Fast Multidimensional Scaling using Vector Extrapolation

Guy Rosman Alexander M. Bronstein
Michael M. Bronstein Avram Sidi Ron Kimmel

February 5, 2008

Abstract

Multidimensional scaling (MDS) is a class of methods used to find a low-dimensional representation of a set of points given a matrix of pairwise distances between them. Problems of this kind arise in various applications, from dimensionality reduction of image manifolds to psychology and statistics. In many of these applications, efficient and accurate solution of an MDS problem is required. In this paper, we propose using vector extrapolation techniques to accelerate the numerical solution of MDS problems. Vector extrapolation is used to accelerate the convergence of fixed-point iterative algorithms. We review the problem of multidimensional scaling and vector extrapolation techniques, and show several examples of our accelerated solver for multidimensional scaling problems in various applications.

1 Introduction

Multidimensional scaling (MDS) [5] is a generic name for a family of algorithms that, given a matrix representing the pairwise distances between a set of points in some abstract metric space, attempts to find a representation of these points in a low-dimensional (typically Euclidean) space. The distances in the target space should be as close to the original ones as possible. MDS algorithms are of great importance in the field of multidimensional data analysis. Originally introduced in the field of psychology [53, 29], MDS algorithms have since then been applied to various applications. The most common applications include dimensionality reduction [40, 52], visualization and analysis of data (for example, financial data ([15, 33, 30]), information retrieval [2], graph visualization [23], texture mapping in computer graphics [58], bioinformatics [22, 26, 16], etc [35, 54, 13]. More recently, MDS methods have been brought into the computer vision community as efficient methods for non-rigid shape analysis and recognition [20, 8].

The data sets encountered in the above applications are often of a very large size. At the same time, nonlinear and non-convex nature of MDS problems
tends to make their solution computationally demanding. As a result, MDS algorithms tend to be slow, which makes their practical application in large-scale problems challenging or even prohibitive. A number of low-complexity algorithms that find an approximate solution to an MDS problem have been recently proposed for large-scale settings [21, 12, 55, 17, 32, 57, 6]. Yet, some of the applications (for example, the representation of intrinsic geometry of non-rigid shapes in computer vision) require a (numerically) exact solution, which makes approximate MDS algorithms inappropriate. Recently, Bronstein et al. proposed an efficient multigrid solver for the exact MDS problem used for non-rigid shapes recognition [10]. The method showed significant improvement over traditional exact MDS algorithms. It made, however, heavy use of the underlying problem geometry. The generalization of this method to generic MDS problems, where the underlying geometry is not given explicitly, is not straightforward.

In this paper we propose a new efficient approach for solving the exact MDS problem. Our method is based on vector extrapolation techniques [51]. These techniques try to approximate the limit of a given sequence of iteration vectors, resulting from an iterative process. The limit is obtained as a linear combination of the last few iteration vectors. The vectors of the sequence in our case are the iteration vectors resulting from an iterative solution technique for the MDS problem.

This paper is organized as follows. In Section 2, we formulate the MDS problems addressed in the following sections and review some algorithms used for their solutions. In Section 3, we review vector extrapolation techniques and discuss their application to the solution of the MDS problem. We present a few numerical examples in Section 4, which demonstrate the efficiency of the proposed method. Section 5 concludes the paper with a discussion on possible future directions.

2 Multidimensional Scaling

Multidimensional scaling is a very broad term, encompassing both a wide class of problems and algorithms used for their solution. Generally speaking, an MDS problem consists of finding a representation for a set of $N$ data points, given a set of dissimilarities between them. MDS problems (and as a result, algorithms thereof) differ in the definition of the dissimilarities and the manner in which the desired representation should approximate the dissimilarities (we refer the reader to [5] for a detailed survey).

When all that is expected of the representation is a correct rank ordering of the distances between various point pairs, the problem is usually referred to as ordinal MDS. Here, we focus on another subset of MDS problems called metric MDS, in which the data is assumed to arise from an unknown metric space, and the dissimilarities are distances satisfying metric axioms. Metric MDS problems attempt to make the resulting distances as close as possible to the given ones.
2.1 Least-squares MDS

Let us denote the coordinates of the new representation as \( \{x_i\}_{i=1}^N \), assuming for simplicity that the points \( x_i \) belong to an \( m \)-dimensional Euclidean space. We write the coordinates in the form of an \( N \times m \) matrix \( X = (x_{ij}) \), where \( x_{ij} \) is the \( j \)th coordinate of point \( i \). Let \( d_{ij}(X) \) denote the Euclidean distance between the points \( x_i \) and \( x_j \), and let \( \delta_{ij} \) be the input distance, which we would like to preserve in the new representation.

A generic metric MDS problem can be formulated as the following optimization problem,

\[
\min_X \sum_{i<j} w_{ij} f_{ERR}(d_{ij}(X), \delta_{ij}),
\]

where \( w_{ij} \) are weights capturing the relative importance of each pairwise distance, and \( f_{ERR} \) is an error cost function used for comparing the input and approximated distances. Choosing the pairwise distance error function to be a quadratic term of the difference between the input and the resulting distances,

\[
f_{STRESS}(d, \delta) = (d - \delta)^2,
\]

we obtain the stress function. This is one of the most common objective functions, on which we will focus in the following discussion. Defining the error function as a quadratic term of the difference between squared distances,

\[
f_{SSTRESS}(d, \delta) = (d^2 - \delta^2)^2,
\]

results in a function commonly known as the sstress. This cost function tends to prefer large distances over the local distances [5].

2.2 SMACOF algorithm

Trying to minimize the stress function,

\[
s(X) = \sum_{i<j} w_{ij} (d_{ij}(X) - \delta_{ij})^2,
\]

we develop the gradient of \( s(X) \) with respect to \( X \), which can be written [5] as

\[
\nabla s(X) = 2 VX - 2B(X)X,
\]

where \( V \) and \( B \) are matrices with elements given by

\[
(V)_{ij} = \begin{cases} 
-w_{ij} & \text{if } i \neq j \\
\sum_{k \neq i} w_{ik} & \text{if } i = j
\end{cases}
\]

and

\[
(B)_{ij} = \begin{cases} 
-w_{ij}\delta_{ij}d_{ij}^{-1}(X) & \text{if } i \neq j \text{ and } d_{ij}(X) \neq 0 \\
0 & \text{if } i \neq j \text{ and } d_{ij}(X) = 0 \\
-\sum_{k \neq i} b_{ik} & \text{if } i = j
\end{cases}
\]
respectively. Using the first-order optimality condition, one obtains

\[ 2VX = 2B(X)X, \]

or alternatively,

\[ X = V^\dagger B(X)X, \] (3)

where \( \dagger \) denotes the matrix pseudoinverse. This equation gave rise to the following iterative scheme proposed by [25],

\[ X^{(k+1)} = V^\dagger B(X^{(k)})X^{(k)}, \] (4)

of which equation (3) can be regarded as the fixed point. Iteratively performing the transformation (3) converges to a local minimum of the stress cost function. This process is known as the SMACOF algorithm, standing from Scaling by Majorizing a Complicated Function [5]. It can be shown to be equivalent to a weighted gradient descent with constant step size (see, e.g., [10]).

A remarkable property of the stress function is that such an algorithm guarantees a monotonously decreasing sequence of stress values, which is uncommon in general optimization problems. This property is shown by developing the iteration formula (4) using a technique known as iterative majorization [5]. At the same time, the convergence of the SMACOF algorithm is slow, and a large number of iterations may be required if a high accuracy is needed, depending on the size of the data set and the composition of the distance matrix.

2.3 Classical Scaling

Another widely used cost function is the strain, giving rise to an algebraic MDS algorithm referred to as classical scaling. Let \( \Delta^{(2)} \) and \( D^{(2)}(X) \) denote the matrices of the squared input and target Euclidean distances, respectively. Let \( J \) be the centering matrix defined by \( J = I - \frac{1}{N}11^T \), whose application to a vector removes its constant component. Let us assume the coordinates of the solution have zero mean. Expressing the squared Euclidean distances, we obtain

\[ d^2(x_i, x_j) = \langle x_i - x_j, x_i - x_j \rangle = \langle x_i, x_i \rangle + \langle x_j, x_j \rangle - 2 \langle x_i, x_j \rangle, \] (5)

or, in matrix notation,

\[ D^2(X) = c1^T + 1c^T - 2XX^T. \]

(here \( c_i = \langle x_i, x_i \rangle \)).

Assuming the given distances to be Euclidean, and applying the centering matrices to the input squared distances matrix, would give us the matrix of inner products of the points (the Gram matrix),

\[ B_\Delta = \frac{1}{2}J\Delta^{(2)}J, \]
where in the case of Euclidean input distances, one would get

$$(B_\Delta)_{ij} = \langle x_i, x_j \rangle.$$

In practice, however, the input distances are not necessarily Euclidean, and $B_\Delta$ is therefore not necessarily equal to the dot products. The discrepancy between the two is a measure of the approximation quality, referred to as strain,

$$\| \frac{1}{2} J(D(2)(X) - \Delta(2)) J \|_F^2$$

$$= \| XX^T + \frac{1}{2} J(\Delta(2)) J \|_F^2$$

$$= \| XX^T - B_\Delta \|_F^2$$

(the norm here is the Frobenius norm).

The global optimum of the strain function can be found by means of eigendecomposition of the matrix $B_\Delta$. Classical scaling, however, is limited both in its versatility and adaptivity (specifically, arbitrary weights for distances, or variants on the cost function usually cannot be cast into an eigenvalue problem). Furthermore, its computational cost in large problems is significant, despite several approximation methods proposed for this problem [17, 57, 6].

These drawbacks make the least-squares MDS problem preferred, and thus in the following discussion, we limit ourselves to the stress function and the SMACOF algorithm. Yet, the fact that a global optimum is guaranteed makes classical scaling attractive for initializing the SMACOF algorithm [27].

### 3 Vector Extrapolation Methods

Looking at the patterns of changes exhibited by the point coordinates undergoing SMACOF iterations, it seems that the general direction of the convergence process can be inferred from past iterates. This hints at the possibility of exploiting the behavior of the iterates to speed up the convergence. One possible way to predict the limit of the convergence is by using methods of vector extrapolation, such as those presented in the following section.

Let us now present the rationale and theory behind vector extrapolation methods. Specifically, we focus on two methods found especially useful by us. The minimal polynomial extrapolation (MPE) [11] and the reduced rank extrapolation (RRE) [31, 19] are two vector extrapolation methods that have proved to be very efficient in accelerating the convergence of vector sequences that arise from fixed-point iteration schemes for nonlinear, as well as linear, large and sparse systems of equations. For a review of these methods and others, covering the relevant developments until mid 1980s, see [51]. The brief review we present here covers the various developments that have taken place since the publication of [51].
Both methods are derived by considering vector sequences $x_0, x_1, \ldots$, generated via a linear fixed-point iteration process, namely,

$$x_{n+1} = Ax_n + b, \quad n = 0, 1, \ldots, \quad (6)$$

where $A$ is a fixed $N \times N$ matrix and $b$ is a fixed $N$-dimensional vector and $x_0$ is an initial vector chosen by the user. Clearly, this sequence has a limit $s$ that is the unique solution to the linear system

$$x = Ax + b, \quad (7)$$

provided $\rho(A) < 1$, where $\rho(A)$ is the spectral radius of $A$. Note that the system in (7) can also be written as $(I - A)x = b$, and the uniqueness of the solution $s$ follows from our assumption that $\rho(A) < 1$, which guarantees that the matrix $I - A$ is nonsingular since 1 is not an eigenvalue of $A$.

We now turn to simple derivations of MPE and RRE, that are based on those given by [51]. Other derivations, based on the Shanks–Schmidt transformation [41, 39] have been given by [47].

Given the sequence $x_0, x_1, \ldots$, generated as in (6), let

$$u_n = \Delta x_n = x_{n+1} - x_n, \quad n = 0, 1, \ldots, \quad (8)$$

and define the error vectors $e_n$ as in

$$e_n = x_n - s, \quad n = 0, 1, \ldots \quad (9)$$

Making also use of the fact that $s = As + b$, one can relate the error in step $n$ to the initial error via

$$e_n = (Ax_{n-1} + b) - (As + b) = A(x_{n-1} - s) = A\epsilon_{n-1}, \quad (10)$$

which, by induction, gives

$$e_n = A^n \epsilon_0, \quad n = 0, 1, \ldots \quad (11)$$

We now seek to approximate $s$ by a “weighted average” of $k + 1$ consecutive $x_i$’s as in

$$s_{n,k} = \sum_{i=0}^{k} \gamma_i x_{n+i}; \quad \sum_{i=0}^{k} \gamma_i = 1. \quad (12)$$

Substituting Equation (8) in Equation (11), and making use of the fact that $\sum_{i=0}^{k} \gamma_i = 1$, we obtain

$$s_{n,k} = \sum_{i=0}^{k} \gamma_i (s + \epsilon_{n+i}) = s + \sum_{i=0}^{k} \gamma_i \epsilon_{n+i} \quad (13)$$

which, by (10), becomes

$$s_{n,k} = s + \sum_{i=0}^{k} \gamma_i A^{n+i} \epsilon_0. \quad (14)$$
From this expression, it is clear that we must choose the scalars \( \gamma_i \) to make the vector \( \sum_{i=0}^{k} \gamma_i A^{n+i} \mathbf{e}_0 \), the weighted sum of the error vectors \( \mathbf{e}_{n+i}, i = 0, 1, \ldots, k \), as small as possible. As we show next, we can actually make this vector vanish by choosing \( k \) and the \( \gamma_i \) appropriately.

Now, given a nonzero \( N \times N \) matrix \( \mathbf{B} \) and an arbitrary nonzero \( N \)-dimensional vector \( \mathbf{u} \), it is known that there exists a unique monic polynomial \( P(z) \) of smallest degree (that is at most \( N \)) that annihilates the vector \( \mathbf{u} \), that is, \( P(\mathbf{B})\mathbf{u} = \mathbf{0} \). This polynomial is called the \textit{minimal polynomial of} \( \mathbf{B} \) \textit{with respect to the vector} \( \mathbf{u} \). It is also known that \( P(z) \) divides the minimal polynomial of \( \mathbf{B} \), which divides the characteristic polynomial of \( \mathbf{B} \). Thus, the zeros of \( P(z) \) are some or all the eigenvalues of \( \mathbf{B} \).

Thus, if the minimal polynomial of \( \mathbf{A} \) with respect to \( \mathbf{e}_n \) is

\[
P(z) = \sum_{i=0}^{k} c_i z^i; \quad c_k = 1,
\]

then

\[
P(\mathbf{A})\mathbf{e}_n = \mathbf{0}.
\]

By (10), this means that

\[
\sum_{i=0}^{k} c_i A^i \mathbf{e}_n = \sum_{i=0}^{k} c_i \mathbf{e}_{n+i} = \mathbf{0}.
\]

This is a set of \( N \) linear equations in the \( k \) unknowns \( c_0, c_1, \ldots, c_{k-1} \), with \( c_k = 1 \). In addition, these equations are consistent and have a unique solution because \( P(z) \) is unique. From these equations, it seems, however, that we need to know the vector \( \mathbf{e}_n = \mathbf{x}_n - \mathbf{s} \), hence the solution \( \mathbf{s} \). Fortunately, this is not the case, and we can obtain the \( c_i \) solely from our knowledge of the vectors \( \mathbf{x}_j \).

This is done as follows: Multiplying Equation (14) by \( \mathbf{A} \), and recalling (9), we have

\[
0 = \sum_{i=0}^{k} c_i \mathbf{A} \mathbf{e}_{n+i} = \sum_{i=0}^{k} c_i \mathbf{e}_{n+i+1}.
\]

Subtracting from this Equation (14), we obtain

\[
0 = \sum_{i=0}^{k} c_i (\mathbf{e}_{n+i+1} - \mathbf{e}_{n+i}) = \sum_{i=0}^{k} c_i (\mathbf{x}_{n+i+1} - \mathbf{x}_{n+i}),
\]

hence the linear system

\[
\sum_{i=0}^{k} c_i \mathbf{u}_{n+i} = \mathbf{0}.
\]

Once \( c_0, c_1, \ldots, c_{k-1} \) have been determined from this linear system, we set \( c_k = 1 \) and let \( \gamma_i = c_i / \sum_{j=0}^{k} c_j, \; i = 0, 1, \ldots, k \). This is allowed because \( \sum_{j=0}^{k} c_j = P(1) \neq 0 \) by the fact that \( \mathbf{I} - \mathbf{A} \) is not singular and hence \( \mathbf{A} \) does
not have 1 as an eigenvalue. Summing up, we have shown that if $k$ is the degree of the minimal polynomial of $A$ with respect to $\epsilon_n$, then there exist scalars $\gamma_0, \gamma_1, \ldots, \gamma_k$, satisfying $\sum_{i=0}^{k} \gamma_i = 1$, such that $\sum_{i=0}^{k} \gamma_i x_{n+i} = s$.

At this point, we note that, $s$ is the solution to $(I - A)x = b$, whether $\rho(A) < 1$ or not. Thus, with the $\gamma_i$ as determined above, $s = \sum_{i=0}^{k} \gamma_i x_{n+i}$, whether $\lim_{n \to \infty} x_n$ exists or not.

In the sequel, we shall use the notation

$$U^{(j)}_s = [u_j | u_{j+1} | \ldots | u_{j+s}]$$

Thus, $U^{(j)}_s$ is an $N \times (j+1)$ matrix. In this notation, Equation (14) reads

$$U^{(n)}_k c = 0; \quad c = [c_0, c_1, \ldots, c_k]^T.$$  

Of course, dividing Equation (17) by $\sum_{i=0}^{k} c_i$, we also have

$$U^{(n)}_k \gamma = 0; \quad \gamma = [\gamma_0, \gamma_1, \ldots, \gamma_k]^T.$$  

### 3.1 Derivation of MPE

As we already know, the degree of the minimal polynomial of $A$ with respect to $\epsilon_n$ can be as large as $N$. This makes the process we have just described a prohibitively expensive one since we have to save all the vectors $x_{n+i}$, $i = 0, 1, \ldots, k+1$, which is a problem when $N$ is very large. In addition, we also do not have a way to know this degree. Given these facts, we modify the approach we have just described as follows: We choose $k$ to be an arbitrary positive integer that is normally (much) smaller than the degree of the minimal polynomial of $A$ with respect to $\epsilon_n$. With this $k$, the linear system in Equation (15) is not consistent, hence does not have a solution for $c_0, c_1, \ldots, c_{k-1}$, with $c_k = 1$, in the ordinary sense. Therefore, we solve this system in the least squares sense. Following that, we compute $\gamma_0, \gamma_1, \ldots, \gamma_k$ precisely as described following Equation (15), and then compute the vector $s_{n,k} = \sum_{i=0}^{k} \gamma_i x_{n+i}$ as our approximation to $s$. The resulting method is known as the minimal polynomial extrapolation method (MPE). Clearly, MPE takes as its input only the integers $k$ and $n$ and the vectors $x_n, x_{n+1}, \ldots, x_{n+k+1}$, hence can be employed whether these vectors are generated by a linear or nonlinear iterative process. We summarize the definition of MPE in Table 1.

### 3.2 Derivation of RRE

Again, we choose $k$ to be an arbitrary positive integer that is normally (much) smaller than the degree of the minimal polynomial of $A$ with respect to $\epsilon_n$. With this $k$, the linear system in Equation (18) is not consistent, hence does not have a solution for $\gamma_0, \gamma_1, \ldots, \gamma_k$, in the ordinary sense. Therefore, we solve this system in the least squares sense, subject to the constraint $\sum_{i=0}^{k} \gamma_i = 1$. 

---

Technion - Computer Science Department - Technical Report CIS-2008-01 - 2008
Choose the integers $k$ and $n$, and input the vectors $x_n, x_{n+1}, \ldots, x_{n+k}$.

1. Form the $N \times k + 1$ matrix $U_k^{(n)}$.

2. Solve the overdetermined linear system $U_k^{(n)} e' = -u_{n+k}$ by least squares. Here $e' = [c_0, c_1, \ldots, c_{k-1}]^T$.

3. Set $c_k = 1$, and $\gamma_i = c_i / \sum_{i=0}^k c_i$, $i = 0, 1, \ldots, k$.

4. Compute the vector $s_{n,k} = \sum_{i=0}^k \gamma_i x_{n+i}$ as approximation to $\lim_{i \to \infty} x_i = s$.

Table 1: The minimal polynomial extrapolation algorithm

Following that, we compute the vector $s_{n,k} = \sum_{i=0}^k \gamma_i x_{n+i}$ as our approximation to $s$. The resulting method is known as the reduced rank extrapolation method (RRE). Clearly, RRE, just as MPE, takes as its input only the integers $k$ and $n$ and the vectors $x_n, x_{n+1}, \ldots, x_{n+k+1}$, hence can be employed whether these vectors are generated by a linear or nonlinear iterative process. We summarize the definition of RRE in Table 2.

Table 2: The reduced rank extrapolation algorithm

3.3 Treatment of Nonlinear Equations

We now turn to the treatment of nonlinear equations, such as those used in the SMACOF algorithm, by vector extrapolation methods. Assume that the system of nonlinear equations in question has been written in the (possibly preconditioned) form

$$x = F(x),$$

where $F(x)$ is an $N$-dimensional vector-valued function and $x$ is the $N$-dimensional vector of unknowns. Let the sequence of approximations $x_n$ to the solution $s$ be generated via

$$x_{n+1} = F(x_n), \quad n = 0, 1, \ldots,$$

and assume that this sequence converges to the solution vector $s$. In our case, $F$ is the right-hand side of the SMACOF iteration (given in a general form in Equation (4)). For $x$ close to $s$, $F(x)$ can be expanded in a Taylor series in the form

$$F(x) = F(s) + F'(s)(x - s) + O(\|x - s\|^2) \quad as \quad x \to s.$$
Here $F'(x)$ is the Jacobian matrix of the vector-valued function $F(x)$. Recalling also that $F(s) = s$, this expansion can be put in the form

$$F(x) = s + F'(s)(x - s) + O(\|x - s\|^2) \quad \text{as } x \to s.$$  

By the assumption that the sequence $x_0, x_1, \ldots$, converges to $s$ [which takes place provided $\rho(F'(s)) < 1$], it follows that $x_n$ is close to $s$ for all large $n$, and hence

$$x_{n+1} = s + F'(s)(x_n - s) + O(\|x_n - s\|^2) \quad \text{as } n \to \infty.$$  

Rewriting this in the form

$$x_{n+1} - s = F'(s)(x_n - s) + O(\|x_n - s\|^2) \quad \text{as } n \to \infty,$$

we realize that, for all large $n$, the vectors $x_n$ behave as if they were generated by a linear system of the form $(I - A)x = b$ via

$$x_{n+1} = Ax_n + b, \quad n = 0, 1, \ldots, \quad (21)$$

where $A = F'(s)$ and $b = [I - F'(s)]s$. This suggests that the extrapolation methods MPE and RRE [that were designed by considering vector sequences generated by a linear fixed-point iterative process as in (6)] can be applied to sequences of vectors obtained from nonlinear fixed-point iterative methods. Indeed, methods such as MPE and RRE have been applied with success to the numerical solution of large and sparse nonlinear systems of equations arising in various areas of science and engineering, such as computational fluid dynamics, semiconductor research, and computerized tomography.

### 3.4 Efficient Implementation of MPE and RRE

In subsections 3.1 and 3.2, we gave the definitions of MPE and RRE. These definitions actually form the basis for efficient implementations of MPE and RRE [44]. The most important aspect of these implementations is the accurate solution of the relevant least-squares problems and minimization of computing time and storage requirements. The implementations we give in the sequel, were developed by [44].

In these implementations, the least-squares problems are solved by using QR factorizations of the matrices $U_k^{(n)}$, as in

$$U_k^{(n)} = Q_k R_k.$$  

Here $Q_k$ is an $N \times (k+1)$ unitary matrix satisfying $Q_k^* Q_k = I_{(k+1) \times (k+1)}$. Thus, $Q_k$ has the columnwise partition

$$Q_k = \begin{bmatrix} q_0 & q_1 & \cdots & q_k \end{bmatrix}, \quad (22)$$

such that the columns $q_i$ form an orthonormal set of $N$-dimensional vectors, that is, $q_i^* q_j = \delta_{ij}$. The matrix $R_k$ is a $(k+1) \times (k+1)$ upper triangular matrix.
1. Compute \( r_{00} = \| \mathbf{u}_n \| \) and \( \mathbf{q}_0 = \mathbf{u}_n / r_{00} \)

2. for \( i = 1, \ldots, k \) do

3. Set \( \mathbf{u}_i^{(0)} = \mathbf{u}_{n+i} \) for \( j = 0, \ldots, i-1 \) do

4. \( r_{jk} = \mathbf{q}_j^* \mathbf{u}_i^{(j)} \) and \( \mathbf{u}_i^{(j+1)} = \mathbf{u}_i^{(j)} - r_{jk} \mathbf{q}_j \)

5. end

6. Compute \( r_{ii} = \| \mathbf{u}_i^{(i)} \| \) and \( \mathbf{q}_i = \mathbf{u}_i^{(i)} / r_{ii} \).

7. end

Table 3: The modified Gram-Schmidt algorithm

with positive diagonal elements. Thus,

\[
\mathbf{R}_k = \begin{bmatrix}
  r_{00} & r_{01} & \cdots & r_{0k} \\
  r_{11} & \cdots & r_{1k} \\
  \vdots & \ddots & \vdots \\
  r_{kk} & \cdots & r_{kk}
\end{bmatrix}; \quad r_{ii} > 0, \quad i = 0, 1, \ldots, k.
\] (23)

This factorization can be carried out easily and accurately using the modified Gram–Schmidt orthogonalization process (MGS), which is a standard numerical algorithm [24, 44]. For completeness, Table 3 describes the steps of MGS as applied to the matrix \( \mathbf{U}_k^{(n)} \): 

Here, \( \| \mathbf{x} \| = \sqrt{\mathbf{x}^* \mathbf{x}} \). In addition, the vector \( \mathbf{u}_i^{(j+1)} \) overwrites \( \mathbf{u}_i^{(j)} \), so that the vectors \( \mathbf{u}_{n+i}, \mathbf{u}_i^{(j)} \) and \( \mathbf{q}_i \) all occupy the same storage location.

Note that \( \mathbf{Q}_k \) is obtained from \( \mathbf{Q}_{k-1} \) by appending to the latter the vector \( \mathbf{q}_k \) as the \((k + 1)\)st column. Similarly, \( \mathbf{R}_k \) is obtained from \( \mathbf{R}_{k-1} \) by appending to the latter the \( \mathbf{0} \) vector as the \((k+1)\)st row and then the vector \([r_{0k}, r_{1k}, \ldots, r_{kk}]^T\) as the \((k + 1)\)st column.

An important point we wish to emphasize is that, when forming the matrix \( \mathbf{U}_k^{(n)} \), we overwrite the vector \( \mathbf{x}_{n+i} \) with \( \mathbf{u}_{n+i} = \Delta \mathbf{x}_{n+i} \) as soon as the latter is computed, for \( i = 1, \ldots, k \). We save only \( \mathbf{x}_n \). Next, when computing the matrix \( \mathbf{Q}_k \), we overwrite \( \mathbf{u}_{n+i} \) with \( \mathbf{q}_i, i = 0, 1, \ldots, k \). This means that, at all stages of the computation of \( \mathbf{Q}_k \) and \( \mathbf{R}_k \), we are keeping only \( k + 2 \) vectors in the memory. The vectors \( \mathbf{x}_{n+1}, \ldots, \mathbf{x}_{n+k+1} \) need not be saved.

With the QR factorization of \( \mathbf{U}_k^{(n)} \) (hence of \( \mathbf{U}_{k-1}^{(n)} \)) available we can give algorithms for MPE and RRE within a unified framework as shown in Table 4.

### 3.5 Error Estimation

One common way of assessing the quality of the approximation \( s_{n,k} \) is by looking at the residual vector \( r(s_{n,k}) \) associated with it.

*For linear sequences:* When the iteration vectors \( \mathbf{x}_i \) are generated linearly as in (6), we have
**Input**: $k$ and $n$ and the vectors $x_n, x_{n+1}, \ldots, x_{n+k+1}$.

1. Compute the vectors $u_{n+i} = \Delta x_{n+i}$, $i = 0, 1, \ldots, k$, and form the $N \times (k+1)$ matrix
   
   $$U_k^{(n)} = \begin{bmatrix} u_n & u_{n+1} & \cdots & u_{n+k} \end{bmatrix},$$

   and form its QR factorization, namely, $U_k^{(n)} = Q_k R_k$, with $Q_k$ and $R_k$ as in (22) and (23).

2. **Determination of the $\gamma_i$**:

   // For MPE

   3. With $\rho_k = [r_{0k}, r_{1k}, \ldots, r_{k-1,k}]^T$, solve the $k \times k$ upper triangular system
      
      $$R_{k-1} c' = -\rho_k; \quad c' = [c_0, c_1, \ldots, c_{k-1}]^T.$$  

      Set $c_k = 1$, and $\gamma_i = c_i / \sum_{i=0}^k c_i$, $i = 0, 1, \ldots, k$.

   // For RRE

   4. With $e = [1, 1, \ldots, 1]^T$, solve the $(k+1) \times (k+1)$ linear system
      
      $$R_k^* R_k d = e; \quad d = [d_0, d_1, \ldots, d_k]^T.$$  

      This amounts to solving two triangular systems: First $R_k^* a = e$ for $a$, and, following that, $R_k d = a$ for $d$. Next, compute $\lambda = 1 / \sum_{i=0}^k d_i$; $\lambda$ is always positive (it becomes zero only when $s_{n,k} = s$ in the linear case). Next, set $\gamma = \lambda d$, that is $\gamma_i = \lambda d_i$, $i = 0, 1, \ldots, k$.

   5. With the $\gamma_i$ computed, compute $\xi = [\xi_0, \xi_1, \ldots, \xi_{k-1}]^T$ via
      
      $$\xi_0 = 1 - \gamma_0; \quad \xi_j = \xi_{j-1} - \gamma_j, \quad j = 1, \ldots, k - 1.$$  

   6. Compute
      
      $$\eta = [\eta_0, \eta_1, \ldots, \eta_{k-1}]^T = R_{k-1} \xi.$$  

   7. Then compute
      
      $$s_{n,k} = x_n + Q_{k-1} \eta = x_n + \sum_{i=0}^{k-1} \eta_i q_i.$$  

---

Table 4: An efficient implementation of the MPE/RRE algorithms
Thus, \[ r(x) = b - (I - A)x = (Ax + b) - x. \]

Thus, \[ r(x_n) = x_{n+1} - x_n = u_n. \]

Invoking also \( \sum_{i=0}^{k} \gamma_i = 1 \), where \( \gamma_i \) are as obtained when applying MPE or RRE, we therefore have

\[ r(s_{n,k}) = \sum_{i=0}^{k} \gamma_i u_{n+i} = U_k^{(n)} \gamma. \] (24)

We actually look at the \( l_2 \)-norm of this vector, namely,

\[ \|r(s_{n,k})\| = \|U_k^{(n)} \gamma\|. \]

For **nonlinear sequences**: When the \( x_i \) are generated nonlinearly as in (20), with \( s_{n,k} \) close to \( s \), we have

\[ r(s_{n,k}) = F(s_{n,k}) - s_{n,k} \approx U_k^{(n)} \gamma. \]

Thus,

\[ \|r(s_{n,k})\| \approx \|U_k^{(n)} \gamma\|. \]

Whether the vectors \( x_i \) are generated linearly or nonlinearly, \( \|U_k^{(n)} \gamma\| \) can be computed and in terms of already computed quantities and at no cost, even without having to compute \( s_{n,k} \). Indeed, we have

\[ \|U_k^{(n)} \gamma\| = \begin{cases} r_{kk} |\gamma_k| & \text{for MPE} \\ \sqrt{\lambda} & \text{for RRE} \end{cases}. \]

Here, \( r_{kk} \) is the last diagonal element of the matrix \( R_k \) and \( \lambda \) is the parameter computed in Step 3 of the algorithms in the preceding subsection [44].

### 3.6 Error Analysis for MPE and RRE

The analysis of MPE and RRE, within the context of linear systems, has been carried out in the works [42, 46, 45, 48, 49]. This analysis sheds considerable light on what vector extrapolation methods can achieve when they are applied in conjunction with iterative methods arising from nonlinear systems as well as linear ones.

The following result was given by [42].

**Theorem 1.** Assume that the vectors \( x_n \) satisfy

\[ x_n = s + \sum_{i=1}^{p} v_i \lambda_i^n, \]
where the vectors $v_i$ are linearly independent, and the nonzero scalars $\lambda_i \neq 1$ are distinct, and ordered as in
\[ |\lambda_1| \geq |\lambda_2| \geq \cdots . \]
If also
\[ |\lambda_k| > |\lambda_{k+1}|, \]
then, both for MPE and RRE, there holds
\[ s_{n,k} - s = O(\lambda_{k+1}^n) \quad \text{as } n \to \infty, \]
and
\[ \lim_{n \to \infty} k \sum_{i=0}^{k} \gamma_i^{(n,k)} z^i = \prod_{i=1}^{k} \frac{\lambda - \lambda_i}{1 - \lambda_i}. \]
Here $\gamma_i^{(n,k)}$ stands for $\gamma_i$.

Note that when the $x_n$ are generated by the iterative procedure in (6), they are precisely as described in this theorem, with $\lambda_i$ being some or all of the distinct nonzero eigenvalues of the iteration matrix $A$ and $v_i$ being eigenvectors corresponding to these eigenvalues, when $A$ is diagonalizable.

When the condition $|\lambda_k| > |\lambda_{k+1}|$ does not hold Theorem 1 above needs to be modified somewhat. This has been done previously [45], and we state a special case of it next.

**Theorem 2.** Assume that the vectors $x_n$ have been generated by the iterative procedure $x_{n+1} = Ax_n + b$, the matrix $I - A$ being nonsingular. Denote the solution of $(I - A)x = b$ by $s$. Let us order the nonzero distinct eigenvalues $\lambda_i$ of $A$ as in
\[ |\lambda_1| \geq |\lambda_2| \geq \cdots . \]
Then, whether $|\lambda_k| > |\lambda_{k+1}|$ or not, there holds
\[ s_{n,k} - s = O(\lambda_{k+1}^n) \quad \text{as } n \to \infty, \]
(i) for RRE unconditionally and (ii) for MPE provided the eigenvalues of $I - A$ lie strictly on one side of a straight line through the origin in the complex plane, which happens when $A + A^*$ is positive definite, for example.

In connection with Theorems 1 and 2, we note that we have not assumed that $\lim_{n \to \infty} x_n$ exists. Clearly, $\lim_{n \to \infty} x_n$ exists and is equal to $s$ provided $|\lambda_1| < 1$. In case $\lim_{n \to \infty} x_n$ does not exist, we have that $|\lambda_1| \geq 1$ necessarily.

Now, if we use the $x_n$, as they are, to approximate $s$, we have the error
\[ \epsilon_n = x_n - s = O(\lambda_1^n) \quad \text{as } n \to \infty. \]
Thus, provided $|\lambda_{k+1}| < 1$, we have that $\lim_{n \to \infty} s_{n,k} = s$, whether $\lim_{n \to \infty} x_n$ exists or not. Furthermore, when the sequence $x_n$ converges (to $s$), $s_{n,k}$ converges (to $s$) faster, provided $|\lambda_{k+1}| < |\lambda_1|$. In words, MPE and RRE accelerate the convergence of the sequence $\{x_n\}$.
Choose integers \( n, k \), and an initial vector \( x_0 \).
1. Compute the vectors \( x_i \), \( 1 \leq i \leq n + k + 1 \), [via \( x_{n+1} = F(x_n) \)], and save \( x_{n+i} \), \( 0 \leq i \leq k + 1 \).
2. Apply MPE or RRE to the vectors \( x_{n+i} \), \( 0 \leq i \leq k + 1 \), precisely as described in subsection (3.4), with end result \( s_{n,k} \).
3. If \( s_{n,k} \) satisfies accuracy test, stop.
4. Otherwise, set \( x_0 = s_{n,k} \), and go to Step 1.

**Table 5: Using vector extrapolation techniques in cycling mode**

### 3.7 Cycling with MPE and RRE

The results of Theorems 1 and 2 pertain to the convergence and acceleration of convergence of MPE and RRE. In these theorems, we keep \( k \) fixed and let \( n \) go to infinity. Obviously, there is no way of letting \( n \) go to infinity in practice. It also follows from Theorems 1 and 2 that increasing \( k \) generally makes MPE and RRE converge faster. However, we have no way of increasing \( k \) indefinitely, because this would increase the storage requirements and also increase the computational cost tremendously.

In case we are solving the system \( \mathbf{x} = F(\mathbf{x}) \) and the vectors \( \mathbf{x} \) are generated via the fixed-point iterative procedure \( \mathbf{x}_{n+1} = F(\mathbf{x}_n) \), we can employ a mode of application called cycling or restarting in which \( n \) and \( k \) are held fixed. Here are the steps of this mode. We will call each application of Steps 1–3 a cycle.

We will also denote the \( s_{n,k} \) that is computed in the \( i \)th cycle \( s_{n,k}^{(i)} \).

A discussion of the error in this mode of usage—in case of linear \( F(x) \), i.e., when \( F(x) = A\mathbf{x} + \mathbf{b} \)—was given previously [48, 49]. The relevant errors can be shown to have upper bounds that are expressible in terms of Jacobi polynomials for certain types of spectra of the matrix \( A \), and these bounds turn out to be quite tight. They also indicate that, with even moderate \( n \), \( s_{n,k}^{(i)} \) may be very good approximations to the solution \( s \) with small \( k \), hence small storage and few iterations. Another advantage of applying MPE and RRE in this mode (that is, with \( n > 0 \)) is that it prevents stagnation in the cases where GMRES stagnates. (See the numerical examples in [48, 49]). Numerical experiments confirm that this is indeed the case. Furthermore, this is the case for nonlinear systems of equations as well, even though the analysis of [48, 49] does not apply to this case in a straightforward manner.

The analysis of MPE and RRE as these are applied to nonlinear systems in the cycling mode has been considered in works by [50, 51]. What can be said heuristically is that, when \( k \) in the \( i \)th cycle is chosen to be \( k_i \), the degree of the minimal polynomial of the matrix \( F'(s) \) with respect to \( \epsilon_0 = x_0 - s \), the sequence \( \{s_{n,k}^{(i)}\}_{i=0}^{\infty} \) converges to \( s \) quadratically. However, we must add that, since the \( k_i \) can be as large as \( N \) and are not known exactly, this usage of cycling is not useful practically for the large-scale problems we are interested in solving. In other words, trying to achieve quadratic convergence from MPE and RRE via cycling may not be realistic. With even moderate values of \( n \) and \( k \), we may
achieve linear but fast convergence nevertheless. This turns out to be the case even when \( x_n \) is far from the solution \( s \).

### 3.8 Connection with Krylov Subspace Methods

When applied to linearly generated sequences, MPE and RRE are very closely related to the method of Arnoldi [1] and to GMRES [38], two well known Krylov subspace methods. The following result was stated by [43].

**Theorem 3.** Consider the linear system \((I - A)x = b\). With \( x_0 \) as the initial vector, let the vector sequence \( \{x_n\} \) be generated via \( x_{n+1} = Ax_n + b \), and let \( s_{MPE}^{0,k} \) and \( s_{RRE}^{0,k} \) be obtained from this sequence by applying, respectively, MPE and RRE. Let also the vectors \( s_{Arnoldi}^k \) and \( s_{GMRES}^k \) be the vectors obtained by applying, respectively, \( k \) steps of the method of Arnoldi and GMRES to \((I - A)x = b\), with \( x_0 \) as the starting vector. Then \( s_{MPE}^{0,k} = s_{Arnoldi}^k \) and \( s_{RRE}^{0,k} = s_{GMRES}^k \).

We also recall that the method of Arnoldi becomes the method of conjugate gradients when \( A \) is a Hermitian matrix.

It must be noted that the equivalence of MPE and RRE to the method of Arnoldi and to GMRES is mathematical and not algorithmic. The algorithms (computational procedures) are different.

### 3.9 Cycling the SMACOF algorithm

In order to accelerate the convergence of the SMACOF algorithm, we use cycling mode. Since the problem at hand is nonlinear, there is no guarantee that the approximate limit vector given by extrapolation will result in a lower stress value. We therefore have to use a safeguard. One possibility is to check the stress value at the extrapolated limit, and if the resulting stress is higher, use the last iteration vector instead, as the initial solution to the next cycle. This is simply done by changing step 5 in Table 5. An algorithmic description is shown in Table 6. Another possibility, instead of a simple safeguard, is to use a diminishing step size, halving the step size several times before resorting to the last iteration vector.

### 4 Results

We now show the effect of using vector extrapolation methods for accelerating SMACOF. We note that in general, when a higher degree of accuracy is needed, extrapolation methods tend to work better, since the iterations tend to better approximate a linear iteration scheme, as discussed in Section 3.3.
Choose integers \(n, k\), and an initial vector \(x_0\).

1. Perform \(n\) iterations of the SMACOF algorithm.
2. Perform additional \(k\) iterations of the SMACOF algorithm, saving the iterants as \(x_{n+i}, 0 \leq i \leq k\).
3. Apply MPE or RRE to the vectors \(x_{n+i}, 0 \leq i \leq k+1\), precisely as described in subsection (3.4), with end result \(s_{n,k}\).
4. If \(\text{stress}(x_{n+k}) \leq \text{stress}(s_{n,k})\), take \(s_{n,k}\) to be \(x_{n+k}\).
5. If \(s_{n,k}\) satisfies accuracy test based on relative stress change, stop.
6. Otherwise, set \(x_0 = s_{n,k}\), and go to Step 1.

Table 6: Using vector extrapolation techniques for accelerating the SMACOF algorithm

### 4.1 Positioning of Random Points

As an example we first try to use MDS to reconstruct a random configuration of points set in \(\mathbb{R}^3\), based on their Euclidean distances. Gaussian noise with different variance levels is added to the distance matrix. The points were positioned in a Gaussian distribution, \(x_i \sim N(0, 1)\), their distance matrix is computed, and the points configuration is then reconstructed using SMACOF. 50 point configurations were attempted for each parameters set. The average speed-up in CPU time comparing SMACOF iterations with and without RRE acceleration is shown in Table 7.

<table>
<thead>
<tr>
<th>Size (N)</th>
<th>(\sigma = 0)</th>
<th>(\sigma = 10^{-3})</th>
<th>(\sigma = 10^{-2})</th>
<th>(\sigma = 10^{-1})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N = 400)</td>
<td>2.2642 ± 1.7209</td>
<td>1.6575 ± 0.9526</td>
<td>1.4879 ± 0.5303</td>
<td>1.5284 ± 0.2864</td>
</tr>
<tr>
<td>(N = 800)</td>
<td>1.9447 ± 1.4139</td>
<td>1.9020 ± 0.9072</td>
<td>1.2429 ± 0.1552</td>
<td>1.5742 ± 0.3211</td>
</tr>
<tr>
<td>(N = 1200)</td>
<td>1.9040 ± 1.2879</td>
<td>1.8419 ± 0.9321</td>
<td>1.4568 ± 0.3374</td>
<td>1.4823 ± 0.4370</td>
</tr>
<tr>
<td>(N = 1600)</td>
<td>1.6292 ± 0.8268</td>
<td>1.6577 ± 0.6187</td>
<td>1.5950 ± 0.4348</td>
<td>1.3912 ± 0.2109</td>
</tr>
</tbody>
</table>

Table 7: The resulting speedup, in terms of the rate of decrease in residual norm, obtained using RRE with \(n = 5, k = 5\)

### 4.2 A Euclidean Dimensionality Reduction Example

In this example, previously shown by [10], the degree of the embedding space is much smaller than that of the original space used to generate the distances between points. For example, the points originally reside in \(\mathbb{R}^{500}\). The points are comprised of two sets, distributed according to

\[ x_i = \text{sign}(N(\pm 0.75, 1)) \]

where

\[ \text{sign}(x) = \begin{cases} +1 & x > 0 \\ -1 & x \leq 0 \end{cases} \]

The speedup obtained compared to SMACOF, in terms of CPU time, is shown in Table 8.
Table 8: The resulting speedup, in terms of CPU time, obtained using RRE with $n = 8$, $k = 10$, for the 2-classes Euclidean dimensionality reduction problem

<table>
<thead>
<tr>
<th>Size (N)</th>
<th>Speedup, CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N = 512$</td>
<td>1.6129 ± 0.1041</td>
</tr>
<tr>
<td>$N = 768$</td>
<td>1.6587 ± 0.0973</td>
</tr>
<tr>
<td>$N = 1024$</td>
<td>1.5416 ± 0.1418</td>
</tr>
<tr>
<td>$N = 1536$</td>
<td>1.4781 ± 0.0782</td>
</tr>
<tr>
<td>$N = 2048$</td>
<td>1.4939 ± 0.0915</td>
</tr>
</tbody>
</table>

4.3 Graph Visualization Using Accelerated MDS

We demonstrate the use of RRE-accelerated MDS on the 1138Bus graph distances matrix, taken from the Matrix Market repository [4], and shown as an example for graph visualization based on MDS [23]. The resulting speed-up values are shown, for various $n$ and $k$ parameters in Table 9.

<table>
<thead>
<tr>
<th></th>
<th>$k = 6$</th>
<th>$k = 8$</th>
<th>$k = 10$</th>
<th>$k = 12$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 1$</td>
<td>1.148 ± 0.147</td>
<td>1.212 ± 0.1746</td>
<td>1.234 ± 0.123</td>
<td>1.218 ± 0.126</td>
</tr>
<tr>
<td>$n = 3$</td>
<td>1.303 ± 0.186</td>
<td>1.267 ± 0.149</td>
<td>1.250 ± 0.135</td>
<td>1.212 ± 0.119</td>
</tr>
<tr>
<td>$n = 5$</td>
<td>1.392 ± 0.167</td>
<td>1.320 ± 0.146</td>
<td>1.238 ± 0.154</td>
<td>1.211 ± 0.097</td>
</tr>
</tbody>
</table>

Table 9: The resulting speed-up, in terms of CPU time, obtained using RRE for the 1138Bus graph, using various $n$ and $k$ parameters

4.4 Isometry Invariant Canonical Forms

Another application of MDS we exemplify here is the representation of non-rigid shapes invariant to deformations. This method, referred to as canonical forms was proposed in [20]. In this application, MDS is used to map the intrinsic geometry of the shape (captured by the matrix of pairwise geodesic distances) into a Euclidean space. This produces a representation invariant to isometric deformations of the surface, and which can be treated as a rigid shape.

The canonical forms method was used in the problem of face recognition by [9]. An example of a facial surface and its canonical form is shown in Figure 2. In order to compare our work to existing acceleration techniques for SMACOF, we show in Figure 3 the resulting stress values, as a function of CPU time, when using RRE and multigrid [10] to accelerate the computation of canonical forms.

Both the RRE and the multigrid methods seem to accelerate MDS significantly for the canonical forms problem, by a factor 4-5 for the specific example at hand. In practice, the speedup gained may vary significantly, based on the problem.
Figure 1: The visual results using SMACOF and RRE-accelerated SMACOF, obtained for the 1138Bus graph.
4.5 TCIE using Accelerated MDS

We now demonstrate the use of accelerated MDS in a nonlinear dimensionality reduction (NLDR) application, as part of the topologically constrained isometric embedding (TCIE) algorithm [36]. Nonlinear dimensionality reduction techniques [40, 52, 37, 3, 56, 14] attempt to describe a given high-dimensional data set of points as a low dimensional manifold, by a nonlinear map preserving certain properties of the data.

These techniques have applications in several fields, such as color perception, pathology tissue analysis [14], motion understanding [34], enhancement of MRI images [18], face recognition [9], and biochemistry [28, 16].

In the TCIE algorithm, a weighted MDS problem is solved, using distances obtained by using local Euclidean distances, and their extension by means of dynamic programming. This approximates the geodesics connecting points in the manifold the data is sampled from. The weights allow the algorithm to avoid approximating geodesics which should not map to a straight line in the new representation [36]. Such geodesics, and their end point pairs, are called inconsistent. An example of such a mapping and an inconsistent point pair is shown in Figure 4.

We have used RRE to accelerate the convergence of the MDS algorithm used in the TCIE algorithm [36]. We used a Swiss roll with a hole in it as an example. The original data, as well as the data after reduction to $\mathbb{R}^2$ are shown in Figure 5. The RRE accelerated MDS, combined with a multiscale optimization framework, to compute the coordinates which minimize the stress function. The resulting speedups, were at least 2-3 times compared to the ordinary SMACOF algorithm computed in a multiscale manner. A graph comparing the progress of the TCIE algorithm with and without using reduced rank extrapolation, and multiscale optimization is shown in Figure 6.

Figure 2: Left: A facial surface used as input to the canonical forms algorithm. Right: The resulting canonical form
Figure 3: Stress, as a function of CPU time for the canonical forms problem. CPU time is approximated. The second sub-figure is a close-up of the first sub-figure.
Figure 4: Example of inconsistent $x_1$ and $x_2$, for which the line connecting them in $\mathbb{R}^m$ (dashed line) is shorter than the geodesic connecting $x_1$ and $x_2$ (solid curve).

Figure 5: Left: Swiss roll surface with a hole in it. Detected boundary points are shown in red. Right: The mapping obtained using the TCIE algorithm.
Figure 6: Convergence (in terms of stress value) of basic SMACOF (top, dashed gray), SMACOF with RRE (top, black), SMACOF with multiresolution acceleration (bottom, dashed gray), and SMACOF with both RRE and multiscale (bottom, black), in terms of CPU time, as part of the TCIE algorithm. CPU time is approximated. Convergence at each scale was stopped at the same relative change of stress value.
5 Conclusions

We presented a way of accelerating the computation of a solution to the MDS problem using the SMACOF algorithm, using vector extrapolation techniques. These are a family of generic methods for accelerating the convergence of iterative algorithms, shown in this paper to be quite effective in the case of SMACOF. While dependent on the specific nonlinear problem at hand, and while there are no global convergence bounds for this nonlinear problem, we demonstrated these methods work well in the case of several typical applications of MDS, making them an attractive options for accelerating the solution of MDS and related problems.

We intend to further study the applications of accelerated MDS, and try to determine cases where vector extrapolation is particularly effective. We would like to combine this method with a multigrid solution to the MDS problem [10], similar in spirit to [7].

Acknowledgements

This research was partly supported by United States -Israel Binational Science Foundation grant No. 2004274 and by the Ministry of Science grant No. 3-3414, and in part by Elias Fund for Medical Research.

References


