Matrix Completion with ε-Algoritm*

Walter Gander

ETH Zurich and HKBU Hong Kong

In memory of Peter Wynn (1931-2017)

The Thirteenth International Conference on Matrix Theory and Applications
Harbin Engineering University, Harbin Heilongjiang, P.R. China
August 17–22, 2018

* Walter Gander and Qiquan Shi, Numerical Algorithms, Springer, August 2018,
DOI 10.1007/s11075-018-0579-y.

Matrix Completion Problem

Matrix completion is the task of filling in the missing entries of a partially observed matrix M[a].

- Problem is not well posed without restrictions
- Therefore one considers
  1. $\min_X \operatorname{rank}(X)$ subject to $X(\Omega) = M(\Omega)$
     $\Omega =$ logical index matrix selecting the observed elements
  2. find X with given rank r such that $X(\Omega) = M(\Omega)$

[a] `https://en.wikipedia.org/wiki/Matrix_completion`

Algorithm R1MC

- Shi et al. proposed an algorithm[a] for Problem 2:

```
% Algorithm R1MC in Matlab pseudocode
Given M with missing entries
Define Omega the index of the observed entries
Define rank r
Initialize Z=0
while not converged
  Z(Omega)=M(Omega);                % copy observed elements
  [U,S,V]=svd(Z);
  Z=U(:,1:r)*S(1:r,1:r)*V(:,1:r)'; % reduce to rank r
end
```

- R1MC converges under suitable conditions to the desired limit

[a] Shi Qiquan, Lu Haiping and Cheung Yiu-ming, *Rank-One Matrix Completion With Automatic Rank Estimation via L1-Norm Regularization*, IEEE Transactions on Neural Networks and Learning Systems, 2017.

Our Experimental Version of R1MC

```
function [Err,rho,Z]=R1MCexp(maxit,B,r,Omega)
% R1MCEXP experimental matrix completion by using R1MC
%     B is the exact rank r matrix, Omega logical index
%     matrix defining the observed elements,
%     maxit the number of iterations or SVDs
%     Err is a vector containing the relative error after each step,
%     rho is the ratio of the last two relative errors,
%     Z approximation of completed matrix after maxit steps
M=B;
Z=zeros(size(B));
Err=[];
for k=1:maxit
  Z(Omega)=M(Omega);                          % copy observed elements
  [U,S,V]=svd(Z);
  Z=U(:,1:r)*S(1:r,1:r)*V(:,1:r)';            % reduce to rank r
  Err=[Err;norm(B-Z,'fro')/norm(B,'fro')]; % store relative error
end
rho=Err(end)/Err(end-1);                      % final error constant
```

Example 1

- Given matrix M, *=missing elements

$$M = \begin{pmatrix} * & 2 & * & 4 & 5 & * \\ * & 4 & 6 & 8 & * & 12 \\ * & * & * & * & 15 & * \\ 4 & * & * & 16 & 20 & * \\ * & * & * & 20 & 25 & 30 \\ 6 & 12 & * & 24 & * & 36 \end{pmatrix} \quad \Omega = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

- Rank $r = 1$
- Start R1MC with $Z = 0$, copy observed elements $Z(\Omega) = M(\Omega)$ and iterate.

Run Example 1

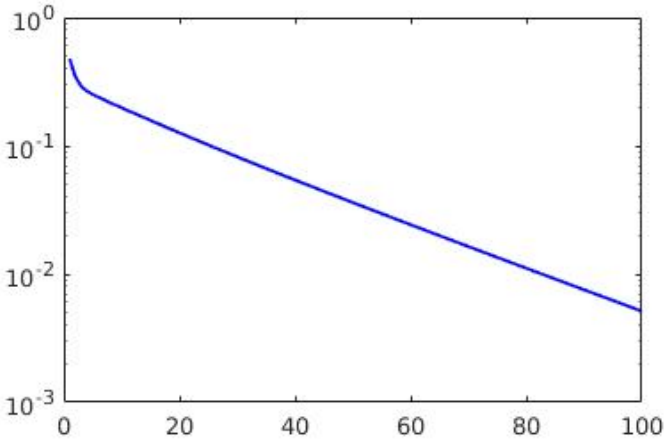
```
% Example1.m
n=6; u=[1:n]; B=u'*u;                          % B exact rank-1 matrix
M=[  0     2     0     4     5     0  % M is B with missing elements
     0     4     6     8     0    12
     0     0     0     0    15     0
     4     0     0    16    20     0
     0     0     0    20    25    30
     6    12     0    24     0    36];
Omega=M>0;                                     % Index array of observed elements
r=1;                                           % rank=r

[Err,rho,Z]=R1MCexp(100,B,r,Omega);
Z
rho
figure(1), semilogy(Err,'b')                   % plot error behaviour
LastRelativeError=Err(end)                     % and error constants
figure(2), plot(Err(2:end)./Err(1:end-1),'b')
```

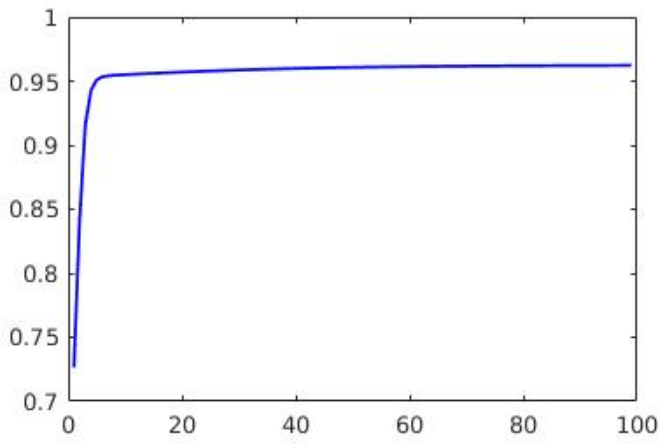
Results after 100 Iterations

R1MCdemo

```
Z =
    0.9998    1.9990    2.9504    3.9989    5.0000    5.9978
    2.0049    4.0084    5.9162    8.0186   10.0262   12.0269
    2.9986    5.9952    8.8486   11.9931   14.9956   17.9880
    3.9992    7.9956   11.8011   15.9948   19.9992   23.9900
    5.0000    9.9965   14.7543   19.9975   25.0040   29.9936
    6.0006   11.9970   17.7070   23.9994   30.0078   35.9959
rho =  0.9623
LastRelatveError =  0.0052
```



Linear convergence



slow convergence $e_{k+1} \sim 0.96\ e_k$

can we accelerate convergence?

Accelerating Convergence for Limit Computations

- Well known methods
  - Romberg integration
  - Richardson Extrapolation
  - Aitken's Δ^2 process
  - Steffenson
- Powerful: the ε-algorithm

  Reference:

  P. Wynn: Acceleration techniques for iterated vector and matrix problems, Math. of Comp., 16 (1962), pp. 301-322

Example: Improve Convergence for Trapezoidal Rule

- Consider computing $\int_a^b f(x)\,dx$ by Trapezoidal Rule

$$h_n = \frac{b-a}{2^n}, \quad T(h_n) = h_n \left(\frac{1}{2} f(a) + \sum_{k=1}^{2^n-1} f(a + k h_n) + \frac{1}{2} f(b) \right), \; n = 0, 1, 2, \ldots$$

- For $n \to \infty$, $h_n \to 0 \implies T(h_n) \to \int_a^b f(x)\,dx$

  f on $[a, b]$ sufficiently differentiable $\implies$ asymptotic expansion:

$$T(h) = a_0 + a_2 h^2 + a_4 h^4 + a_6 h^6 + \cdots \text{ (only even powers)}$$

- Romberg scheme: interpolating $T(h)$ by polynom $P(h^2)$

$$P(h_i^2) = T(h_i), i = 0, \ldots, k$$

  and evaluating $a_0 \approx P(0)$

Convergence acceleration with Romberg

$$\begin{array}{cccc} T_{00} & & & \\ T_{10} & T_{11} & & \\ \vdots & & \ddots & \\ T_{k,0} & \dots & \dots & T_{k,k} \end{array}$$

- Results for $I = \int_a^b \sqrt{x}\, dx$

$a = 1$	$b = 2$	$T_{8,0} = 1.2189506716330 83$	$T_{8,8} = 1.218951416497460$	good
$a = 0$	$b = 1$	$T_{15,0} = 0.666666633974581$	$T_{15,15} = 0.666666633974885$	no improvement!

- Reason: for $a = 0$ and $b = 1$ the asymptotic expansion is

$$T(h) = \zeta\left(-\frac{1}{2}\right) h^{3/2} + \frac{2}{3} \sum_{j=0}^{\infty} \binom{3/2}{2j} B_{2j}\, h^{2j}.$$

Romberg does not eliminate the term with $h^{3/2}$.

Accelerating Linear Convergence

- Assume $x_k \to s$ with linear convergence

$$x_n - s \sim \rho\,(x_{n-1} - s) \sim \rho^n\,(x_0 - s), \quad |\rho| < 1$$

$$\Longleftrightarrow \; x_n \sim s + C\rho^n \quad \text{model of asymptotic behaviour}$$

- replace "$\sim$" by "$=$" and solve for ρ, C and s for $n-2, n-1, n$

```
solve({x[n-2]=s+C*rho^(n-2),x[n-1]=s+C*rho^(n-1),x[n]=s+C*rho^n},{rho,C,s}):
assign(%): s:=simplify(s);
```

$$s := \frac{x_n x_{n-2} - x_{n-1}^2}{x_n - 2\,x_{n-1} + x_{n-2}}$$

- $\Longrightarrow$ new sequence $x'_n = s$ (rearranged):

$$x'_n = x_{n-2} - \frac{(x_{n-1} - x_{n-2})^2}{x_n - 2\,x_{n-1} + x_{n-2}} = x_{n-2} - \frac{(\Delta x_{n-2})^2}{\Delta^2 x_{n-2}}$$

Aitken's Δ^2 process

Iterating Aitken's Δ^2 Process

– accelerating trapezoidal values $T(h)$ for $\int_0^1 \sqrt{x}\,dx = \frac{2}{3} = 0.6666666666666\ldots$

– still linear convergence but faster!

$T(h)$	x': Aitken for $T(h)$	x'': Aitken for x'	x''': Aitken for x''
0.500000000000000			
0.603553390593274			
0.643283046242747	0.668014371343284		
0.658130221624454	0.666989411481856		
0.663581196877228	0.666743446595054	0.666665784058695	
0.665558936278942	0.666685106026354	0.666666965403696	
0.666270811378507	0.666671143512465	0.666666750535015	0.666666783601981
0.666525657296826	0.666667762975738	0.666666683023628	0.666666652093451
0.666616548976528	0.666666936801258	0.666666669586115	0.666666666246855
0.666648881549952	0.666666733518171	0.666666667175926	0.666666666649143
0.666660362218984	0.666666683260887	0.666666666755209	0.666666666666239
0.666664433592971	0.666666670794491	0.666666666682097	0.666666666666719
0.666665876127180	0.666666667694990	0.666666666669366	0.666666666666682

Shanks Transform

- Aitken assumes the model $x_n \sim s + C\rho^n$

$$S_1(x_n) := \frac{x_n x_{n-2} - x_{n-1}^2}{x_n - 2\,x_{n-1} + x_{n-2}}$$ first Shanks Transform of $\{x_n\}$

- Generalization for model with more terms:

$$x_n \sim s + \sum_{i=1}^{k} a_i \rho_i^n$$

The Shanks Transform transforms $\{x_n\}$ to $S_k(x_n) = s_{n,k}$ by solving the non-linear system of $2k+1$ equations

$$x_{n+j} = s_{n,k} + \sum_{i=1}^{k} \tilde{a}_i \tilde{\rho}_i^{n+j}, \quad j = 0, 1, \ldots, 2k$$

for the $2k+1$ unknowns $s_{n,k}$, $\tilde{a}_i$ and $\tilde{\rho}_i \rightarrow$ difficult for larger k.

ε-Algorithm: Peter Wynn found 1956 a remarkable recursion for $s_{n,k}$

- Initialize $\varepsilon_{-1}^{(n)} = 0$ and $\varepsilon_0^{(n)} = x_n$ for $n = 0, 1, 2, \ldots$.

$$\varepsilon_{k+1}^{(n)} = \varepsilon_{k-1}^{(n+1)} + \frac{1}{\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}}$$

- ε-scheme

$$\begin{array}{lllll}
\varepsilon_{-1}^{(0)} & & & & \\
& \varepsilon_0^{(0)} & & & \\
\varepsilon_{-1}^{(1)} & & \varepsilon_1^{(0)} & & \\
& \varepsilon_0^{(1)} & & \varepsilon_2^{(0)} & \\
\varepsilon_{-1}^{(2)} & & \varepsilon_1^{(1)} & & \varepsilon_3^{(0)} \\
& \varepsilon_0^{(2)} & & \varepsilon_2^{(1)} & \cdots \\
\varepsilon_{-1}^{(3)} & & \varepsilon_1^{(2)} & \cdots & \\
& \varepsilon_0^{(3)} & & \cdots & \\
\varepsilon_{-1}^{(4)} & & \cdots & &
\end{array}$$

- even columns contain $\varepsilon_{2k}^{(n)} = s_{n,k}$

  odd columns are $\varepsilon_{2k+1}^{(n)} = \dfrac{1}{S_k(\Delta x_n)}$, Shanks Transform of $\Delta x_n = x_{n+1} - x_n$

Peter Wynn

1931-2017

- 1959 PhD with Friedrich L. Bauer
- Professor positions at several universities in the United States, Canada and Mexico.

Picture by Claude Brezinski 1974

Matlab-Implementation

- ε-scheme as lower triangular matrix E, shift indices

$$
\begin{array}{llllll}
0 = \varepsilon_{-1}^{(0)} = E_{11}, & & & & & \\
0 = \varepsilon_{-1}^{(1)} = E_{21} & x_1 = \varepsilon_0^{(0)} = E_{22}, & & & & \\
0 = \varepsilon_{-1}^{(2)} = E_{31} & x_2 = \varepsilon_0^{(1)} = E_{32} & \varepsilon_1^{(0)} = E_{33}, & & & \\
0 = \varepsilon_{-1}^{(3)} = E_{41} & x_3 = \varepsilon_0^{(2)} = E_{42} & \varepsilon_1^{(1)} = E_{43} & \varepsilon_2^{(0)} = E_{44}. &
\end{array}
$$

- `Er` contains the reduced ε-scheme (only even columns):

```
function Er=EpsilonAlgorithm(x,k);
%
n=2*k+1; E=zeros(n+1,n+1);
for i=1:n
  E(i+1,2)=x(i);
end
for i=3:n+1
  for j=3:i
    E(i,j)=E(i-1,j-2)+1/(E(i,j-1)-E(i-1,j-1));
  end
end
Er=E(:,2:2:n+1);
```

Apply to Integration $\int_0^1 \sqrt{x}\,dx = \frac{2}{3}$ by trapezoidal rule

we display the error of the last row of each scheme

Romberg	iterated Aitken	ε-Algorithm
$T(9,1:9) - 2/3$	$A(9,:) - 2/3$	$Er(9,:) - 2/3$
-5.0118e-05	-5.0118e-05	-5.0118e-05
-1.9820e-05	2.7013e-07	2.7013e-07
-1.7404e-05	2.9194e-09	-1.0057e-10
-1.6899e-05	-4.1981e-10	1.7014e-12
-1.6778e-05	-1.7950e-09	-6.9138e-13
-1.6748e-05		
-1.6741e-05		
-1.6739e-05		
-1.6738e-05		

Fixed Point Iteration:

solve $x = e^{-x}$

iterate and restart

with extrapolated values

Steffensen's method:

```
>> f=@(x) exp(-x);
>> x=Steffensen(f,0)
x =
   0.567143290409784
```

$x_{k+1} = e^{-x_k}$	Aitken Δ^2
0.000000000000000	
1.000000000000000	
0.367879441171442	→ 0.612699836780282
0.612699836780282 ↙	
0.541885888907111	
0.581650289552568	→ 0.567350857702887
0.567350857702887 ↙	
0.567025582228799	
0.567210051743956	→ 0.567143294830715
0.567143294830715 ↙	
0.567143287902483	
0.567143291831783	→ 0.567143290409784

Our matrix completion problem is a fixed point problem.

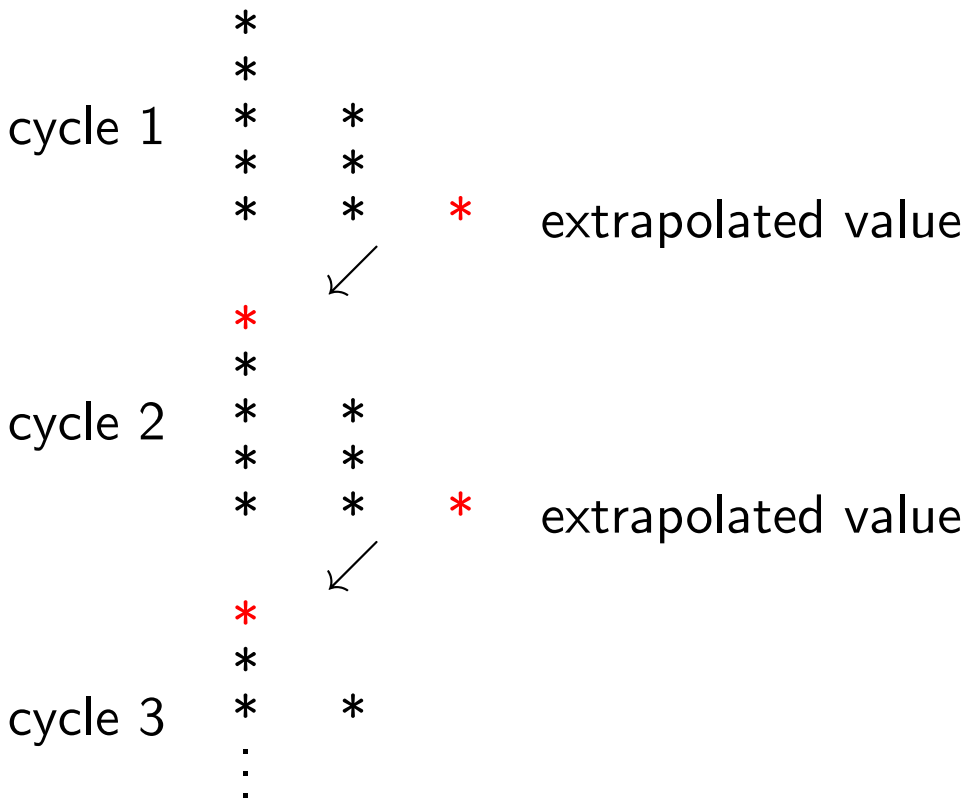
Alternative Implementation of ε-Algorithm

Don't store the whole ε-scheme, keep only last row and diagonal

step i	0	$e_i = x_i$	e_{i-1}	e_{i-2}	$\cdots$	e_2	e_1	
step $i+1$	0	$e_{i+1} = x_{i+1}$	e_i	e_{i-1}	$\cdots$	e_3	e_2	e_1

```
function [W,e]=EpsilonAlgorithmLowStorage(x,k);
e(1)=x(1);
for i=2:2*k+1
  v=0; e(i)=x(i);              % last row is e_{i-1}, ... e_2,e_1
  for j=i:-1:2                 % compute new row
    d=e(j)-e(j-1); w=v+1/d;    % w and v hold copies of elements
    v=e(j-1); e(j-1)=w;        % which are overwritten
  end;
  W(i-1)=w;                    % store diagonal elements
end
```

Restarted ε-Algorithm (Generalization of Steffensen)



k number of columns of `Er=E(:,2:2:end)`

$2k+1$ basic iterations per cycle

Example above: $k = 2$, ($k = 1$: Steffensen)

ε-Algorithm for Matrices

- accelerate matrix sequence $\{Z_k\}$
- scalars $\varepsilon_k^{(n)} \rightarrow$ matrices $\mathcal{E}_k^{(n)}$
- recursion

$$\varepsilon_{k+1}^{(n)} = \varepsilon_{k-1}^{(n+1)} + \frac{1}{\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}} \quad \rightarrow \quad \mathcal{E}_{k+1}^{(n)} = \mathcal{E}_{k-1}^{(n+1)} + \left(\mathcal{E}_k^{(n+1)} - \mathcal{E}_k^{(n)}\right)^{-1}$$

- use 3d-arrays in MATLAB for ε-scheme
- use `EpsilonAlgorithmLowStorage` to save memory
- use restarted ε-Algorithm

Applying Matrix ε-Algorithm

- MATLAB program segment for one cycle:

```
E(:,:,1)=X(:,:,1);
for i=2:2*k+1
  V=zeros(n,n); E(:,:,i)=X(:,:,i);
  for j=i:-1:2
    W=V+inv(E(:,:,j)-E(:,:,j-1));
    V=E(:,:,j-1); E(:,:,j-1)=W;
  end;
  D(:,:,k-1)=W;
end
```

```
e(1)=x(1);
for i=2:2*k+1
  v=0; e(i)=x(i);
  for j=i:-1:2
    d=e(j)-e(j-1); w=v+1/d;
    v=e(j-1); e(j-1)=w;
  end;
  D(i-1)=w;
end
```

- Does not work for sequence $\{Z_j\}$ for matrix completion!
- Reason: $\operatorname{rank}(Z_j) = r < n \implies$ cannot invert $(Z_j - Z_{j-1})$
- After copying observed elements $Z(\Omega) = M(\Omega)$ very likely that $\operatorname{rank}(Z(\Omega)) = n \implies$ extrapolate with $\{Z_j(\Omega)\}$

Rearrange R1MC

Original R1MC	Rearranged R1MC
$Z = Z_k$	$[U, \Sigma, V] = \mathrm{svd}(Z_k)$
$Z(\Omega) = M(\Omega)$	$Z = U_r \Sigma_r V_r$
$[U, \Sigma, V] = \mathrm{svd}(Z)$	$Z(\Omega) = M(\Omega)$
$Z_{k+1} = U_r \Sigma_r V_r$	$Z_{k+1} = Z$
$\mathrm{rank}(Z_{k+1}) = r$	$\mathrm{rank}(Z_{k+1}) = n$

Results

- Small example:
  M is 6×6, 50% elements observed, rank=1

	svd	rel. error	ρ		
R1MC	100	0.0052	0.9623		
			cycles	k	inv
MatrixEps	19	0.0041	2	4	72

- Lena Dataset:
  M is 512×512, 50% elements observed, rank=29

	svd	rel. error	time	ρ		
R1MC	200	1.1086e-10	17.46	0.9184		
				cycles	k	inv
MatrixEps	85	3.2603e-10	18.55	4	10	840

Pro and Cons for Matrix ε-Algorithm

- Pro
  - For same error: less SVD's than R1MC
- Cons
  - memory intensive
  - (too) many inversions
  - only for square matrices
  - if few missing elements: $\Longrightarrow$ $\mathcal{E}_k^{(n+1)} - \mathcal{E}_k^{(n)}$ gets ill conditioned
  - Lena dataset: no convergence for $k < 10$
- numerical analyst's wisdom proved true again:

  Never invert a matrix!

New Approach: Accelerate convergence only for missing elements!

- store missing elements as vector, apply ε-algorithm to vectors
- "inverse of a vector": use pseudoinverse $\mathbf{y}^{-1} := \frac{1}{||\mathbf{y}||^2}\mathbf{y}$

```
E(:,1)=Z(OmegaC);                          % OmegaC = Omega complement
for i=2:2*k+1                              % Z(OmegaC) is a vector
  V=zeros(size(OmegaC));
  Z(Omega) = B(Omega);
  [u,s,v]=svd(Z); countSVD=countSVD+1;
  Z=u(:,1:r)*s(1:r,1:r)*v(:,1:r)';
  E(:,i)=Z(OmegaC);                        % compute epsilon scheme
  for j=i:-1:2                             % for extracted vectors
    d=E(:,j)-E(:,j-1);                     % store only last row and
    w=V+d/(d'*d);
    V=E(:,j-1); E(:,j-1)=w;
  end;
  D(:,i-1)=w;                              % diagonal of epsilon scheme
```

Results

- Small example:
  M is 6×6, 50% elements observed, rank=1

	svd	rel. error	ρ	
R1MC	100	0.0052	0.9623	
			cycles	k
VectorEps	27	0.0033	3	4

- Lena Dataset:
  M is 512×512, 50% elements observed, rank=29

	svd	rel. error	time	ρ	
R1MC	200	1.1086e-10	17.46	0.9184	
				cycles	k
VectorEps	65	2.2363e-10	6.22	5	6

Variing k

R1MC 100 iterations

	ρ	svd	Error	time
	9.1221e-01	100	6.4182e-07	8.73

VectorEps

k	cycles	svd	Error	time
1	14	42	1.5539e-05	3.72
2	7	35	8.6707e-06	3.18
3	5	35	5.3019e-06	3.18
4	4	36	6.6667e-06	3.29
5	4	44	3.7303e-07	4.04
6	3	39	1.5705e-05	3.61
7	3	45	6.9305e-06	4.22
8	3	51	2.0060e-07	4.79
9	3	57	6.5517e-08	5.37
10	3	63	1.7938e-08	6.09

R1MC 200 iterations

	ρ	svd	Error	time
	0.9185	200	1.1086e-10	17.07

VectorEps

k	cycles	svd	Error	time
1	32	96	7.6425e-09	8.41
2	15	75	1.8416e-09	6.75
3	9	63	1.9328e-09	5.80
4	7	63	4.0925e-10	5.81
5	6	66	5.6073e-11	6.09
6	5	65	2.2363e-10	6.01
7	5	75	3.6199e-11	7.13
8	4	68	1.6096e-10	6.53
9	4	76	4.2561e-11	7.43
10	4	84	1.0494e-11	8.37

Adapt $k = 5$ as default

Choose Tolerance

- Goal: Given M, index Ω and rank r, compute completed matrix.
- Terminate iteration when

$$\|Z_{2k} - Z_{2k+1}\|_2 \leq \tau \|Z_{2k+1}\|_2$$

τ prescribed tolerance

- Since $\|Z_j\|_2 = \sigma_{\max}(Z_j)$ and SVD of basic iterations available, compare $\sigma_{\max}$ of the two last matrices of a cycle.

τ	Rel.Error	#svd
$1e{-}02$	$5.9392e{-}02$	11
$1e{-}03$	$3.0263e{-}03$	22
$1e{-}04$	$5.1112e{-}05$	33
$1e{-}05$	$5.1112e{-}05$	33
$1e{-}06$	$3.7303e{-}07$	44
$1e{-}07$	$3.7303e{-}07$	44
$1e{-}08$	$2.6960e{-}09$	55
$1e{-}09$	$2.6960e{-}09$	55
$1e{-}10$	$5.6073e{-}11$	66

Use default $\tau = 10^{-5}$

```
function [Z,countSVD] = VectorEps(M,r,Omega,k,tol)
% VECTOREPS matrix completion by accelerating convergence of algorithm
%     R1MC with the vector epsilon algorithm.
%     M is the matrix to be completed with rank r.
%     tol: stop iteration if
%     |sigma_1(Z_k)-sigma_1(Z_{k+1})|<sigma_1(Z_{k+1}
%     Omega logical index matrix defining the observed elements,
%     2k+1 basic iterations per cycle
%     countSVD = total number of svd after each cycle
if nargin<5; tol=1e-5; end
if nargin<4 | isempty(k); k=5; end
[m,n]=size(M);
Z=zeros(size(M));
OmegaC=find(Omega==0);                    % index vector of missing elements
countSVD=0;
sigma1old=0; sigma1=1;
while abs(sigma1-sigma1old)>tol*sigma1
  Z(Omega)=M(Omega);                      % add observed elements
  [u,s,v]=svd(Z); countSVD=countSVD+1;    % svd of current approximation
  Z=u(:,1:r)*s(1:r,1:r)*v(:,1:r)';        % reduce to rank r
```

```
  E(:,1)=Z(OmegaC);                          % extract vector of missing elements
  for i=2:2*k+1
    V=zeros(size(OmegaC));
    Z(Omega) = M(Omega);
    sigma1old=sigma1;
    [u,s,v]=svd(Z); countSVD=countSVD+1;
    sigma1=s(1,1);
    Z=u(:,1:r)*s(1:r,1:r)*v(:,1:r)';
    E(:,i)=Z(OmegaC);                        % compute epsilon scheme
    for j=i:-1:2                             % for extracted vectors
      d=E(:,j)-E(:,j-1);                     % store only last row and
      w=V+d/(d'*d);
      V=E(:,j-1); E(:,j-1)=w;
    end;
    D(:,i-1)=w;                              % diagonal of epsilon scheme
  end
  Z(OmegaC)=w;                               % w=D(:,2*k) best value
end
```

Example of Picture Reconstruction

```
% L1.m
Lena=imread('Lena.png');           % read picture m x n x 3 as uint8 tensor
figure(1), imshow(uint8(Lena))     % show original picture
LenaGrey=rgb2gray(Lena);           % reduce to m x n uint8 matrix
figure(2),imshow(uint8(LenaGrey)) % show black and white picture
LenaReal=double(LenaGrey);         % convert to real matrix
r=30                               % choose rank
[u,s,v]=svd(LenaReal);
LenaR=u(:,1:r)*s(1:r,1:r)*v(:,1:r)';
figure(3), imshow(uint8(LenaR))    % exact low rank image
[m,n]=size(LenaR); N=m*n;
ObsRatio=0.5                       % choose observation ratio
rng(1)                             % seed for random generator
idd=randperm(N)<=ObsRatio*N;       % construct index of
Omega=reshape(idd,m,n);            % observed element
M=zeros(m,n); M(Omega)=LenaR(Omega); % form incomplete matrix
Z=M; figure(4); imshow(uint8(Z)); % show incomplete picture
[Z,Tsvd]=VectorEps(Z,r,Omega);     % reconstruct the picture
Tsvd, figure(5), imshow(uint8(Z)) % show completed picture
```

original

after rgb2gray

rank 30

incomplete

completed with `VectorEps`