POLYNOMIAL EXTRAPOLATION TO ACCELERATE FIXED POINT ITERATIONS

JAN DE LEEUW

ABSTRACT. The paper discusses iterative methods for linear systems and various ways to accelerate their convergence. The results are then applied to non-linear fixed point iterations, in particular to multidimensional scaling. The purpose of the paper is didactic and it contains absolutely nothing that is original. In fact, it closely follows Sidi [2008]. We do provide code in R.

1. INTRODUCTION

In this paper we are interested in accelerating the convergence of the general non-linear fixed point iteration

$$x^{(k+1)} = F(x^{(k)}). \tag{1}$$

The iteration (1) can be linearized in the neighborhood of a fixed point $x_\star$. This gives

$$x^{(k+1)} = F(x^{(k)}) = F(x_\star + (x^{(k)} - x_\star)) \approx x_\star + F'(x_\star)(x^{(k)} - x_\star),$$

which can be written as

$$x^{(k+1)} \approx F'(x_\star)x^{(k)} + (I - F'(x_\star))x_\star.$$

This shows that in the neighborhood of a fixed point the non-linear iteration behaves approximately like the linear iteration

$$x^{(k+1)} = Tx^{(k)} + b. \tag{2}$$

This suggest that if we have methods to accelerate (2), then these can be extended at least locally to the general non-linear iteration (1). So we start by studying the linear iterations 2.

Date: July 18, 2008 — 13h 53min — Typeset in TIMES ROMAN.

2. MATRICES

Suppose T is a real $n \times n$ matrix. We suppose T is *semi-simple*, i.e. there is an $n \times n$ real nonsingular matrix S and a real diagonal matrix Λ such that $T = S\Lambda S^{-1}$. The elements of Λ are the *eigenvalues* of T, while the columns of S are the *eigenvectors*. Thus a semi-simple matrix has n linearly independent eigenvectors.

The results in this note can be extended, with some effort, to general matrices that are not semi-simple [Sidi and Bridger, 1988]. In that case we have to use the Jordan or Schur canonical form. Extending the results to semi-simple matrices with complex eigenvalues and eigenvectors is easy.

We create an example by using the code in B.1.

```
> set.seed(12345)
> print(s<-matrix(rnorm(16),4,4))
            [,1]       [,2]       [,3]       [,4]
[1,]  0.5855288  0.6058875 -0.2841597  0.3706279
[2,]  0.7094660 -1.8179560 -0.9193220  0.5202165
[3,] -0.1093033  0.6300986 -0.1162478 -0.7505320
[4,] -0.4534972 -0.2761841  1.8173120  0.8168998
> print(t<-s%*%diag(c(.9,.5,.5,.1))%*%solve(s))
             [,1]         [,2]       [,3]       [,4]
[1,]  0.84056482  0.413111525  1.0076204  0.3266863
[2,]  0.41141976  1.021384862  1.2886200  0.4105122
[3,] -0.05178042 -0.276632049 -0.3366922 -0.2015567
[4,] -0.28288032  0.003306532  0.2704800  0.4747425
> print(eval<-eigen(t)$values)
[1] 0.9 0.5 0.5 0.1
```

As the example shows, the n eigenvalues λ_i are not necessarily distinct. Suppose there are $d(T) \leq n$ distinct eigenvalues, which we will write as $\underline{\lambda}_1 < \underline{\lambda}_2 < \cdots < \underline{\lambda}_{d(T)}$. Each $\underline{\lambda}_s$ has a multiplicity n_s, with the n_s adding up to n.

The *characteristic polynomial* of T is the monic polynomial [1] of degree n

$$\pi_T(z) = \prod_{i=1}^{n}(x - \lambda_i) = \prod_{s=1}^{d(T)}(z - \underline{\lambda}_s)^{n_s}.$$

[1] A monic polynomial has its leading coefficient equal to one.

The *minimal polynomial* of T is the monic polynomial of degree $d(T)$ defined by

$$\underline{\pi}_T(z) = \prod_{s=1}^{d(T)} (z - \underline{\lambda}_s).$$

Clearly the minimal polynomial divides the characteristic polynomial. Also $\pi_T(\lambda_i) = \underline{\pi}_T(\lambda_i) = 0$ for all i. The characteristic and minimal polynomials of our example can be computed by the code B.2.

```
> library(polynom)
> p1<-polynomial(c(-.9,1))
> p2<-polynomial(c(-.5,1))
> p3<-polynomial(c(-.5,1))
> p4<-polynomial(c(-.1,1))
> print(cpol<-p1*p2*p3*p4)
0.0225 - 0.34*x + 1.34*x^2 - 2*x^3 + x^4
> print(mpol<-p1*p2*p4)
-0.045 + 0.59*x - 1.5*x^2 + x^3
```

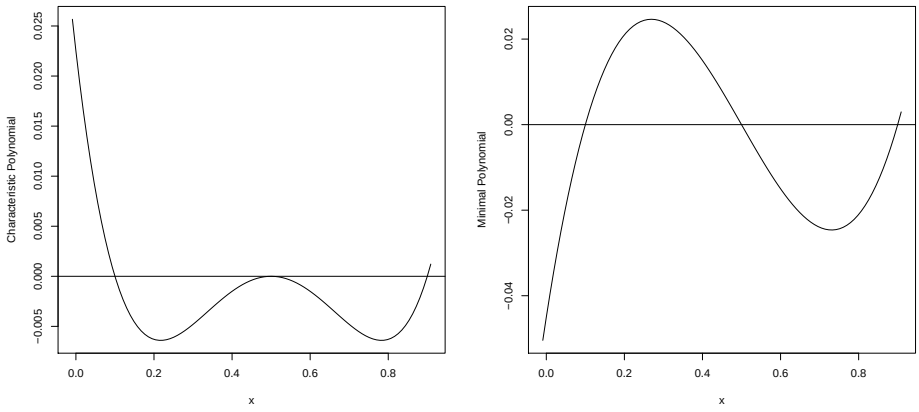

FIGURE 1. Characteristic and Minimal Polynomial

We define matrix functions from scalar functions as

$$f(T) = Sf(\Lambda)S^{-1},$$

where $f(\Lambda)$ is diagonal with elements $f(\lambda_i)$. Thus if $\pi(z) = c_0 + c_1 x + \cdots + c_r z^r$ then $\pi(T) = c_0 I + c_1 T + \cdots + c_r T^r$. The Cayley-Hamilton Theorem says that $\pi(T) = \underline{\pi}(T) = 0$. This follows easily from $\pi(T) = S\pi(\Lambda)S^{-1}$ and $\pi(\Lambda) = 0$.

3. EQUATIONS

Now b is an n vector and suppose $x = Tx + b$ or, equivalently, $(I - T)x = b$ is solvable. If $\tilde{x} = S^{-1}x$ and $\tilde{b} = S^{-1}b$ then $(I - T)x = b$ is solvable if and only if $(I - \Lambda)\tilde{x} = \tilde{b}$ is solvable. And this last system is solvable if and only if for all i with $\lambda_i = 1$ we have $\tilde{b}_i = 0$. The general solution is

$$x_\star = \sum_{i=1}^{n} \theta_i s_i,$$

with

$$\theta_i = \begin{cases} \frac{\tilde{b}_i}{1-\lambda_i} \text{ if } \lambda_i \neq 1, \\ \text{arbitrary if } \lambda_i = 1 \text{ and } \tilde{b}_i = 0. \end{cases}$$

The solution is unique if and only if $\lambda_i \neq 1$ for all i. In our example, using the code in B.3, we see

```
> print(b<-rnorm(4))
[1]  0.7796219  1.4557851 -0.6443284 -1.5531374
> print(xstar<-solve(diag(4)-t,b))
[1] -0.80218719 -0.03206138 -0.05898958 -2.55546279
```

4. ITERATIONS

Start the iterations with $x^{(0)} \in \mathbb{R}^n$, and define

$$x^{(k+1)} = Tx^{(k)} + b. \tag{3}$$

Assume that all $\lambda_i \neq 1$. Suppose $x_\star$ is the unique solution of $x = Tx + b$, and define $e^{(k)} \triangleq x^{(k)} - x_\star$. Then $e^{(k+1)} = Te^{(k)}$ and thus $e^{(k)} = T^k e^{(0)}$. Let $\tilde{e}^{(k)} = S^{-1}e^{(k)} = \Lambda^k \tilde{e}^{(0)}$. Thus $x^{(k)} \to x_\star$ if and only if $\tilde{e}^{(k)} \to 0$, which happens for any $e^{(0)}$ if and only if $\Lambda^k \to 0$, i.e. if and only if $\max_{i=1}^{n} |\lambda_i| = \max_{s=1}^{d(T)} |\underline{\lambda}_s| < 1$. The iteration (3) converges globally (from any starting point) if and only if the *spectral radius*, i.e. the largest eigenvalue in modulus, satisfies $\|T\|_\infty < 1$.

For convergent sequences it is fairly simply to study convergence speed. Note that $\|e^{(k)}\|^2 = \{\tilde{e}^{(0)}\}'\Lambda^{2k}\tilde{e}^{(0)} = \Sigma_{s=1}^{d(T)} \underline{\lambda}_s^{2k} \sigma_s$, where the σ_s are the sum of squares of the elements of $\tilde{e}^{(0)}$ corresponding to the n_s eigenvalues equal to $\underline{\lambda}_s$. Suppose, without loss of generality, that $0 < \|T\|_\infty = |\underline{\lambda}_1| < 1$. Then, if $\sigma_1 > 0$,

$$\|e^{(k)}\|^2 = \sigma_1 \underline{\lambda}_1^{2k} \left\{ 1 + \sum_{s=2}^{d(T)} \left(\frac{\underline{\lambda}_s}{\underline{\lambda}_1} \right)^{2k} \frac{\sigma_s}{\sigma_1} \right\}$$

and thus

$$\frac{\|e^{(k+1)}\|}{\|e^{(k)}\|} \approx |\underline{\lambda}_1| \left\{ 1 + \frac{1}{2} \sum_{s=2}^{d(T)} \left(\frac{\underline{\lambda}_s}{\underline{\lambda}_1} \right)^{2(k+1)} \frac{\sigma_s}{\sigma_1} \right\} \left\{ 1 - \frac{1}{2} \sum_{s=2}^{d(T)} \left(\frac{\underline{\lambda}_s}{\underline{\lambda}_1} \right)^{2k} \frac{\sigma_s}{\sigma_1} \right\} \approx$$
$$\approx |\underline{\lambda}_1| \left(1 + \frac{1}{2} \sum_{s=2}^{d(T)} \left[\left(\frac{\underline{\lambda}_s}{\underline{\lambda}_1} \right)^{2(k+1)} - \left(\frac{\underline{\lambda}_s}{\underline{\lambda}_1} \right)^{2k} \right] \frac{\sigma_s}{\sigma_1} \right) \approx |\underline{\lambda}_1|. \tag{4}$$

If we happen to stumble onto an initial estimate for which $\sigma_1 = 0$ then (at least in exact arithmetic) the next largest eigenvalue takes over. The code in B.4 does 100 iterations, and then plots of the elements of $e^{(k)}$ and the ratio in (4).

Note that the expansion in (4) of the ratio of successive errors remains valid if $|\underline{\lambda}_1| = 1$. In that case $\tilde{e}^{(k)}$ converges (in exact arithmetic) to the vector $\tilde{e}^\star$ with elements

$$\tilde{e}_i^\star = \lim_{k\to\infty} \tilde{e}_i^{(k)} = \begin{cases} 0 & \text{if } \lambda_i < 1, \\ \tilde{e}_i^{(0)} & \text{if } \lambda_i = 1, \end{cases}$$

and thus x^k converges to $x_\star + S\tilde{e}^\star$.

5. Solution from Iterations

Define *the minimal polynomial of T with respect to a vector y*. It is the monic polynomial π of lowest degree for which $\pi(T)y = 0$. Again using $\tilde{y} = S^{-1}y$ we see that we must have $\pi(\Lambda)\tilde{y} = 0$. This means that

$$\underline{\pi}_{T,y}(z) = \prod_{s\in I(y)} (z - \underline{\lambda}_s),$$

where $I(y)$ is the index set for which $\tau_s > 0$, and where the τ_s are the sum of squares of the elements of $\tilde{y}$ corresponding with the n_s eigenvalues equal to $\underline{\lambda}_s$.

Thus the minimal polynomial of T with repect to y is unique and divides the minimal polynomial of T. Its degree is less than or equal to $d(T)$. If π is another monic polynomial for which $\pi(T)y = 0$ then $\mathbf{deg}(\pi) > \mathbf{deg}(\underline{\pi}_{T,y})$ and $\underline{\pi}_{T,y}$ divides π.

Now consider the minimal polynomial of T with respect to $e^{(0)} = x^{(0)} - x_\star$, and suppose it has degree $d(T, e^{(0)})$. Suppose $\underline{\pi}_{T,e^{(0)}}(z) = \sum_{s=0}^{d(T,e^{(0)})} c_s z^s$, with $c_{d(T,e^{(0)})} = 1$ and $d(T, e^{(0)}) \le d(T)$. By definition $\underline{\pi}_{T,e^{(0)}}(T)e^{(0)} = 0$, which means

$$\sum_{s=0}^{d(T,e^{(0)})} c_s T^s (x^{(0)} - x_\star) = \sum_{s=0}^{d(T,e^{(0)})} c_s (x^{(s)} - x_\star) = 0,$$

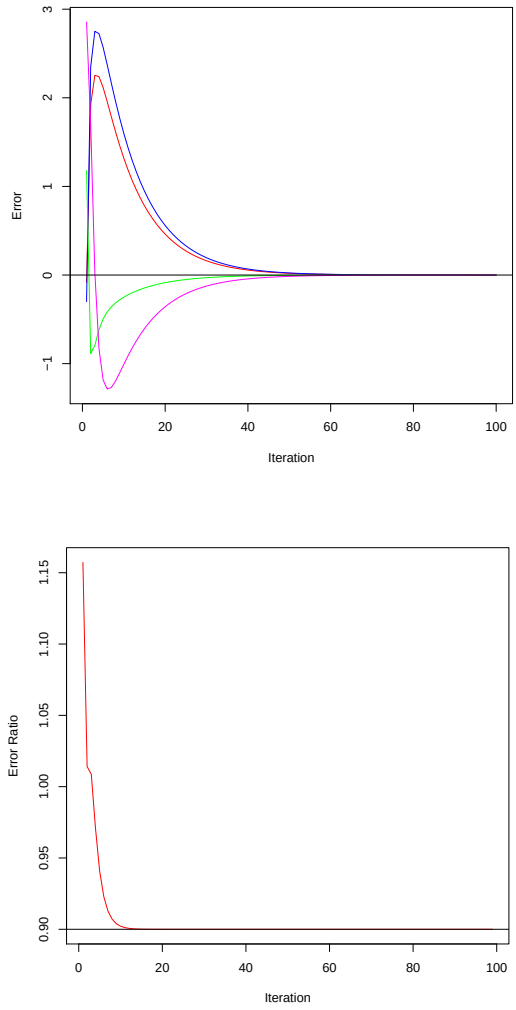

FIGURE 2. Convergence and Convergence Speed

or

$$x_\star = \frac{\sum_{s=0}^{d(T,e^{(0)})} c_s x^{(s)}}{\sum_{s=0}^{d(T,e^{(0)})} c_s}. \tag{5}$$

Note $\sum_{s=0}^{d(T,e^{(0)})} c_s = \underline{\pi}_{T,e^{(0)}}(1) \neq 0$, because $\underline{\pi}_{T,e^{(0)}}$ divides the characteristic polynomial π_T, and T does not have an eigenvalue equal to one. Result (5) shows we can find $x_\star$ by a suitable weighted average of the iterates $x^{(k)}$, provided we know $\underline{\pi}_{T,e^{(0)}}$.

Let us illustrate and verify (5) for our example by the chunk in B.5. Remember that the degree of the minimal polynomial $\underline{\pi}(T)$ is 3, so we use 4 iterations.

```
> print(c<-as.vector(mpol))
[1] -0.045  0.590 -1.500  1.000
> print(sol<-colSums(c*xx[1:4,])/sum(c))
[1] -0.80218719 -0.03206138 -0.05898958 -2.55546279
> print(xstar)
[1] -0.80218719 -0.03206138 -0.05898958 -2.55546279
```

Since $\underline{\pi}_{T,e^{(0)}}$ depends on $x_\star$, the result (5) does not seem to be very useful. By a simple trick, however, we can actually make it ready for computation. Define

$$u^{(k)} \triangleq \Delta x^{(k)} = x^{(k+1)} - x^{(k)},$$

and now consider $\underline{\pi}_{T,u^{(0)}}$. Note that $u^{(k)} = T^k u^{(0)}$, and thus the speed of convergence of $u^{(k)}$ to zero is equal to that of $e^{(k)}$, i.e. it is $|\underline{\lambda}_1|$.

It can (and will) be shown that the polynomials $\underline{\pi}_{T,u^{(0)}}$ and $\underline{\pi}_{T,e^{(0)}}$ are identical. Assuming this for the moment, it follows that

$$\sum_{s=0}^{d(T,u^{(0)})} c_s T^s u^{(0)} = \sum_{s=0}^{d(T,u^{(0)})} c_s u^{(s)} = 0. \tag{6}$$

But this implies

$$Uc = -u^{d(T,u^{(0)})}, \tag{7}$$

where

$$U = \left[u^{(0)} \;\middle|\; u^{(1)} \;\middle|\; \cdots \;\middle|\; u^{(d(T,u^{(0)})-1)} \right].$$

This means we can solve (7) for c, and then use (5) to compute $x_\star$. See code chunk B.6. Note that U is 4×3 and we use the Moore-Penrose inverse to solve (7).

```
> print(ulft<-t(uu[1:3,]))
           [,1]        [,2]        [,3]
[1,]  2.010802  0.32653082 -0.01391910
[2,]  2.650832  0.39877135 -0.02571096
[3,] -2.064965  0.09108946  0.17830499
[4,] -1.157277 -1.66799165 -0.85827927
> print(urgt<--uu[4,])
[1]  0.1230457  0.1545541 -0.1207913  0.3553813
> print(uinv<-ginv(ulft))
```

```
             [,1]       [,2]       [,3]       [,4]
[1,] -0.4435363 -0.3552426  -1.238054 -0.2393672
[2,]  3.7458212  3.4252807   7.287715  1.3506446
[3,] -6.6909106 -6.1698379 -12.492658 -3.4670993
> print(cc<-c(drop(uinv%*%urgt),1))
[1] -0.045  0.590 -1.500  1.000
```

It remains to be shown that $\underline{\pi}_{T,u^{(0)}} \equiv \underline{\pi}_{T,e^{(0)}}$. We follow Sidi [2008, Theorem 3.3]. Because $\underline{\pi}_{T,e^{(0)}} e^{(0)} = 0$ and $u^{(k)} = (T-I)e^{(k)}$ we have

$$0 = (T-I)\underline{\pi}_{T,e^{(0)}} e^{(0)} = \underline{\pi}_{T,e^{(0)}}(T-I)e^{(0)} = \underline{\pi}_{T,e^{(0)}} u^{(0)}.$$

This uses the fact that any two polynomials in T commute. Thus $\underline{\pi}_{T,u^{(0)}}$ divides $\underline{\pi}_{T,e^{(0)}}$. Because $\underline{\pi}_{T,u^{(0)}} u^{(0)} = 0$ we have

$$0 = \underline{\pi}_{T,u^{(0)}} u^{(0)} = \underline{\pi}_{T,u^{(0)}}(I-T)e^{(0)} = (T-I)\underline{\pi}_{T,u^{(0)}} e^{(0)}$$

Since $T-I$ is non-singular we have $\underline{\pi}_{T,u^{(0)}} e^{(0)} = 0$, which implies that $\underline{\pi}_{T,e^{(0)}}$ divides $\underline{\pi}_{T,u^{(0)}}$. Because both polynomials are monic, they are identical.

Thus this section shows that we can find the solution of the linear system from the first $d(T,u^{(0)})+2$ iterations $x^{(0)}, x^{(1)}, \cdots, x^{(d(T,u^{(0)})+1)}$.

6. CYCLIC MPE/RRE

In the previous section we assumed that $d(T,u^{(0)})$ was known. In most practical situations r will be equal to n, the order of the matrix (and the degree of the characteristic polynomial). Let us consider the system $Uc = 0$, or more explictly

$$\begin{bmatrix} u^{(0)} & | & u^{(1)} & | & u^{(r)} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_r \end{bmatrix} = 0. \tag{8}$$

If $r = d(T,u^{(0)})$ then the unique vector in the null=space of U will be proportional to the coefficients of the minimal polynomial. Taking r too large leads to linear dependencies in (8), but since we use Moore-Penrose inverses (least squares solutions) our results will still be true. We are just doing too much work.

If r is too small, then (8) will be an over-determined system, which generally does not have a solution. But we can still compute some approximate solution, which

then serves as an approximation to $x_\star$. This leads to the general idea of choosing an r which is not too big and not too small, computing $x^{(0)}, x^{(1)}, \cdots, x^{(r+1)}$, and solve (8) by least squares with some identification condition to find a unique c, and then use (5) to compute an approximation to $x_\star$. Chossing a relatively small r is the most interesting case from the practical point of view.

Once we have done such a cycle to find a new approximation to $x_\star$ we can start a new cycle, using the current approximation of $x_\star$ for the new $x^{(0)}$. We then repeat cycles until convergence.

6.1. **MPE.** Minimal Polynomial Extrapolation (MPE) was first discussed by Cabay and Jackson [1976]. In MPE we identify the solution to (8) by requiring $c_r = 1$. This means that if $r = d(T, u^{(0)})$ we will indeed find the coefficients of the minimal polynomial. But if $r < d(T, u^{(0)})$ we minimize the sum of squares

$$\left\| \begin{bmatrix} u^{(0)} & | & u^{(1)} & | & \cdots & | & u^{(r-1)} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{r-1} \end{bmatrix} + u^{(r)} \right\|^2,$$

and then set

$$x_\star \approx \frac{x^{(r)} + \sum_{s=0}^{r-1} c_s x^{(s)}}{1 + \sum_{s=0}^{r-1} c_s}$$

In Appendix C we see an implementation of MPE in `R`. Actually both MPE and RRE are implemented as options of the single function `sidi()`. If $r = 0$ we use the simple iteration $x^{(k+1)} = Tx^{(k)} + b$. If $r \geq 1$ we solve a least squares system with r predictors to find the $r+1$ elements of c, which are then used to compute the weighted mean approximating $x_\star$.

6.2. **RRE.** Reduced Rank Extrapolation (RRE), introduced independently by Mešina [1977] and Eddy [1979], is another way of tackling the system (8). We minimize the sum of squares $c'U'Uc$, requiring that $c'e = 1$, where e is a vector will all elements equal to one. The solution is simply

$$c = \frac{1}{e'(U'U)^{-1}e}(U'U)^{-1}e.$$

The RRE option in the `sidi()` program in Appendix C does not actually compute the cross-product $U'U$, however, because that is computationally wasteful and will

introduce ill-conditioning if r is large. Instead, we minimize the sum of squares

$$\left\|\begin{bmatrix} u^{(0)}-u^{(r)} & | & u^{(1)}-u^{(r)} & | & \cdots & | & u^{(r-1)}-u^{(r)}\end{bmatrix}\begin{bmatrix} c_0\\ c_1\\ \vdots\\ c_{r-1}\end{bmatrix}+u^{(r)}\right\|^2,$$

and then set

$$c_r=1-\sum_{s=0}^{r-1}c_s,$$

and

$$x_\star\approx\sum_{s=0}^{r}c_s x^{(s)}=x^{(r)}+\sum_{s=1}^{r-1}c_s(x^{(s)}-x^{(r)}).$$

6.3. **Stabilization.** It has been suggested, for example by Roland et al. [2007], to stabilize polynomial extrapolation algorithms by starting the new cycle not with the extrapolated $x_\star$, but by performing another basic iteration first. Thus we use $x^{(i,0)},\cdots,x^{(i,k+1)}$ to compute $x_\star^{(i)}$ by MPE or RRE, and then we set $x^{(i+1,0)}=F(x_\star^{(i)})$. This has been observed to improve both global and local convergence at relatively little extra cost.

7. EXAMPLES

7.1. **Small Example.** The small example we have used throughout the paper has a matrix T of order 4, with $d(T)=3$. With a random start 222 linear iterations are needed to get an error of less than 1E-10.

In Table 1 we give the number of cycles for running the example in B.7. We run MME and RRE with and without stabilization. Both converge in one iteration for $r=3$. How MPE and RRE behave for different values of r depends very much on the starting point. There are cases in which either MPE or RRE does not converge (indicated by F in the tables), or converges to a vector which is not the solution of the original system.

Because the example is so small almost all time is overhead, not computation, so it does not make sense to compare running times. Note that the number of cycles is a somewhat misleading performance indicator, because MPE/F and RRE/F compute the basic iteration map $r+1$ times in each cycle, while MPE/T and RRE/T compute it $r+2$ times. Thus the number of cycles makes RRE and MPE with larger r

look too good. Table 1 we also give the number of function evaluations. Because this does not take overhead corresponding with cycling into account, this actually makes RRE and MPE with larger r look too bad.

Insert Table 1 *about here*

7.2. **Positive Definite Example.** A larger, and better behaved, example with a symmetric positive definite T of order 10 is in code chunk B.8. MPE and RRE are indistinguishable, both in number of cycles used and number of fucntion evluations used. Clearly at some point it is not useful any more to increase r, and in this example $r = 1$ is not very good either. Table 2 gives the number of cycles and the number of function evaluations for different values of r.

Insert Table 2 *about here*

7.3. **Nonlinear Example.** Finally, we apply this to the multidimensional scaling (MDS) example also studied in Rosman et al. [2008] and De Leeuw [2008]. The code to generate the data is in code chunk B.9 and the code for the MDS computations in C.2.

For an MDS example with 10 parameters and 15 dissimilarities, starting at x equal to $1,2,\cdots,10$, we find the results in Table 3. There is little difference between MPE and RRE if they both work, but generally RRE seems to be slightly more fragile (in the sense that it does not converge or converges to the wrong solution). We also give empirical estimates of the convergence rate (the ratio of the norm of the change of the solution over a cycle) in Table 4 and the time the algorithm takes (using elapsed time from `system-time()`) in Table 5. We see that in this example for $r = 3$ or $r = 4$ MPE is about 8-10 times as fast as the basic SMACOF algorithm. Of course this is highly implementation dependent and the figures may come out differently if we use compiled code. It is unclear as yet if the failures of RRE are bugs in the code, occur because of loss of precision, or are cases of sublinear convergence to a saddle point.

Insert Table 3 *about here*

Insert Table 4 *about here*

Insert Table 5 *about here*

REFERENCES

S. Cabay and L.W. Jackson. A Polynomial Extrapolation Methods for Finding Limits and Antilimits of Vector Sequences. *SIAM Journal Numerical Analysis*, 13:734–752, 1976.

J. De Leeuw. Accelerated Least Squares Multidimensional Scaling. Preprint 493, Department of Statistics, UCLA, 2008. URL `http://idisk.mac.com/jdeleeuw-Public/algorithms/rate/rate.pdf`.

R.P. Eddy. Extrapolating to the Limit of a Vector Sequence. In P.C.C. Wang, editor, *Information Linkage between Applied Mathematics and Industry*, pages 387–396, New York, N.Y., 1979. Academic Press.

M. Mešina. Convergence Acceleration for the Iterative Solution of the equations X=AX+f. *Computational Methods in Applied Mechanics and Engineering*, 10: 165–173, 1977.

Ch. Roland, R. Varadhan, and C.E. Frangakis. Squared Polynomial Extrapolation Methods with Cycling: an Application to the Positron Emission Tomography Problem. *Numerical Algorithms*, 44:159–172, 2007.

G. Rosman, A.M. Bronstein, M.M. Bronstein, A. Sidi, and R. Kimmel. Fast Multidimensional Scaling using Vector Extrapolation. Technical Report CIS-2008-01, Computer Science Department, Technion, Haifa, Israel, 2008. URL `http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi/2008/CIS/CIS-2008-01`.

A. Sidi. Vector Extrapolation Methods with Applications to Solution of Large Systems of Equations and to PageRank Computations. *Computers and Mathematics with Applications*, 56:1–24, 2008.

A. Sidi and J. Bridger. Convergence and Stability Analysis for Some Vector Extrapolation Methods in the Presence of Defective Iteration Matrices. *Journal of Computational and Applied Mathematics*, 22:35–61, 1988.

APPENDIX A. TABLES

TABLE 1. Small Example

r	MPE/F	MPE/T	RRE/F	RRE/T
0	214 (214)	108 (216)	214 (214)	108 (216)
1	108 (206)	47 (141)	F	20 (60)
2	39 (117)	32 (128)	F	32 (128)
3	1 (4)	1 (5)	1 (4)	1 (5)

TABLE 2. Positive Definite Example

r	MPE/F	MPE/T	RRE/F	RRE/T
0	94(94)	48(96)	94(94)	48(96)
1	52(104)	20(60)	52(104)	20(60)
2	18(54)	11(44)	18(54)	11(44)
3	11(44)	8(40)	11(44)	8(40)
4	7(35)	6(36)	7(35)	6(36)
5	6(36)	5(35)	6(36)	5(35)

TABLE 3. Multidimensional Scaling Example

r	MPE/F	MPE/T	RRE/F	RRE/T
0	191(191)	96(192)	191(191)	96(192)
1	63(126)	21(63)	F	F
2	21(63)	11(44)	F	F
3	9(36)	7(35)	10(40)	7(35)
4	6(30)	5(30)	7(35)	5(30)
5	5(30)	4(28)	5(30)	5(35)

TABLE 4. Multidimensional Scaling Rate

r	MPE/F	MPE/T	RRE/F	RRE/T
0	0.9031	0.8157	0.9430	0.8892
1	0.4104	0.3543	F	0.1701
2	0.3777	0.0903	F	F
3	0.0707	0.0212	0.1781	F
4	0.0114	0.0035	F	0.0178
5	0.0037	0.0002	F	0.0060

TABLE 5. Multidimensional Scaling Time

r	MPE/F	MPE/T	RRE/F	RRE/T
0	0.680	0.343	0.687	0.346
1	0.257	0.104	F	F
2	0.100	0.063	F	0.066
3	0.072	0.048	0.076	F
4	0.084	0.047	F	0.077
5	0.071	0.040	F	0.093

APPENDIX B. CODE CHUNKS

B.1. **Example Chunk.**

```
set.seed(12345)
print(s<-matrix(rnorm(16),4,4))
print(t<-s%*%diag(c(.9,.5,.5,.1))%*%solve(s))
print(eval<-eigen(t)$values)
```

B.2. **Polynom Chunk.**

```
library(polynom)
p1<-polynomial(c(-.9,1))
p2<-polynomial(c(-.5,1))
p3<-polynomial(c(-.5,1))
p4<-polynomial(c(-.1,1))
print(cpol<-p1*p2*p3*p4)
print(mpol<-p1*p2*p4)
pdf("cpol.pdf")
plot(cpol,ylab="Characteristic Polynomial")
abline(h=0)
dev.off()
pdf("mpol.pdf")
plot(mpol,ylab="Minimal Polynomial")
abline(h=0)
dev.off()
```

B.3. **Equation Chunk.**

```
print(b<-rnorm(4))
print(xstar<-solve(diag(4)-t,b))
```

B.4. **Iterations Chunk.**

```
print(x0<-rnorm(4))
xx<-matrix(0,100,4)
ee<-matrix(0,100,4)
xx[1,]<-x0
for (i in 2:100) xx[i,]<-drop(t%*%xx[i-1,])+b
```

```
ee<-t(t(xx)-xstar)
eup<-max(ee); elw<-min(ee)
pdf("error.pdf")
plot(1:100,ee[,1],type="l",col="RED",xlab="Iteration",
    ylab="Error",ylim=c(elw,eup))
lines(1:100,ee[,2],type="l",col="BLUE")
lines(1:100,ee[,3],type="l",col="GREEN")
lines(1:100,ee[,4],type="l",col="MAGENTA")
abline(h=0)
dev.off()
enorm<-sqrt(rowSums(ee^2))
eratio<-enorm[2:100]/enorm[1:99]
pdf("eratio.pdf")
plot(1:99,eratio,type="l",col="RED",xlab="Iteration",ylab
    ="Error Ratio")
abline(h=.9)
dev.off()
```

B.5. **Iteration Solution Chunk.**

```
print(cc<-as.vector(mpol))
print(sol<-colSums(cc*xx[1:4,])/sum(cc))
print(xstar)
```

B.6. **Change Solution Chunk.**

```
library(MASS}
print(ulft<-t(uu[1:3,]))
print(urgt<--uu[4,])
print(uinv<-ginv(ulft))
print(cc<-c(drop(uinv%*%urgt),1))
```

B.7. **Small Example Chunk.** set.seed(12345) s<-matrix(rnorm(16),4,4) t<-sx0<-rnorm(4) b<-rnorm(4)

B.8. **Symmetric Example Chunk.**

```
set.seed(12345)
x<-matrix(rnorm(10000),100,100)
t<-crossprod(x)/500
b<-rnorm(100)
x0<-rnorm(100)
xstar<-solve(diag(100)-t,b)
```

B.9. **MDS Example Chunk.**

```
set.seed(12345)
mkMDS(10,15)
x0<-1:10
```

APPENDIX C. PROGRAM

C.1. **Main.**

```
1
2  sidi<-function(f,xinit,r,itmax=100,eps=1e-10,method="mpe",
       verbose=FALSE,noisy=FALSE,stable=TRUE) {
3  xold<-xinit; itel<-1; xx<-matrix(0,r+2,length(xinit)); oaps
       <-1; oerr<-1; aps<-Inf; s<-1/(r+1)
4  repeat{
5      xx[1,]<-xold
6      for (i in 2:(r+2)) xx[i,]<-f(xx[i-1,])
7      if (r == 0) xnew<-xx[2,]
8      if (r > 0) {
9          if (method == "rre") {
10         u<-t(diff(xx)); v<-rowMeans(u)
11         ac<-lsfit(u[,-(r+1)]-u[,r+1],-u[,r+1],intercept=FALSE)
               $coef
12         cc<-c(ac,1-sum(ac))
13         xnew<-drop(cc%*%xx[-(r+2),])
14         }
15         else {
16             u<-diff(xx)
17             if (r == 1) uu<-as.matrix(u[1,])
18                 else uu<-t(u[1:r,])
19             cc<-c(lsfit(uu,-u[r+1,],intercept=FALSE)$coef,1)
20             xnew<-drop(cc%*%xx[-(r+2),])/sum(cc)
21             }
22         }
23     xstb<-f(xnew)
24     naps<-norm(xold-xnew); crat<-naps/oaps
25     nerr<-norm(xnew-xstb); erat<-nerr/oerr
26     if (stable) xnew<-xstb
27     if (verbose) {
28         cat("Iteration: ",
29             formatC(itel,digits=6,width=6),
30             "    Change:    ",
31             formatC(naps,digits=10,width=15,format="f"),
32             "    Error:    ",
33             formatC(nerr,digits=10,width=15,format="f"),
```

```
                "   CRatio:    ",
                formatC(crat,digits=10,width=15,format="f"),
                "   ERatio:    ",
                formatC(erat,digits=10,width=15,format="f"),
               "\n")
            }
        if (noisy) {
            cat("Solution:   ",
                format(xnew,digits=10,width=15,format="f"),
               "\n")
            }
        if ((itel == itmax) || (nerr < eps)) break()
        itel<-itel+1
        xold<-xnew
        oaps<-naps
        oerr<-nerr
        }
    return(list(itel=itel,aps=naps,err=nerr,crat=crat,erat=erat,x=
        xnew))
    }
```

C.2. **Auxilaries.**

```

mkMDS<-function(p,m) {
asum<-matrix(0,p,p); ab<-array(0,c(p,p,m))
for (i in 1:m) {
          x<-matrix(rnorm(100*p),100,p)
          a<-crossprod(x); ab[,,i]<-a
          asum<-asum+a
          }
eig<-eigen(asum); k<-eig$vectors; d<-1/sqrt(outer(eig$values,
    eig$values))
for (i in 1:m) ab[,,i]<-(crossprod(k,ab[,,i])%*%k)*d
aa<<-ab; del<-rchisq(m,1); del<<-del/norm(del)
}

fmds<-function(x) {
m<-dim(aa)[3]; p<-length(x); asum<-matrix(0,p,p); d<-rep(0,m)
for (i in 1:m) {
```

```
17              d[i]<-sqrt(sum(aa[,,i]*outer(x,x)))
18              asum<-asum+(del[i]/d[i])*aa[,,i]
19              }
20  return(drop(asum%*%x))
21  }
22
23  flin<-function(z) drop(t%*%z)+b
24
25  norm<-function (x) sqrt(sum(x^2))
```

DEPARTMENT OF STATISTICS, UNIVERSITY OF CALIFORNIA, LOS ANGELES, CA 90095-1554

E-mail address, Jan de Leeuw: deleeuw@stat.ucla.edu

URL, Jan de Leeuw: http://gifi.stat.ucla.edu