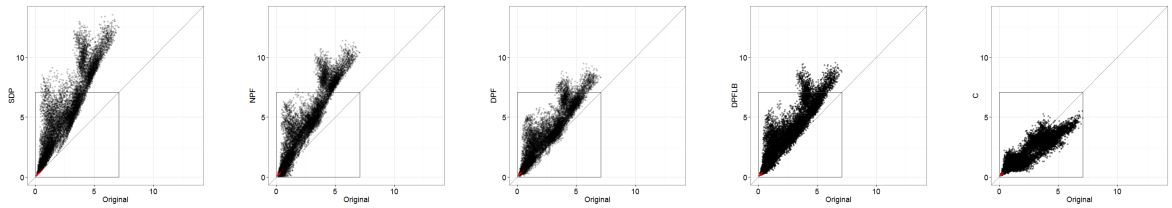


Figure 8: *Plots of* $(d_{ij}, \widehat{d}_{ij})$ *for all* $i < j$ *for a single reconstruction when the matrix to be completed contained only minimal spanning tree distances (shown in red); the* $y = x$ *line indicates perfect matching; the box in each plot shows the extent of the original distances.*

```
R> D <- matrix(rep(NA,n^2), nrow=n)
R> diag(D) <- 0

R> #Fill in the entries in D with the Iris MST distances

R> for(i in 1:n){
R>  D[IrisMST[i,1],IrisMST[i,2]] <- IrisMST[i,3]
R>  D[IrisMST[i,2],IrisMST[i,1]] <- IrisMST[i,3]
R> }

R> #Create Adjacency Matrix
R> A <- matrix(rep(0,n^2),nrow=n)
R> A[which(!is.na(D))] <- 1
```

Now, with a partial distance matrix in hand, use each of the algorithms to complete the matrix. Where possible, specify the embedding dimension to be four:

```
R> #Complete the Distance Matrix with each of the three completion methods
R> #NB: Long Run Time

R> IrisSDP <- edmc(D, method="sdp", A=A)
R> IrisNPF <- edmc(D, method="npf", A=A, d=4)
R> IrisDPF <- edmc(D, method="dpf", d=4)
R> IrisDPFLB <- edmc(D, method="dpf", d=4, retainMST = TRUE)
R> IrisC <- edmc(D, method="grs", d=4)
```

Finally, use the completed distance matrices to obtain a four dimensional point configuration for each completion method using `getConfig`

```
R> IrisSDPConfig <- getConfig(IrisSDP$D, 4)
R> IrisNPFConfig <- getConfig(IrisNPF$D, 4)
R> IrisDPFConfig <- getConfig(IrisDPF$D, 4)
R> IrisDPFLBConfig <- getConfig(IrisDPFLB$D, 4)
R> IrisCConfig <- getConfig(IrisC$D, 4)
```

Figure 9 shows the results of this experiment.

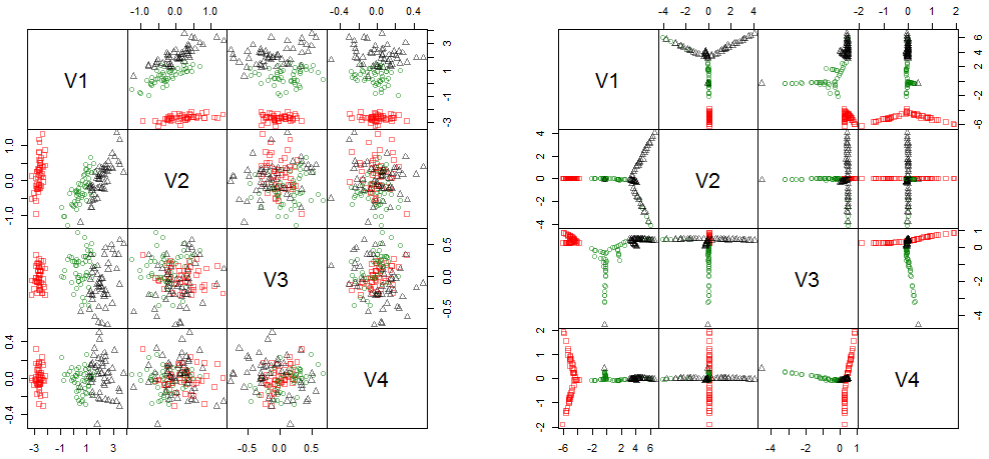
Of the methods considered, the point configuration produced by the guided random search algorithm most closely matches the original Iris data set. The other algorithms produce a star-shaped configuration which provide a good separation between species, but do a poor job reconstructing the original point configuration. Of these, it seems as though the configuration that was produced using the `DPF` algorithm with a specified mst-preserving lower bound provides a slightly noisier shape that most closely resembles the original configuration. This lends credibility to the concept of retaining the minimum spanning tree.

4.3 Distance-based dimension reduction

The statue dataset is a well known data set in dimension reduction due to the belief that it represents a three dimensional manifold, gaining popularity due to its use in the original Isomap manuscript (Tenenbaum et al. [2000]). It consists of 698 images in 64x64 greyscale of a bust at various angles and lighting conditions. Viewing each pixel as a point in a given dimension, we can view this data set as 698 point in 4096 dimensional space.

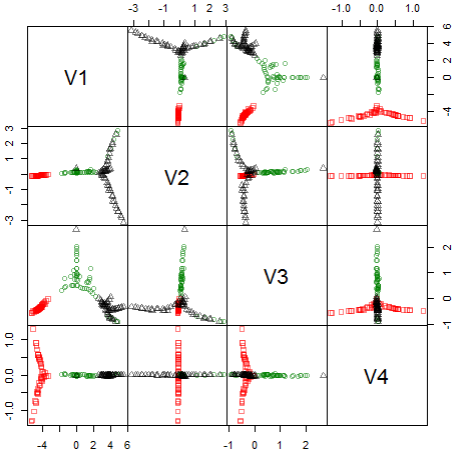
As discussed previously, some of the Euclidean distance matrix completion algorithms can also be used for dimension reduction. In particular, the algorithms that are capable of preserving the minimum spanning tree are most compelling, due to the clustering information contained in the mst. To explore this idea, consider using the `grs` algorithm to reduce the dimension of the statue data set from 4096 dimensions to three using only the minimum spanning tree. For comparison, we will also use the isomap algorithm of Tenenbaum et al. [2000] using a 6-nearest neighbour graph to be consistent with the original work. For this purpose the `Isomap` function in the RDRToolbox Bartenhagen [2014] is used. The statue data is available in edmcr. Figure 10, created using Loon (Waddell [2016]) illustrates the results.

```
R> library(RDRToolbox)

R> data(statue)

R> #Create partial distance matrix
R> out <- spantree(statue.dist)

R> statue.mst <- matrix(NA,nrow(statue),nrow(statue))
R> diag(statue.mst) <- 0

R> for(i in 2:nrow(statue)){
R>   statue.mst[i,out$kid[i-1]] <- out$dist[i-1]
R>   statue.mst[out$kid[i-1],i] <- out$dist[i-1]
R> }

R> #Dimension Reduction by isomap
R> out.isomap <- Isomap(statue.dist,3,6)

R> #Dimension Reduction by grs
R> out.grs <- grs(statue.mst,3)
```

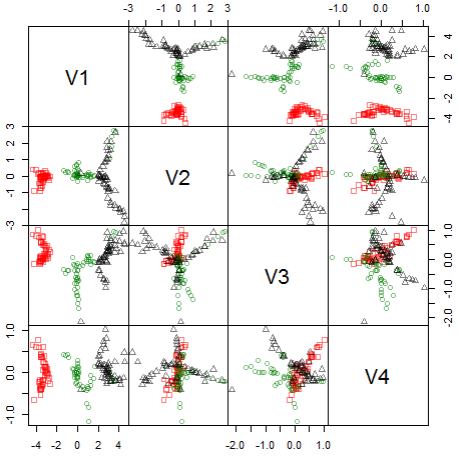


(a) Original

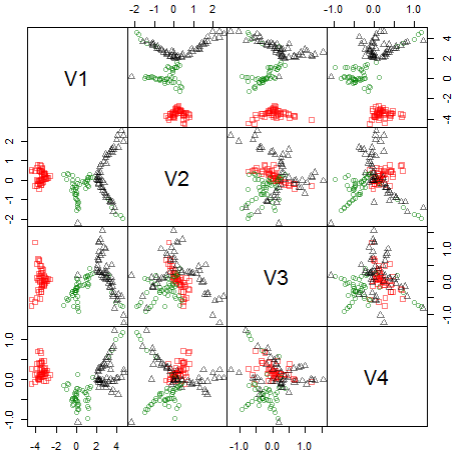
(b) `SDP`



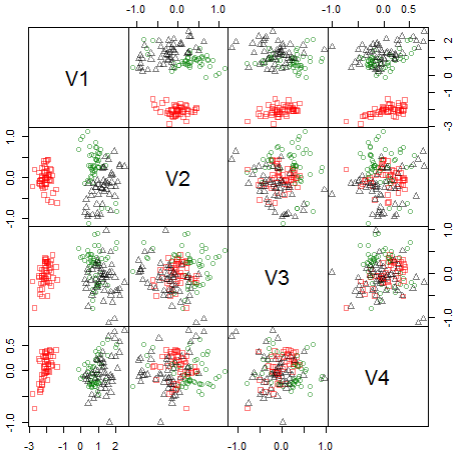
(c) `NPF`



(d) `DPF`



(e) `DPFLB`



(f) `C`

Figure 9: *Iris data point configurations reconstructed in the four dimensions given by each configuration's principal coordinates. The three species of flower are distinguished both by colour and by shape of the point symbols.*

Figure 10: *The result of dimension reduction to three dimensions by running isomap & `grs` on the statue manifold data set in 4096 dimension.*

There is no reason to expect these images to be "matching" in any way - and they don't - especially since Isomap is making use of far more information than grs. However, there are certainly similarities in the way that the faces were grouped. We can see in both, that darker faces tend to be grouped together - in isomap, these are placed on the far right of the image, while in grs, they are placed in the center. There is also an obvious fade from these dark faces, to lighter faces - in isomap, this occurs from the right of the image to the left of the image along the top, while in grs this occurs from the center of the image in both the right and left directions. There is also clear grouping in the orientation of the bust in both images.

Also, note that while isomap is capable of producing only a single configuration, grs is capable of producing many such configurations, allowing for the complete exploration of the mst-preserving solution space.

4.4 A simple sensor network localization problem

Consider a network containing four anchors, with known positions, and six sensors, where distances between sensors (and sensors and anchors) are known only they are within a radio range of 0.25 to each other. Since the anchors have known position, the distances between anchors are also known.

```
R>#Anchor Positions
R> anchors
       [,1]   [,2]
[1,] 0.5131 0.9326
[2,] 0.3183 0.3742
[3,] 0.5392 0.7524
[4,] 0.2213 0.7631

R>#Radio Range
R> R
[1] 0.25

R>#Distance Matrix
R> D
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,] 0.00   NA 0.20   NA 0.06   NA 0.02 0.23 0.00  0.09
 [2,]   NA 0.00 0.05   NA   NA 0.05   NA 0.21 0.22    NA
 [3,] 0.20 0.05 0.00   NA   NA 0.12   NA   NA 0.16    NA
 [4,]   NA   NA   NA 0.00   NA   NA   NA 0.23   NA    NA
 [5,] 0.06   NA   NA   NA 0.00   NA 0.11 0.09 0.06  0.02
 [6,]   NA 0.05 0.12   NA   NA 0.00   NA   NA   NA    NA
 [7,] 0.02   NA   NA   NA 0.11   NA 0.00 0.35 0.03  0.11
 [8,] 0.23 0.21   NA 0.23 0.09   NA 0.35 0.00 0.19  0.16
 [9,] 0.00 0.22 0.16   NA 0.06   NA 0.03 0.19 0.00  0.10
[10,] 0.09   NA   NA   NA 0.02   NA 0.11 0.16 0.10  0.00
```

Note the form of the distance matrix `D`. Rows/columns 7-10 correspond to the distances between the anchors and the sensors (in rows/columns 1-6), and the other anchors (rows/columns 7-10). Since the anchors have known positions, the distances between anchors is always known - there are

no unknown entries in the lower right 4×4 corner. Distances between anchors and sensors, on the other hand, are known only if the sensors are within the radio range of `R`. This problem can be solved using `edmc` as follows

```
R> edmc(D, method="snl", d=2, anchors=anchors)
\end{CodeInput}
\begin{CodeOutput}
          [,1]      [,2]
[1,] 0.4359152 0.7483225
[2,] 0.3870077 0.6819505
```

Of the six sensors to be placed, the algorithm was only able to localize the positions of two sensors. For a relatively sparse matrix, this is to be expected. This problem can also be run as an "anchorless" problem, in which case it is treated similarly to the edmc problems considered previously. Unlike the other algorithms however, `snl` returns the positions of only those sensors that were localized, resulting in the potential of only a partially-completed distance matrix. Running the example above as an anchorless problem results in the following

```
R> edmc(D, method="snl", d=2,anchors=NULL)
```

```
             [,1]         [,2]
[1,]  0.016934663  0.048956965
[2,] -0.004757603 -0.030583106
[3,] -0.007601688  0.156167720
[4,]  0.007282167 -0.193730966
[5,]  0.041458581  0.027923602
[6,] -0.053316120 -0.008734216
```

In this case, of the ten sensors, the algorithm was able to localize six, and their positions are given in the returned matrix `X`.

4.5 Reconstructing a protein molecule

To demonstrate the syntax and illustrate the output variables of `sprosr`, consider reconstructing a demo protein molecule from CYANA (Güntert [2004]. The data has been made available in edmcr. As the protein molecule is relatively large, only the first five positions are shown as output.

```
R> data(sprosr_upl)
R> data(sprosr_aco)
R> data(sprosr_seq)

R> out <- sprosr(sprosr_seq, sprosr_aco, sprosr_upl)


out$X
```

The individual elements of the `report` output variable can be accessed by appending the `$` operator to the element desired. The first five rows (where appropriate) of each element are displayed below.

```
out$report$eq_err

out$report$lo_error

out$report$up_error

out$report$chiral

out$report$hbond

out$report$dihed
```

5 Methodology

As before, $\mathbf{D}^*$ represents a squared Euclidean distance matrix, now with some entries possibly unknown. Let $\mathbf{A}$ be an adjacency matrix such that $a_{ij} = 1$ if d_{ij} is known, and 0 otherwise. In this way, we can make use of the Hadamard (element-wise) product $\mathbf{D}^* \circ \mathbf{A}$ whose only non-zero values will be the known entries in $\mathbf{D}^*$. This allows the completion problem to be stated as follows:

$$\boldsymbol{\Delta}_0 = \underset{\boldsymbol{\Delta} \in \mathcal{D}_n^-}{\arg\min} \, ||\mathbf{A} \circ (\mathbf{D}^* - \boldsymbol{\Delta})||_F^2 \tag{5}$$

where the norm taken is the Frobenius Norm, defined as $|||\mathbf{M}||_F = \sqrt{trace(\mathbf{M}^T\mathbf{M})}$. With a solution $\boldsymbol{\Delta}_0$ representing a completed (square) Euclidean distance matrix, we need only find a Gram matrix $\mathbf{G} = \mathbf{X}\mathbf{X}^T$, which can be done using the linear operator $\mathcal{K}$ defined previously. With a Gram matrix, a simple eigendecomposition allows us to obtain a point configuration.

5.1 Semidefinite programming algorithm

One of the classic algorithms in the Euclidean distance matrix completion literature is that of Alfakih et al. [1999], who propose a semidefinite programming approach to solving the problem. To do so, they modify equation 5 to work with a positive semi-definite matrix $\mathbf{S}$ instead of a squared distance matrix $\mathbf{D}^*$.

Alfakih et al. [1999] show that a squared Euclidean distance matrix $\mathbf{D}^*$ can be transformed to a positive semidefinite matrix $\mathbf{S}$ (and vice-versa), using the following operators:

$$\mathcal{T}_v(\mathbf{D}^*) = \mathbf{V}^T\mathcal{T}(\mathbf{D}^*)\mathbf{V} = -\frac{1}{2}\mathbf{V}^T\mathbf{D}^*\mathbf{V}$$
$$\mathcal{K}_v(\mathbf{S}) = \mathcal{K}(\mathbf{V}\mathbf{S}\mathbf{V}^T)$$

where $\mathbf{P} = \mathbf{V}\mathbf{V}^T$, where $\mathbf{V} \in \mathbb{R}^{n\times(n-1)}$ whose columns form an orthonormal basis for the subspace $\{x \in \mathbb{R}^n : x^T e = 0\}$, satisfying the following two properties:

$$\mathbf{V}^T\mathbf{1} = \mathbf{0}$$
$$\mathbf{V}^T\mathbf{V} = \mathbf{I}_{n-1}$$

These operators have been implemented in `edmcr` as the `edm2psd` and `psd2edm` functions. In each, a default value of $\mathbf{V}$ has been provided which satisfies the conditions specified above. However, the user is free to provide their own $\mathbf{V}$ as input as well.

Equation 5 can be written in terms of a positive semi-definite matrix $\mathbf{S}$ as follows:

$$\mathbf{S}_0 = \underset{\mathbf{S}\in\mathcal{S}^+_{n-1}}{\arg\min} ||\mathbf{A} \circ (\mathbf{D}^* - \mathcal{K}_v(\mathbf{S}))||_F^2 \tag{6}$$

The solution for this problem is a squared distance matrix, $\mathbf{\Delta}_0 = \mathcal{K}_v(\mathbf{S}_0)$. This formulation is the basis of the implementation of Alfakih et al. [1999], who show that the search space $\mathcal{S}^+_{n-1}$ can be expanded to the less restrictive set $\mathcal{P}^+_{n-1}$, the set of real $(n-1)\times(n-1)$ positive semi-definite matrices, allowing the problem to be solved using well known methods in semi-definite programming. Thus, the problem can be formulated as

$$\begin{aligned} \text{Minimize} \quad & f(\mathbf{S}) = ||\mathbf{A} \circ (\mathbf{D}^* - \mathcal{K}_V(\mathbf{S}))||_F^2 \\ \text{subject to:} \quad & a(\mathbf{S}) = \mathbf{c} \\ & \mathbf{S} \succeq 0. \end{aligned}$$

where the constraint $a(\mathbf{S}) = \mathbf{c}$ allows for the constraint that the known entries in $\mathbf{S}$ be preserved to be enforced during completion. This formulation results in a convex search space (Alfakih et al. [1999]), thus the minimum achieved is by definition the global minimum. The convexity of this algorithm does come at a price, as the run time for the algorithm is on the order of $\mathcal{O}(n^2)$, which makes it very slow for even moderately sized problems. The embedding dimension can also not be specified, resulting in completions with large embedding dimension.

5.2 Non-convex position formulation

Fang and O'Leary [2012] propose an algorithm which works with the point configuration $\mathbf{X}$ instead of the squared distance matrix $\mathbf{D}^*$, allowing for the embedding dimension of the completion to be specified. In doing so, the run time of the algorithm is on the order $\mathcal{O}(np)$, where p is the dimensionality of the point configuration, which comes at the cost of convexity of the algorithm. In considering the point matrix $\mathbf{X}$, Fang and O'Leary [2012] solve a modified completion problem

$$\begin{aligned} \underset{\mathbf{G}\in\mathcal{G}^+_n}{\text{Minimize}} \quad & ||\mathbf{A} \circ (\mathbf{D}^* - \mathcal{K}(\mathbf{G}))||_F^2 \\ \text{subject to:} \quad & \text{rank}(\mathbf{G}) = p. \end{aligned} \tag{7}$$

Recall that a Gram matrix $\mathbf{G}$ can be transformed to the space of squared distance matrices using the $\mathcal{K}$ operator. Also note that the rank of the Gram matrix has been fixed in the formulation, allowing for the desired completion dimension to be specified (which was an issue with the `SDP`

formulation). Fixing the embedding dimension p allows the problem to be written in terms of a point configuration $\mathbf{X}$

$$\underset{\mathbf{X}\in\mathbb{R}^{n\times p}}{\text{Minimize}} \quad f_{A,D}(\mathbf{X}) = ||\mathbf{A} \circ (\mathbf{D}^* - \mathcal{K}(\mathbf{X}\mathbf{X}^\mathsf{T}))||_F^2. \tag{8}$$

Due to the lack of convexity of the search space, Fang and O'Leary [2012] introduce a number of techniques in an attempt to give the algorithm a higher chance of reaching the global minimum.

First, a random start is employed for each run. Through empirical evidence, Fang and O'Leary [2012] randomly start their algorithm by creating a pre-Euclidean distance matrix $\widehat{\mathbf{F}} = [\widehat{f}_{ij}]$, where

$$\widehat{f}_{ij} = \frac{f_{ij}}{s_{ij}}$$

where f_{ij} is the shortest path between node i and j using the known distances, and $s_{ij} \sim TN(1.5, .3)$, TN being a truncated normal distribution with domain [1,k_{ij}], where k_{ij} is the number of segments in the shortest path between nodes i and j. Note that a pre-distance matrix is a matrix $\mathbf{F}$ that is both symmetric and hollow.

Also through empirical observation, Fang and O'Leary [2012] implement *stretching*, which is a numerical scaling of the point configuration $\mathbf{X}$ generated from the spectral decomposition of $\widehat{\mathbf{F}}$. For a vector $\mathbf{s}$, stretching would result in the matrix $\mathbf{sX}$. While it is noted that there is no theoretical basis for stretching in this context, Fang and O'Leary [2012] did observe an increased chance of landing on the global minimum in some instances where it was employed.

Finally, and perhaps most importantly, is the implementation of dimension relaxation. By artificially inflating the dimension of the point configuration, the `NPF` algorithm may be able to reach the global minimum with a higher probability. For instance, in the higher dimensional space, there may be a way to pass between local minima that would have been converged on in lower dimensional space on our way to the global minimum. Fang and O'Leary [2012] address three issues in implementing their dimension relaxation scheme. We present those issues here, and illustrate how they are handled in `edmcr`.

1. What is the largest dimension that should be considered? This is handled by the input variable `dmax` in `edmcr`. The default value of the highest dimension is set to be $n-1$, where n is the number of nodes in $\mathbf{X}$. For instances where this is very large, a smaller dimension can be chosen.

2. How fast should the dimension be reduced? This is handled using the input variable `decreaseDim` in `edmcr`. Generally, a slower rate of reduction is preferable, however this results in a longer computation time. The largest jump allowable is to go directly from the higher dimensional space to the lower dimensional space, but Fang and O'Leary [2012] show that this results in inaccurate completions. The default is set to be 1.

3. How should the dimension be reduced? Fang and O'Leary [2012] provide two solutions:

   - Principal Component Analysis
   - Nonlinear Dimension Reduction

This is handled using the input variable `dimMethod`, which takes one of "Linear" or "NLP", with the default set to "Linear".

While the first method is well known - simple multidimensional scaling - the second dimension reduction technique requires some discussion. To decrease from dimension d' to d, the nonlinear dimension reduction is done through optimization, where the following program is solved:

$$\begin{aligned} \underset{W}{\text{Minimize}} \quad & ||W_2||_F^2 \\ \text{subject to:} \quad & f_{A,D}(W) < 0 \end{aligned}$$

where $W = [W_1,\ W_2] \in \mathbb{R}^{n \times d'}$ is a configuration in d'-dimensional space and W_1 has dimension d, where $d < d'$.

5.3 Dissimilarity parameterization formulation

The problem in Trosset [2000]is formulated by focusing on the set:

$$\mathcal{C}_n(\mathbf{A} \circ \mathbf{\Delta}^\star) = \{\mathbf{\Delta} \in \mathcal{C}_n : \mathbf{A} \circ \mathbf{\Delta} = \mathbf{A} \circ \mathbf{\Delta}^\star\} \tag{9}$$

which contains all of the completions of the partial dissimilarity matrix $\mathbf{A} \circ \mathbf{\Delta}^\star$, and where $\mathcal{C}_n$ is the set of $n \times n$ dissimilarity matrices. Next, let $\mathcal{D}_n^-(p)$ denote the $n \times n$ Euclidean distance matrices from p dimensional point configurations.

Then, a solution to the p dimensional embedding problem exists only if $\mathcal{C}_n(\mathbf{A} \circ \mathbf{D}^*) \cap \mathcal{D}_n^-(p) \neq \varnothing$. This intersection is shown to be non-empty in Trosset [2000] if and only if the following minimization has a global minimum of zero

$$\begin{aligned} \underset{\mathbf{G},\mathbf{\Delta}}{\text{Minimize}} \quad & ||\mathbf{G} - \mathcal{T}(\mathbf{\Delta}))||_F^2 \\ \text{subject to:} \quad & \mathbf{G} \in \mathcal{S}_n^+ \ \text{ and } \ \text{rank}(\mathbf{G}) \leq p \\ & \mathbf{\Delta} \in \mathcal{C}_n(\mathbf{A} \circ \mathbf{D}^*) \end{aligned} \tag{10}$$

Note that while this formulation is similar to the others we have considered, there is a large distinction in that that the masking matrix $\mathbf{A}$ and the known distances $\mathbf{A} \circ \mathbf{D}^*$ now appear as part of the constraints given by the set of allowed dissimilarity matrices $\mathcal{C}_n(\mathbf{A} \circ \mathbf{D}^*)$. This will allow us to impose restrictions on the unknown values.

Letting

$$\mathcal{C}_n(\mathbf{\Delta}^L, \mathbf{\Delta}^U) = \{\mathbf{\Delta} \in \mathcal{C}_n : \delta_{ij}^L \leq \delta_{ij} \leq \delta_{ij}^U\}$$

where $\mathbf{\Delta}^L = [\delta_{ij}^L]$ and $\mathbf{\Delta}^U = [\delta_{ij}^U]$ are both in $\mathcal{C}_n$, we see that formulation (10) is a special case of

$$\begin{aligned} \underset{\mathbf{G},\mathbf{\Delta}}{\text{Minimize}} \quad & ||\mathbf{G} - \mathcal{T}(\mathbf{\Delta}))||_F^2 \\ \text{subject to:} \quad & \mathbf{G} \in \mathcal{S}_n^+ \ \text{ and } \ \text{rank}(\mathbf{G}) \leq p \\ & \mathbf{\Delta} \in \mathcal{C}_n(\mathbf{\Delta}^L, \mathbf{\Delta}^U) \end{aligned} \tag{11}$$

where $\mathbf{\Delta}^L$ and $\mathbf{\Delta}^U$ are fixed matrices chosen such that $\mathbf{A} \circ \mathbf{\Delta}^L = \mathbf{A} \circ \mathbf{\Delta}^U$. That is, when d_{ij} is known, the upper and lower bounds are set equal to d_{ij}. Whenever $a_{ij} = 0$, then the default value of the bounds are $\delta_{ij}^L = 0$ and $\delta_{ij}^U = +\infty$. An advantage of this formulation is that tighter bounds can be imposed on the unknown d_{ij}, restricting the space of possible solutions. For example using the structure of the graph given by $\mathbf{A}$, upper bounds δ_{ij}^U can be determined for all unknown δ_{ij} by simply invoking the triangle inequality.

Trosset [2000] use the L-BFG-S (Zhu et al. [1994]) optimization algorithm to efficiently solve problem (11), which we will denote by `DPF` to emphasize its dissimilarity parameterized formulation. Again, this formulation is non-convex. Similar to Fang and O'Leary [2012], Trosset [2000] introduces a random start algorithm to increase the probability of convergence to the global minimum.

5.3.1 Retaining the minimum spanning tree

As mentioned previously, it may be desirable to retain the minimum spanning tree during the completion of a Euclidean distance matrix. Since the dissimilarity parameterization formulation allows for a non-zero lower bound to be implemented for all known d^*_{ij}, if a lower bound D^*_L can be found such that for any unknown d^*_{ij}, having $d^*_{ij} > d^*_{L_{ij}}$ results in the preservation of the underlying minimum spanning tree. Such an algorithm is presented in Algorithm 1. This algorithm is implemented in `edmcr` in the `mstLB` function, which takes a (partial) Euclidean distance matrix $\mathbf{D}$ and returns a matrix containing the mst-preserving lower bounds for each d_{ij}, regardless of whether d_{ij} is known.

Algorithm 1 MST lower bounds algorithm

Structures
tree: $T = (V, E)$ is a spanning tree with vertex set $V = \{v\}$ and edge set $E = \{e\}$;
edges: e will be a set of two indices $\{i, j\} = nodes(e)$ and have a weight $wt(e) = \delta_{ij} \geq 0$;
$\mathbf{\Delta}^L = [\delta^L_{ij}]$ is the matrix of dissimilarity lower bounds to be determined;

procedure splitTree($T, splitEdge$) ▷ T is a spanning tree
  $restEdges \leftarrow edges(T) - \{splitEdge\}$ ▷ Remove $splitEdge$ from the edge set of T
  $(v_1, v_2) \leftarrow nodes(splitEdge)$
  $V_1 \leftarrow \{v_1\}$; $E_1 \leftarrow \{e \in restEdges : v_1 \in nodes(e)\}$ ▷ E_1 could be empty
  $V_2 \leftarrow \{v_2\}$; $E_2 \leftarrow \{e \in restEdges : v_2 \in nodes(e)\}$ ▷ E_2 could be empty
  while $restEdges \neq \varnothing$ **do**
    $e \leftarrow restEdges[1]$
    $restEdges \leftarrow restEdges - \{e\}$
    if $nodes(e) \cap nodes(E_1)$ **then**
      $E_1 \leftarrow E_1 \cup \{e\}$
    else
      $E_2 \leftarrow E_2 \cup \{e\}$
    end if
  end while
  $V_1 \leftarrow V_1 \cup nodes(E_1)$; $V_2 \leftarrow V_2 \cup nodes(E_2)$
  return $\{T_1 := (V_1, E_1),\ \ T_2 := (V_2, E_2)\}$ ▷ Return the two trees
end procedure

procedure MSTLowerBounds($T, \mathbf{\Delta}^L$) ▷ Recursively determines the lower bounds
  if $edges(T) \neq \varnothing$ **then** ▷ Ensure there are edges left in T
    $maxEdge \leftarrow \arg\max_{e \in edges(T)} wt(e)$ ▷ Split on the biggest edge
    $Trees \leftarrow$ splitTree($T, maxEdge$)
    for $v_1 \in nodes(Trees[T_1])$ **do**
      for $v_2 \in nodes(Trees[T_2])$ **do**
        $\mathbf{\Delta}^L[v_1, v_2] \leftarrow \mathbf{\Delta}^L[v_2, v_1] \leftarrow wt(maxEdge)$ ▷ Set the lower bound
      end for
    end for
    for $Tree \in Trees$ **do**
      $\mathbf{\Delta}^L \leftarrow$ MSTLowerBounds($Tree, \mathbf{\Delta}^L$) ▷ Recursively get lower bounds
    end for
  end if
  return $\mathbf{\Delta}^L$ ▷ Return the matrix of lower bounds
end procedure

5.4 Guided random search

In the guided random search algorithm, since only distances of the minimum spanning tree are known, the search space of possible completions is quite large. To exploit this, we simply construct

a completion by locating points $\mathbf{x}_i \in \mathbb{R}^p$ (p being the embedding dimension) one at a time, while checking that the (partial) minimal spanning tree is preserved as each point is added.

More formally, we define $\mathbf{X}_k = [\mathbf{x}_1, \ldots, \mathbf{x}_k]^\mathsf{T}$ to be a $k \times p$ matrix whose rows are point locations $\mathbf{x}_1, \ldots, \mathbf{x}_k \in \mathbb{R}^p$. The locations are chosen so that the minimal spanning tree from the Euclidean distances of these k locations in $\mathbb{R}^p$ is identical to that of k connected nodes from the minimal spanning tree $\mathbf{A} \circ \mathbf{D}$. The matrices $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n$ are constricted by growing (and preserving) the minimal spanning tree one node at a time. The distance matrix from $\mathbf{X}_n$ provides an mst-preserving completion.

The construction begins by choosing the node of maximal degree from the minimal spanning tree of $\mathbf{A} \circ \mathbf{D}$ and locating it at $\mathbf{x}_1 = \mathbf{0}$. The second node to locate will be that of maximal degree amongst those connected to the first. If the dissimilarity between these two nodes is, say, δ_{12}, then the location of $\mathbf{x}_2$ is chosen at random from a uniform distribution on the surface of a sphere S^{p-1} in $\mathbb{R}^p$ of radius $\sqrt{\delta_{12}}$ centered at $\mathbf{x}_1$ (assuming squared distances for the completion matrix). The two points $\mathbf{x}_1$ and $\mathbf{x}_2$ are trivially a subtree of the minimal spanning tree. The remaining nodes with connections to $\mathbf{x}_1$ are then added in similar fashion.

As each location is proposed, its (squared) distance to all other placed points is calculated and the resulting distance matrix checked to see whether the minimal spanning tree (so far) is preserved. If it is, the point is accepted; if not, points are generated until one is acceptable. When new nodes are added, they are chosen amongst those without locations that share an edge in the minimal spanning tree with nodes already located. At each step, available nodes of highest degrees are added before nodes of low degree in $\mathbf{A} \circ \mathbf{D}$; since these are harder to place, they appear earlier.

Algorithm 2 describes the method in detail, which we denote by `C`. More technical details of this algorithm can be found in Rahman and Oldford [2016].

6 Concluding remarks

Through a series of motivating examples, we introduced the problem of Euclidean distance matrix completion, and its use in a number of fields. The `R` package `edmcr`, and the main function `edmc` containing several well known edmc algorithms, were the introduced to solve these problems in `R`.

First, the generic Euclidean distance matrix completion problem was considered in which a given distance matrix with some unknown distances is given. The target solution is to then complete the matrix, while retaining a proper Euclidean distance matrix. For this purpose, three algorithms were introduced, the `sdp` algorithm of Alfakih et al. [1999], the `npf` algorithm of Fang and O'Leary [2012], and the `dpf` algorithm of Trosset [2000]. Each of these algorithms were shown to have their own strengths and weaknesses through simple examples.

Next, the sensor network localization problem was considered, where the goal was to to localize the positions of sensors with unknown positions. There is two major differences between this problem and general Euclidean distance matrix completion problem - there is a number of anchor points with known positions, and distances are only known between sensors and between sensors and anchors if they are within some radio range R in Euclidean distance. To solve this problem, the sensor network localization algorithm of Krislock and Wolkowicz [2010] was implemented in the `snl` function.

Next, the potential role of Euclidean distance matrix completion in dimension reduction was considered, drawing comparisons to the popular Isomap algorithm. Instead of filling in unknown distances with shortest paths, these distances were instead treated as being unknown, resulting in a partial Euclidean distance matrix. Euclidean distance matrix completion algorithms were used to complete the partial distance matrix. It was shown that the guided random search algorithm of Rahman and Oldford [2016] and the mst-preserving alteration of Trosset [2000] performed adequately

Algorithm 2 Constructive Completion Algorithm

Structures

$T_V = (V, E)$ is a tree spanning the vertex set $V = \{1, \ldots, n\}$ with edge set E;
$\mathbf{A} = [a_{ij}]$ and $\mathbf{A}^* = [a^*_{ij}]$ are $n \times n$ symmetric adjacency matrices;
$\mathbf{\Delta} = [\delta_{ij}]$, is an $n \times n$ symmetric squared dissimilarity matrix;
$\mathbf{X}$ is an $n \times p$ point configuration matrix to be constructed.

procedure MSTCONFIGURE($\mathbf{A}, \mathbf{\Delta}$, `maxIn = 100`, `maxOut = 100`)
  `nTries` $\leftarrow 0$; `Converged?` $\leftarrow$ `FALSE`; $T_V \leftarrow (V, E) \leftarrow tree(\mathbf{A})$;
  repeat
    `nTries` $\leftarrow$ `nTries` $+ 1$
    $\mathbf{X} \leftarrow \mathbf{0}$; $\boldsymbol{A} \leftarrow \mathbf{0}$; ▷ Initialization
    $i \leftarrow \arg\max_{j \in V} degree(node(j))$ ▷ Start at any maximal degree node i in T_V
    $P \leftarrow \{i\}$ ▷ Initial vertex set
    $T_P \leftarrow (P, \varnothing)$ ▷ Root the tree
    `Grow?` $\leftarrow$ `TRUE` ▷ Keep growing flag

    while `Grow?` **do**
      $(g, B) \leftarrow$ GETBUDS(T_P, T_V) ▷ grow $B \subset V$, $B \cap P = \varnothing$ from $g \in T_P$
      for $b \in B$ **do**
        $(T_P, \mathbf{\Delta}, \mathbf{A}, \mathbf{X}$, `Converged?`, `Grow?`$) \leftarrow$ GROWTREE($g, b, T_P, T_V, \mathbf{\Delta}, \mathbf{A}, \mathbf{X}$, `maxIn`)
      end for
    end while
  until `nTries` $>$ `maxOut` or `Converged?` `== TRUE`
  return ($\mathbf{X}$, `Converged?`) ▷ Return the point configuration
end procedure

procedure GROWTREE($i, j, T_P, T_V, \mathbf{\Delta}, \mathbf{A}, \mathbf{X}$, `maxTries` $= 100$)
  `Placed?` $\leftarrow$ `Converged?` $\leftarrow$ `FALSE`; `nTries` $\leftarrow 0$; $\mathbf{A}^* \leftarrow \mathbf{A}$; $\mathbf{\Delta}^* \leftarrow \mathbf{\Delta}$; ▷ Initialization
  repeat
    `nTries` $\leftarrow$ `nTries` $+ 1$
    $\mathbf{z} \sim$ `Uniform`(S^{p-1}) ▷ Generate a random direction vector
    $\mathbf{x}_j \leftarrow \mathbf{x}_i + \mathbf{z} \times \sqrt{\delta_{ij}}$ ▷ Propose the point
    for $k \in P$ **do** ▷ Try values for all placed nodes
      $\delta^*_{jk} \leftarrow \delta^*_{kj} \leftarrow ||\mathbf{x}_k - \mathbf{x}_j||^2$
      $a^*_{jk} \leftarrow a^*_{kj} \leftarrow 1$
    end for
    if MST($\mathbf{A}^* \circ \mathbf{\Delta}^*$) $\subset T_V$ **then** `Placed?` $\leftarrow$ `TRUE` ▷ Preserves the MST?
    end if
  until `Placed?` or `nTries` $>$ `maxTries`
  if `Placed?` **then** ▷ Accept the point
    $\mathbf{X}[j,] \leftarrow \mathbf{x}_j^\mathsf{T}$; $\mathbf{A} \leftarrow \mathbf{A}^*$; $\mathbf{\Delta} \leftarrow \mathbf{\Delta}^*$;
    $nodes(T_P) \leftarrow P \cup \{j\}$; $edges(T_P) \leftarrow edges(T_P) \cup \{(i,j),(j,i)\}$
    if $T_P = T_V$ **then** `Converged?` $\leftarrow$ `TRUE`
    end if
  end if
  return ($T_P, \mathbf{\Delta}, \mathbf{A}, \mathbf{X}$, `Converged?`, `Placed?`)
end procedure

procedure GETBUDS(T_P, T_V)
  $B \leftarrow V - P$
  $E_* = \{(i,j) : i \in P, j \in B, (i,j) \in edges(T_V)\}$ ▷ edges in T_V connecting P and B
  $g \leftarrow \arg\max_{i \in P} \#\{e : e \in E_* \text{ and } i \in e\}$ ▷ $g \in P$ having most connections to B
  $B \leftarrow \{b : (g,b) \in E_*\}$ ▷ reduce B to nodes connected to g
  return (g, B)
end procedure

in this regard using both the Iris and statue data sets.

Finally, the problem of molecular conformation was considered. To solve this problem, the function `sprosr` was introduced, an `R` implementation of algorithm developed by Ramandi [2011]. Through the use of a simple demo molecule provided by CYANA, the specialized input syntax and the information provided as output were demonstrated.

edmcr represents the first attempt at Euclidean distance matrix completion, sensor network localization, and molecular conformation in `R`, significantly broadening the range of problems that can be solved.

References

Abdo Y Alfakih, Amir Khandani, and Henry Wolkowicz. Solving euclidean distance matrix completion problems via semidefinite programming. *Computational optimization and applications*, 12 (1-3):13–30, 1999.

Abdo Y Alfakih, Miguel F Anjos, Veronica Piccialli, and Henry Wolkowicz. Euclidean distance matrices, semidefinite programming and sensor network localization. *Portugaliae Mathematica*, 68(1):53, 2011.

E Anderson. The irises of the gaspe peninsula. *Bulletin of the American Iris Society*, 59:2–5, 1935.

Christoph Bartenhagen. *RDRToolbox: A package for nonlinear dimension reduction with Isomap and LLE.*, 2014. R package version 1.20.0.

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, volume 14, pages 585–591, 2001.

Yichuan Ding, Nathan Krislock, Jiawei Qian, and Henry Wolkowicz. Sensor network localization, euclidean distance matrix completions, and graph realization. *Optimization and Engineering*, 11 (1):45–66, 2010.

Haw-ren Fang and Dianne P O'Leary. Euclidean distance matrix completion problems. *Optimization Methods and Software*, 27(4-5):695–717, 2012.

Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7 (2):179–188, 1936.

Jerome H. Friedman and Lawrence C. Rafsky. Multivariate generalizations of the wald-wolfowitz and smirnov two-sample tests. *The Annals of Statistics*, 7(4):697–717, 1979. ISSN 00905364. URL `http://www.jstor.org/stable/2958919`.

Jerome H. Friedman and Lawrence C. Rafsky. Graphics for the multivariate two-sample problem. *Journal of the American Statistical Association*, 76(374):277–287, 1981. ISSN 01621459. URL `http://www.jstor.org/stable/2287825`.

W Glunt, TL Hayden, and M Raydan. Molecular conformations from distance matrices. *Journal of Computational Chemistry*, 14(1):114–120, 1993.

John C Gower and GJS Ross. Minimum spanning trees and single linkage cluster analysis. *Applied statistics*, pages 54–64, 1969.

Peter Güntert. Automated nmr structure calculation with cyana. *Protein NMR Techniques*, pages 353–378, 2004.

Timothy F Havel and Kurt Wüthrich. An evaluation of the combined use of nuclear magnetic resonance and distance geometry for the determination of protein conformations in solution. *Journal of molecular biology*, 182(2):281–294, 1985.

Bruce Hendrickson. The molecule problem: Exploiting structure in global optimization. *SIAM Journal on Optimization*, 5(4):835–857, 1995.

Nathan Krislock and Henry Wolkowicz. Explicit sensor network localization using semidefinite representations and facial reductions. *SIAM Journal on Optimization*, 20(5):2679–2708, 2010.

Nathan Krislock and Henry Wolkowicz. Euclidean distance matrices and applications. In Miguel Anjos and Jean Lasserre, editors, *Handbook on Semidefinite, Cone and Polynomial Optimization: Theory, Algorithms, Software and Applications*, volume 166 of *International Series in Operations Research & Management Science*, chapter 30, pages 879–914. Springer Science & Business Media, 2011.

Maryann Lawlor. Small systems, big business. Signal Magazine, 2005.

Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97. ACM, 2002.

Marvin McNett and Geoffrey M Voelker. Access and mobility of wireless pda users. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(2):40–55, 2005.

Jorge J Moré and Zhijun Wu. Distance geometry optimization for protein structures. *Journal of Global Optimization*, 15(3):219–234, 1999.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

Adam Rahman and Wayne Oldford. Euclidean distance matrix completion and point configurations from the minimal spanning tree. *SIAM Journal on Optimization*, 28(1):528–550, 2018.

Babak Alipanahi Ramandi. *New approaches to protein NMR automation*. PhD thesis, University of Waterloo, 2011.

R. Rangarajan, R. Raich, and A. O. Hero. Euclidean matrix completion problems in tracking and geo-localization. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5324–5327, March 2008. doi: 10.1109/ICASSP.2008.4518862.

Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

Robert Szewczyk, Eric Osterweil, Joseph Polastre, Michael Hamilton, Alan Mainwaring, and Deborah Estrin. Habitat monitoring with sensor networks. *Communications of the ACM*, 47(6):34–40, 2004.

Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

Michael W Trosset. Applications of multidimensional scaling to molecular conformation. In *Computing Science and Statistics*, volume 29, pages 148–152, 1998.

Michael W Trosset. Distance matrix completion by numerical optimization. *Computational Optimization and Applications*, 17(1):11–22, 2000.

Adrian Waddell. *Interactive Visualization and Exploration of High-Dimensional Data*. PhD thesis, University of Waterloo, 2016.

Kurt Wüthrich. Protein structure determination in solution by nmr spectroscopy. *Journal of Biological Chemistry*, 265(36):22059–22062, 1990.

Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Lbfgs-b: Fortran subroutines for large-scale bound constrained optimization. *Report NAM-11, EECS Department, Northwestern University*, 1994.