Meshfree Approximation with MATLAB

Lecture III: Dealing with Ill-Conditioned RBF Systems

Greg Fasshauer

Department of Applied Mathematics
Illinois Institute of Technology

Dolomites Research Week on Approximation
September 8–11, 2008

Outline

Multivariate RBF Interpolation

Use data-dependent linear function space

$$\mathcal{P}_f(\boldsymbol{x}) = \sum_{j=1}^{N} c_j \Phi(\boldsymbol{x}, \boldsymbol{x}_j), \qquad \boldsymbol{x} \in \mathbb{R}^s$$

Here $\Phi : \mathbb{R}^s \times \mathbb{R}^s \to \mathbb{R}$ is strictly positive definite (reproducing) kernel

Multivariate RBF Interpolation

Use data-dependent linear function space

$$\mathcal{P}_f(\boldsymbol{x}) = \sum_{j=1}^{N} c_j \Phi(\boldsymbol{x}, \boldsymbol{x}_j), \qquad \boldsymbol{x} \in \mathbb{R}^s$$

Here $\Phi : \mathbb{R}^s \times \mathbb{R}^s \to \mathbb{R}$ is strictly positive definite (reproducing) kernel

To find c_j solve interpolation equations

$$\mathcal{P}_f(\boldsymbol{x}_i) = f(\boldsymbol{x}_i), \quad i = 1, \ldots, N$$

Leads to linear system $A\boldsymbol{c} = \boldsymbol{f}$ with symmetric positive definite — often ill-conditioned — system matrix

$$A_{ij} = \Phi(\boldsymbol{x}_i, \boldsymbol{x}_j), \quad i, j = 1, \ldots, N$$

Consider the linear system

$$A\boldsymbol{c} = \boldsymbol{f}$$

with $N \times N$ symmetric positive definite matrix A.

Consider the linear system

$$A\boldsymbol{c} = \boldsymbol{f}$$

with $N \times N$ symmetric positive definite matrix A.

Standard textbook knowledge suggests:

1. Compute Cholesky factorization $A = LL^T$.
2. Solve the lower triangular system $L\boldsymbol{y} = \boldsymbol{f}$ for $\boldsymbol{y}$ by forward substitution.
3. Solve the upper triangular system $L^T\boldsymbol{c} = \boldsymbol{y}$ for $\boldsymbol{c}$ by back substitution.

Consider the linear system

$$A\boldsymbol{c} = \boldsymbol{f}$$

with $N \times N$ symmetric positive definite matrix A.

Standard textbook knowledge suggests:

1. Compute Cholesky factorization $A = LL^T$.
2. Solve the lower triangular system $L\boldsymbol{y} = \boldsymbol{f}$ for $\boldsymbol{y}$ by forward substitution.
3. Solve the upper triangular system $L^T\boldsymbol{c} = \boldsymbol{y}$ for $\boldsymbol{c}$ by back substitution.

Problem: Doesn't work well if A is ill-conditioned.

The Basic Idea

Let P be a preconditioning matrix so that

$$\mathrm{cond}(PA) \ll \mathrm{cond}(A) \quad \text{or} \quad \mathrm{cond}(AP) \ll \mathrm{cond}(A)$$

The Basic Idea

Let P be a preconditioning matrix so that

$$\mathrm{cond}(PA) \ll \mathrm{cond}(A) \quad \text{or} \quad \mathrm{cond}(AP) \ll \mathrm{cond}(A)$$

- Change both sides (left preconditioning)

$$(PA)\boldsymbol{c} = P\boldsymbol{f} \Longleftrightarrow \boldsymbol{c} = (PA)^{-1}P\boldsymbol{f}$$

The Basic Idea

Let P be a preconditioning matrix so that

$$\operatorname{cond}(PA) \ll \operatorname{cond}(A) \quad \text{or} \quad \operatorname{cond}(AP) \ll \operatorname{cond}(A)$$

- Change both sides (left preconditioning)

$$(PA)\boldsymbol{c} = P\boldsymbol{f} \iff \boldsymbol{c} = (PA)^{-1}P\boldsymbol{f}$$

- Change only the left-hand-side (right preconditioning)

$$(AP)\widetilde{\boldsymbol{c}} = \boldsymbol{f} \iff \widetilde{\boldsymbol{c}} = (AP)^{-1}\boldsymbol{f} \qquad \widetilde{\boldsymbol{c}} = P^{-1}\boldsymbol{c}$$

We will use this approach

How to find *P*

- **Previous work on RBF preconditioning:**
  - Dyn and co-workers in the mid 1980s [Dyn (1987), Dyn *et al.* (1986)]: discretize associated differential operator (bi-Laplacian for TPSs)
  - more recent papers [Barba & Rossi (2008), Baxter (2002), Beatson *et al.* (1999), Brown *et al.* (2005), Ling & Kansa (2005)]: mostly with local approximate cardinal functions
  - better basis of the approximation space [Beatson *et al.* (2000)]: use homogeneous kernel for TPSs

How to find P

- **Previous work on RBF preconditioning:**
  - Dyn and co-workers in the mid 1980s [Dyn (1987), Dyn *et al.* (1986)]: discretize associated differential operator (bi-Laplacian for TPSs)
  - more recent papers [Barba & Rossi (2008), Baxter (2002), Beatson *et al.* (1999), Brown *et al.* (2005), Ling & Kansa (2005)]: mostly with local approximate cardinal functions
  - better basis of the approximation space [Beatson *et al.* (2000)]: use homogeneous kernel for TPSs
- **Our approach:**
  - use global approximate cardinal functions ([Zhang (2007)]) (related to polynomial preconditioners, e.g., [Dubois *et al.* (1979), Ashby *et al.* (1992)])

On polynomial preconditioners

From [Benzi (2002)]:

Preconditioning as a means of reducing the condition number in order to improve convergence of an iterative process seems to have been first considered by [Cesari (1937)]. Cesari's idea was to use a low degree polynomial $p(A)$ in A as a preconditioner for a Richardson-type iteration applied to the preconditioned system $p(A)A\boldsymbol{x} = p(A)\boldsymbol{b}$.

On polynomial preconditioners

From [Benzi (2002)]:

Preconditioning as a means of reducing the condition number in order to improve convergence of an iterative process seems to have been first considered by [Cesari (1937)]. Cesari's idea was to use a low degree polynomial $p(A)$ in A as a preconditioner for a Richardson-type iteration applied to the preconditioned system $p(A)A\boldsymbol{x} = p(A)\boldsymbol{b}$.

Polynomial preconditioners for Krylov subspace methods came into vogue in the late 1970s with the advent of vector computers but they are currently out of favor because of their limited effectiveness and robustness, especially for nonsymmetric problems.

On polynomial preconditioners

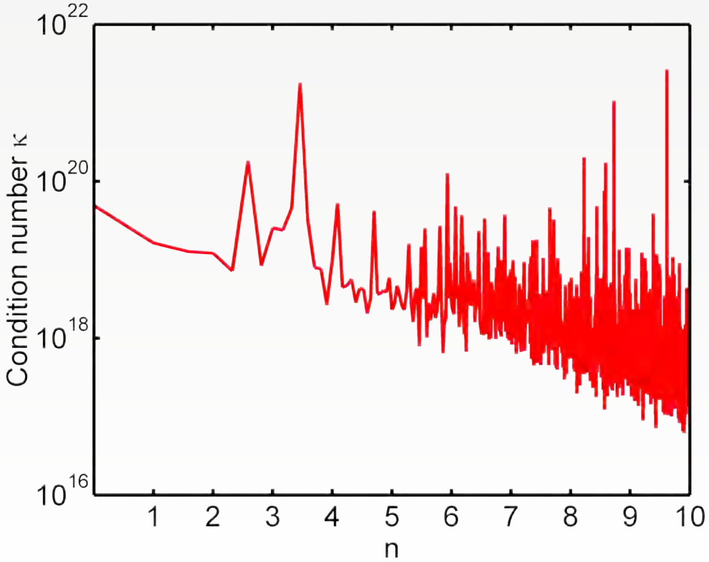
From [Benzi (2002)]:

Preconditioning as a means of reducing the condition number in order to improve convergence of an iterative process seems to have been first considered by [Cesari (1937)]. Cesari's idea was to use a low degree polynomial $p(A)$ in A as a preconditioner for a Richardson-type iteration applied to the preconditioned system $p(A)A\boldsymbol{x} = p(A)\boldsymbol{b}$.

Polynomial preconditioners for Krylov subspace methods came into vogue in the late 1970s with the advent of vector computers but they are currently out of favor because of their limited effectiveness and robustness, especially for nonsymmetric problems.

- Conclusion reported in [Ashby *et al.* (1992)]: $\deg(p) = 2$ "optimal"
- We may be providing new insight via acceleration

Condition number after $2^n - 1$ iterations



Theorem

Part I (without acceleration)

$$\mathcal{Q}_f^{(n)} = \boldsymbol{\Phi}^T \underbrace{\sum_{k=0}^{n} (I - A)^k}_{=P^{(n)}} \boldsymbol{f} =: \boldsymbol{\Phi}^{(n)T} \boldsymbol{f},$$

i.e., $\{\Phi^{(n)}(\cdot, \boldsymbol{x}_1), \ldots, \Phi^{(n)}(\cdot, \boldsymbol{x}_N)\}$ *provides new — approximately cardinal — basis for* $\mathrm{span}\{\Phi(\cdot, \boldsymbol{x}_1), \ldots, \Phi(\cdot, \boldsymbol{x}_N)\}$.

Theorem

Part I (without acceleration)

$$\mathcal{Q}_f^{(n)} = \boldsymbol{\Phi}^T \underbrace{\sum_{k=0}^{n} (I - A)^k}_{=P^{(n)}} \boldsymbol{f} =: \boldsymbol{\Phi}^{(n)T} \boldsymbol{f},$$

i.e., $\{\Phi^{(n)}(\cdot, \boldsymbol{x}_1), \ldots, \Phi^{(n)}(\cdot, \boldsymbol{x}_N)\}$ *provides new — approximately cardinal — basis for* $\text{span}\{\Phi(\cdot, \boldsymbol{x}_1), \ldots, \Phi(\cdot, \boldsymbol{x}_N)\}$.

Recursion for preconditioner

$$P^{(n+1)} = I + P^{(n)} (I - A), \qquad P^{(0)} = I.$$

Right Preconditioning without Acceleration

Since $A\sum_{k=0}^{n}(I-A)^k \to I$, we take $P^{(n)} = \sum_{k=0}^{n}(I-A)^k$

Right Preconditioning without Acceleration

Since $A\sum_{k=0}^{n}(I-A)^k \to I$, we take $P^{(n)} = \sum_{k=0}^{n}(I-A)^k$

1. $P^{(0)} = I$
2. For $n = 1, 2, 3, \ldots$
$$P^{(n)} = I + P^{(n-1)}(I-A)$$
3. Solve $\left(AP^{(n)}\right)\boldsymbol{c} = \boldsymbol{f}$ for $\boldsymbol{c}$

Right Preconditioning without Acceleration

Since $A\sum_{k=0}^{n}(I-A)^k \to I$, we take $P^{(n)} = \sum_{k=0}^{n}(I-A)^k$

1. $P^{(0)} = I$
2. For $n = 1, 2, 3, \ldots$

$$P^{(n)} = I + P^{(n-1)}(I-A)$$

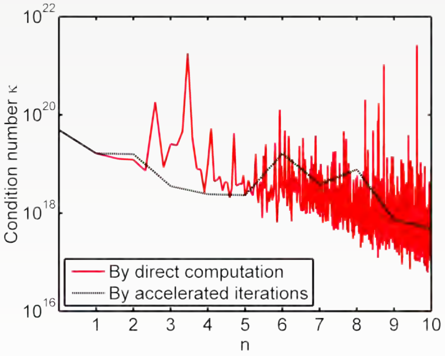
3. Solve $\left(AP^{(n)}\right)\boldsymbol{c} = \boldsymbol{f}$ for $\boldsymbol{c}$

Evaluation on $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_M\} \subset \mathbb{R}^s$:

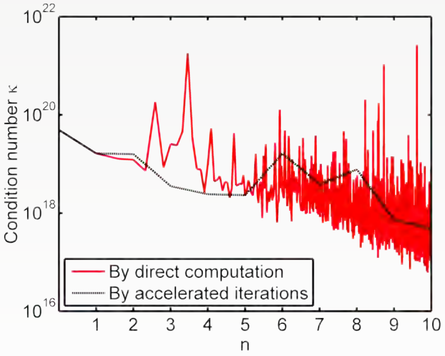
$$\mathcal{P}_f(\boldsymbol{y}) = \left(BP^{(n)}\right)\boldsymbol{c}$$

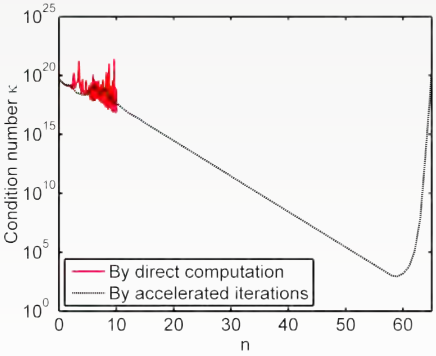
with $B_{ij} = \Phi(\boldsymbol{y}_i, \boldsymbol{x}_j), \quad i = 1, \ldots, M, \; j = 1, \ldots, N.$

Condition number after $2^n - 1$ iterations



Condition number after $2^n - 1$ iterations





Theorem

Part II (with acceleration)

$$\widetilde{\mathcal{Q}}_f^{(n)} = \boldsymbol{\Phi}^T \underbrace{\left[\sum_{k=0}^{2^n-1} (I - A)^k\right]}_{=P^{(n)}} \boldsymbol{f} =: \boldsymbol{\Phi}^{(n)T} \boldsymbol{f},$$

i.e., $\{\Phi^{(n)}(\cdot, \boldsymbol{x}_1), \ldots, \Phi^{(n)}(\cdot, \boldsymbol{x}_N)\}$ *provides new — approximately cardinal — basis for* $\operatorname{span}\{\Phi(\cdot, \boldsymbol{x}_1), \ldots, \Phi(\cdot, \boldsymbol{x}_N)\}$.

Theorem

Part II (with acceleration)

$$\widetilde{\mathcal{Q}}_f^{(n)} = \boldsymbol{\Phi}^T \underbrace{\left[\sum_{k=0}^{2^n-1} (I-A)^k\right]}_{=P^{(n)}} \boldsymbol{f} =: \boldsymbol{\Phi}^{(n)T} \boldsymbol{f},$$

i.e., $\{\Phi^{(n)}(\cdot,\boldsymbol{x}_1),\ldots,\Phi^{(n)}(\cdot,\boldsymbol{x}_N)\}$ *provides new — approximately cardinal — basis for* $\mathrm{span}\{\Phi(\cdot,\boldsymbol{x}_1),\ldots,\Phi(\cdot,\boldsymbol{x}_N)\}$.

Recursion for preconditioner

$$P^{(n+1)} = P^{(n)}\left[2I - AP^{(n)}\right], \qquad P^{(0)} = I.$$

Right Preconditioning with Acceleration

Now use $P^{(n)} = \sum_{k=0}^{2^n-1} (I - A)^k$

Right Preconditioning with Acceleration

Now use $P^{(n)} = \sum_{k=0}^{2^n-1} (I - A)^k$

1. $P^{(0)} = I, \qquad A_P^{(0)} = A$
2. For $n = 1, 2, 3, \ldots$

$$P^{(n)} = P^{(n-1)} \left(2I - A_P^{(n-1)}\right), \qquad A_P^{(n)} = AP^{(n)}$$

3. Solve $A_P^{(n)} \boldsymbol{c} = \boldsymbol{f}$ for $\boldsymbol{c}$

Right Preconditioning with Acceleration

Now use $P^{(n)} = \sum_{k=0}^{2^n-1} (I - A)^k$

1. $P^{(0)} = I, \qquad A_P^{(0)} = A$
2. For $n = 1, 2, 3, \ldots$
$$P^{(n)} = P^{(n-1)} \left(2I - A_P^{(n-1)}\right), \qquad A_P^{(n)} = AP^{(n)}$$
3. Solve $A_P^{(n)} \boldsymbol{c} = \boldsymbol{f}$ for $\boldsymbol{c}$

Evaluation on $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_M\} \subset \mathbb{R}^s$:
$$\mathcal{P}_f(\boldsymbol{y}) = BP^{(n)}\boldsymbol{c}$$
with $B_{ij} = \Phi(\boldsymbol{y}_i, \boldsymbol{x}_j), \quad i = 1, \ldots, M, \; j = 1, \ldots, N.$

Program (`IAMLSPrecond_sD.m`)

```
s = 2;  N = 289;  M = 500;  maxn = 50;
global rbf;  rbf_definition;  D = 2*s;
[dsites, N] = CreatePoints(N,s,'h');
ctrs = dsites;
epoints = CreatePoints(M,s,'r');
rhs = testfunctionsD(dsites);
h = 1/(nthroot(N,s)-1); ep = 1/(sqrt(D)*h);
DM_data = DistanceMatrix(dsites,ctrs);
IM = rbf(ep,DM_data)/(sqrt(pi*D)^s);
DM_eval = DistanceMatrix(epoints,ctrs);
EM = rbf(ep,DM_eval)/(sqrt(pi*D)^s);
P = eye(N);  AP = IM*P;
for n=1:maxn
    P = P*(2*eye(N)-AP);    AP = IM*P;
end
c = gmres(AP, rhs, [], 1e-10, 40);   % or use pcg
Pf = (EM*P) * c;
exact = testfunctionsD(epoints);
maxerr = norm(Pf-exact,inf)
rms_err = norm(Pf-exact)/sqrt(M)
```

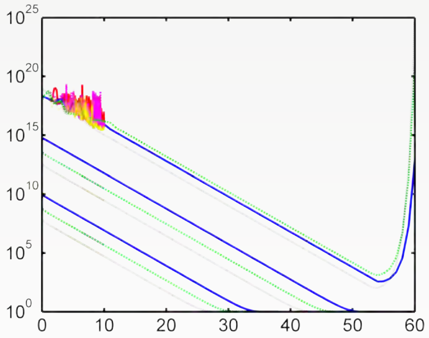
Comparison with [Beatson *et al.* (1999)]

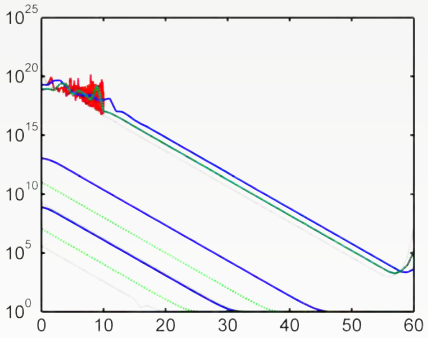
Cond. No.	289		1089		4225	
	no pre	pre	no pre	pre	no pre	pre
MQ BCM	1.506(8)	5.742(1)	2.154(9)	2.995(3)	3.734(10)	4.369(4)
TPS BCM	4.005(6)	3.330(0)	2.753(8)	1.411(2)	2.605(9)	2.025(3)
Gauss	8.796(9)	1.000(0)	6.849(10)	1.000(0)	7.632(10)	1.000(0)
IQ	1.186(8)	1.000(0)	4.284(8)	1.000(0)	1.082(9)	1.000(0)

No.	289		1089		4225	
GMRES iter.	no pre	pre	no pre	pre	no pre	pre
MQ BCM	145	8	>150	15	>150	28
TPS BCM	103	5	145	6	>150	9
Gauss	>150	2	>150	2	>150	2
IQ	>150	2	>150	2	>150	2

2D Halton points, $n = 40$, $\varepsilon_G = 6.4, 12.8, 25.6$, $\varepsilon_I = 3.2, 6.4, 12.8$

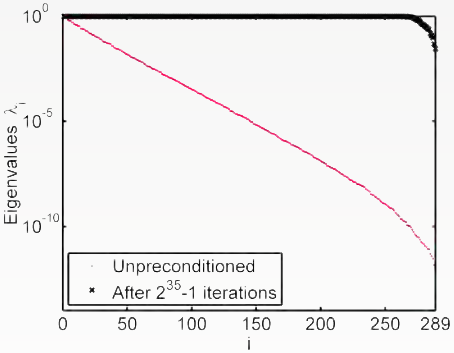
Condition number drop

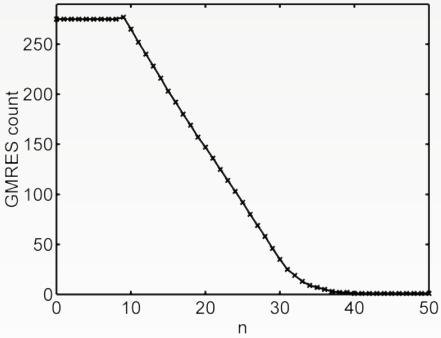




Laguerre-Gaussians (left, with $\varepsilon = 3.2, 4.8, 6.4$) and generalized IMQs (right, with $\varepsilon = 0.204, 2.04, 3.06$) for $d = 0, 1, 2$ and $N = 289$ Halton points

Eigenvalue distribution & GMRES convergence





Gaussian with $\varepsilon = 5.95$ and $N = 289$ Halton points

Summary of IAMLS Preconditioning

- Current implementation too slow to be useful (matrix-matrix product)
- Preconditioning with accelerated iterated AMLS very effective for “good” (i.e., reasonably conditioned) problems
- Preconditioner does not perform well for “bad” problems (cond(AP) can be made small, but P itself becomes ill-conditioned, so that evaluation unreliable)
- Automatic stopping criterion in [F. & Zhang (2008)]
- Accelerated iterated AMLS (as well as SVD) may be used to approximately solve “bad” problems or problems with noise
- Generalized inverse MQs and Laguerre-Gaussians seem to behave similarly

Our Latest Approach — Riley's Algorithm [Riley (1955)]

Instead of solving the ill-conditioned system

$$A\boldsymbol{x} = \boldsymbol{b}$$

we regularize, i.e., let

$$C = A + \mu I$$

and solve

$$C\boldsymbol{y} = \boldsymbol{b}.$$

Our Latest Approach — Riley's Algorithm [Riley (1955)]

Instead of solving the ill-conditioned system

$$A\boldsymbol{x} = \boldsymbol{b}$$

we regularize, i.e., let

$$C = A + \mu I$$

and solve

$$C\boldsymbol{y} = \boldsymbol{b}.$$

Note: If A is symmetric positive definite, so is C.

Our Latest Approach — Riley's Algorithm [Riley (1955)]

Instead of solving the ill-conditioned system

$$A\boldsymbol{x} = \boldsymbol{b}$$

we regularize, i.e., let

$$C = A + \mu I$$

and solve

$$C\boldsymbol{y} = \boldsymbol{b}.$$

Note: If A is symmetric positive definite, so is C.

This is well-known as Tikhonov regularization or ridge regression.

Our Latest Approach — Riley's Algorithm [Riley (1955)]

Instead of solving the ill-conditioned system

$$A\boldsymbol{x} = \boldsymbol{b}$$

we regularize, i.e., let

$$C = A + \mu I$$

and solve

$$C\boldsymbol{y} = \boldsymbol{b}.$$

Note: If A is symmetric positive definite, so is C.

This is well-known as Tikhonov regularization or ridge regression.

But this is not the end of Riley's algorithm.

Since $A = C - \mu I$ we can derive

$$A^{-1} = \frac{1}{\mu} \sum_{k=1}^{\infty} \left(\mu C^{-1}\right)^k \tag{1}$$

and therefore we get the solution to the original system as

$$\boldsymbol{x} = A^{-1}\boldsymbol{b}$$

Since $A = C - \mu I$ we can derive

$$A^{-1} = \frac{1}{\mu} \sum_{k=1}^{\infty} \left(\mu C^{-1}\right)^k \tag{1}$$

and therefore we get the solution to the original system as

$$\begin{aligned} \boldsymbol{x} &= A^{-1}\boldsymbol{b} \\ &\overset{(1)}{=} \frac{1}{\mu} \sum_{k=1}^{\infty} \left(\mu C^{-1}\right)^k \boldsymbol{b} \end{aligned}$$

Since $A = C - \mu I$ we can derive

$$A^{-1} = \frac{1}{\mu} \sum_{k=1}^{\infty} \left(\mu C^{-1}\right)^{k} \tag{1}$$

and therefore we get the solution to the original system as

$$\begin{aligned} \boldsymbol{x} &= A^{-1}\boldsymbol{b} \\ &\overset{(1)}{=} \frac{1}{\mu} \sum_{k=1}^{\infty} \left(\mu C^{-1}\right)^{k} \boldsymbol{b} \\ &\overset{\boldsymbol{y}=C^{-1}\boldsymbol{b}}{=} \sum_{k=1}^{\infty} \left(\mu C^{-1}\right)^{k-1} \boldsymbol{y} \end{aligned}$$

Since $A = C - \mu I$ we can derive

$$A^{-1} = \frac{1}{\mu}\sum_{k=1}^{\infty}\left(\mu C^{-1}\right)^k \tag{1}$$

and therefore we get the solution to the original system as

$$\begin{aligned}
\boldsymbol{x} &= A^{-1}\boldsymbol{b} \\
&\overset{(1)}{=} \frac{1}{\mu}\sum_{k=1}^{\infty}\left(\mu C^{-1}\right)^k \boldsymbol{b} \\
&\overset{\boldsymbol{y}=C^{-1}\boldsymbol{b}}{=} \sum_{k=1}^{\infty}\left(\mu C^{-1}\right)^{k-1} \boldsymbol{y} \\
&= \boldsymbol{y} + \underbrace{\mu C^{-1}\boldsymbol{y} + \left(\mu C^{-1}\right)^2 \boldsymbol{y} + \ldots}_{=\mu C^{-1}\left[\boldsymbol{y}+\mu C^{-1}\boldsymbol{y}+\ldots\right]}
\end{aligned}$$

Since $A = C - \mu I$ we can derive

$$A^{-1} = \frac{1}{\mu} \sum_{k=1}^{\infty} \left(\mu C^{-1}\right)^k \qquad (1)$$

and therefore we get the solution to the original system as

$$\begin{aligned}
\boldsymbol{x} &= A^{-1}\boldsymbol{b} \\
&\overset{(1)}{=} \frac{1}{\mu} \sum_{k=1}^{\infty} \left(\mu C^{-1}\right)^k \boldsymbol{b} \\
&\overset{\boldsymbol{y}=C^{-1}\boldsymbol{b}}{=} \sum_{k=1}^{\infty} \left(\mu C^{-1}\right)^{k-1} \boldsymbol{y} \\
&= \boldsymbol{y} + \underbrace{\mu C^{-1}\boldsymbol{y} + \left(\mu C^{-1}\right)^2 \boldsymbol{y} + \ldots}_{=\mu C^{-1}\left[\boldsymbol{y}+\mu C^{-1}\boldsymbol{y}+\ldots\right]}
\end{aligned}$$

So

$$\boldsymbol{x}_{k+1} = \boldsymbol{y} + \mu C^{-1}\boldsymbol{x}_k, \qquad k = 0, 1, 2, \ldots, \qquad \boldsymbol{x}_0 = \boldsymbol{0} \qquad (2)$$

Alternative version of Riley's Algorithm

From above

$$\boldsymbol{x} = \boldsymbol{y} + \mu C^{-1}\boldsymbol{y} + \left(\mu C^{-1}\right)^2 \boldsymbol{y} + \ldots$$

Alternative version of Riley's Algorithm

From above

$$\boldsymbol{x} = \boldsymbol{y} + \mu C^{-1}\boldsymbol{y} + \left(\mu C^{-1}\right)^2 \boldsymbol{y} + \dots$$

So

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \left(\mu C^{-1}\right)^k \boldsymbol{y}, \qquad k = 0, 1, 2, \dots, \quad \boldsymbol{x}_0 = \boldsymbol{0}$$

Alternative version of Riley's Algorithm

From above

$$\boldsymbol{x} = \boldsymbol{y} + \mu C^{-1}\boldsymbol{y} + \left(\mu C^{-1}\right)^2 \boldsymbol{y} + \ldots$$

So

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \left(\mu C^{-1}\right)^k \boldsymbol{y}, \qquad k = 0, 1, 2, \ldots, \quad \boldsymbol{x}_0 = \boldsymbol{0}$$

Compute product iteratively:

$$\begin{aligned} \boldsymbol{x}_0 &= \boldsymbol{0} \\ \boldsymbol{y}_0 &= C^{-1}\boldsymbol{b} \\ \boldsymbol{y}_{k+1} &= \mu C^{-1}\boldsymbol{y}_{k-1} \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \boldsymbol{y}_k, \qquad k = 0, 1, 2, \ldots \end{aligned}$$

This is our main version of Riley's algorithm

Riley’s Algorithm in MATLAB

```
function x = Riley(A,b,mu)
    C = A + mu*eye(size(A));
    L = chol(C,'lower');
    z = L\b;
    y = L'\z;
    x = y;
    for k = 1:kend
        z = mu*(L\y);
        y = L'\z;
        x = x + y;
    end
end
```

Another Interpretation of Riley's Algorithm

Let $C = A + \mu I$ as before. Then

$$A\boldsymbol{x} = \boldsymbol{b} \quad \Longleftrightarrow \quad (C - \mu I)\boldsymbol{x} = \boldsymbol{b}.$$

Another Interpretation of Riley's Algorithm

Let $C = A + \mu I$ as before. Then

$$A\boldsymbol{x} = \boldsymbol{b} \quad \Longleftrightarrow \quad (C - \mu I)\boldsymbol{x} = \boldsymbol{b}.$$

Now split A and iterate

$$C\boldsymbol{x}_{k+1} = \boldsymbol{b} + \mu \boldsymbol{x}_k, \quad k = 0, 1, 2, \ldots \tag{3}$$

where $\boldsymbol{x}_0 = \boldsymbol{0}$.

Another Interpretation of Riley's Algorithm

Let $C = A + \mu I$ as before. Then

$$A\boldsymbol{x} = \boldsymbol{b} \quad \Longleftrightarrow \quad (C - \mu I)\boldsymbol{x} = \boldsymbol{b}.$$

Now split A and iterate

$$C\boldsymbol{x}_{k+1} = \boldsymbol{b} + \mu \boldsymbol{x}_k, \quad k = 0, 1, 2, \ldots \tag{3}$$

where $\boldsymbol{x}_0 = \boldsymbol{0}$.

This corresponds to

$$\boldsymbol{x}_{k+1} = \boldsymbol{y} + \mu C^{-1}\boldsymbol{x}_k, \qquad k = 0, 1, 2, \ldots, \quad \boldsymbol{x}_0 = \boldsymbol{0},\ \boldsymbol{y} = C^{-1}\boldsymbol{b}$$

which is the same as (2).

[Golub (1965)] showed this is equivalent to iterative improvement (see [Kincaid and Cheney (2002)]):

$$C\boldsymbol{e} = \boldsymbol{b} - A\boldsymbol{x}_k \tag{4}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{e} \tag{5}$$

also with $\boldsymbol{x}_0 = \boldsymbol{0}$.

[Golub (1965)] showed this is equivalent to iterative improvement (see [Kincaid and Cheney (2002)]):

$$\begin{aligned} C\boldsymbol{e} &= \boldsymbol{b} - A\boldsymbol{x}_k && (4) \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \boldsymbol{e} && (5) \end{aligned}$$

also with $\boldsymbol{x}_0 = \boldsymbol{0}$.

Verification of equivalence

$$C\boldsymbol{x}_{k+1} \overset{(5)}{=} C\boldsymbol{x}_k + C\boldsymbol{e}$$

[Golub (1965)] showed this is equivalent to iterative improvement (see [Kincaid and Cheney (2002)]):

$$\begin{aligned} C\boldsymbol{e} &= \boldsymbol{b} - A\boldsymbol{x}_k \qquad (4) \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \boldsymbol{e} \qquad (5) \end{aligned}$$

also with $\boldsymbol{x}_0 = \boldsymbol{0}$.

Verification of equivalence

$$\begin{aligned} C\boldsymbol{x}_{k+1} &\overset{(5)}{=} C\boldsymbol{x}_k + C\boldsymbol{e} \\ &\overset{(4)}{=} C\boldsymbol{x}_k + \boldsymbol{b} - A\boldsymbol{x}_k \end{aligned}$$

[Golub (1965)] showed this is equivalent to iterative improvement (see [Kincaid and Cheney (2002)]):

$$\begin{aligned} C\boldsymbol{e} &= \boldsymbol{b} - A\boldsymbol{x}_k & (4) \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \boldsymbol{e} & (5) \end{aligned}$$

also with $\boldsymbol{x}_0 = \boldsymbol{0}$.

Verification of equivalence

$$\begin{aligned} C\boldsymbol{x}_{k+1} &\overset{(5)}{=} C\boldsymbol{x}_k + C\boldsymbol{e} \\ &\overset{(4)}{=} C\boldsymbol{x}_k + \boldsymbol{b} - A\boldsymbol{x}_k \\ &\overset{C=A+\mu I}{=} \boldsymbol{b} + \mu I\boldsymbol{x}_k \end{aligned}$$

[Golub (1965)] showed this is equivalent to iterative improvement (see [Kincaid and Cheney (2002)]):

$$\begin{aligned} C\boldsymbol{e} &= \boldsymbol{b} - A\boldsymbol{x}_k & (4) \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \boldsymbol{e} & (5) \end{aligned}$$

also with $\boldsymbol{x}_0 = \boldsymbol{0}$.

Verification of equivalence

$$\begin{aligned} C\boldsymbol{x}_{k+1} &\overset{(5)}{=} C\boldsymbol{x}_k + C\boldsymbol{e} \\ &\overset{(4)}{=} C\boldsymbol{x}_k + \boldsymbol{b} - A\boldsymbol{x}_k \\ &\overset{C=A+\mu I}{=} \boldsymbol{b} + \mu I\boldsymbol{x}_k \overset{(3)}{=} C\boldsymbol{x}_{k+1} \end{aligned}$$

[Golub (1965)] showed this is equivalent to iterative improvement (see [Kincaid and Cheney (2002)]):

$$\begin{aligned} C\boldsymbol{e} &= \boldsymbol{b} - A\boldsymbol{x}_k && (4) \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \boldsymbol{e} && (5) \end{aligned}$$

also with $\boldsymbol{x}_0 = \boldsymbol{0}$.

Verification of equivalence

$$\begin{aligned} C\boldsymbol{x}_{k+1} &\overset{(5)}{=} C\boldsymbol{x}_k + C\boldsymbol{e} \\ &\overset{(4)}{=} C\boldsymbol{x}_k + \boldsymbol{b} - A\boldsymbol{x}_k \\ &\overset{C=A+\mu I}{=} \boldsymbol{b} + \mu I\boldsymbol{x}_k \overset{(3)}{=} C\boldsymbol{x}_{k+1} \end{aligned}$$

Remark

[Neumaier (1998)] calls this method iterated Tikhonov regularization or preconditioned Landweber iteration (but attributes it to [Riley (1955)]).

Iterative Improvement Version of Riley's Algorithm in MATLAB

```
function x = Riley_Residuals(A,b,mu)
    C = A + mu*eye(size(A));
    L = chol(C,'lower');
    z = L\b;
    x = L'\z;
    for k=1:kend
        z = L\(b - A*x);
        x = x + L'\z;
    end
end
```

Major problem with Tikhonov regularization:

How should we choose μ?

Major problem with Tikhonov regularization:

How should we choose μ?

Usual approach: cross validation or maximum likelihood

Major problem with Tikhonov regularization:

How should we choose μ?

Usual approach: cross validation or maximum likelihood

Practical suggestion in [Riley (1955)]: choose μ small, i.e.,

$$\mu \approx 10^{-p+\alpha},$$

where p is desired precision, and $\alpha = 2$ or 3.

Convergence of the algorithm

The eigenvalues of μC^{-1} are $0 < \frac{\mu}{\lambda_i+\mu} < 1$, so the series

$$\boldsymbol{x} = \sum_{k=0}^{\infty} \left(\mu C^{-1}\right)^k \boldsymbol{y}$$

converges.

Convergence of the algorithm

The eigenvalues of μC^{-1} are $0 < \frac{\mu}{\lambda_i+\mu} < 1$, so the series

$$\boldsymbol{x} = \sum_{k=0}^{\infty} \left(\mu C^{-1}\right)^k \boldsymbol{y}$$

converges.
For $\mu \ll \lambda_{\min}$ we have fast convergence.

Convergence of the algorithm

The eigenvalues of μC^{-1} are $0 < \frac{\mu}{\lambda_i+\mu} < 1$, so the series

$$\boldsymbol{x} = \sum_{k=0}^{\infty} \left(\mu C^{-1}\right)^k \boldsymbol{y}$$

converges.

For $\mu \ll \lambda_{\min}$ we have fast convergence.

The matrix C is better conditioned than A since

$$\mathrm{cond}(C) = \frac{\lambda_{\max} + \mu}{\lambda_{\min} + \mu} \ll \mathrm{cond}(A) = \frac{\lambda_{\max}}{\lambda_{\min}}$$

provided $\mu > \lambda_{\min}$.

Convergence of the algorithm

The eigenvalues of μC^{-1} are $0 < \frac{\mu}{\lambda_i+\mu} < 1$, so the series

$$\boldsymbol{x} = \sum_{k=0}^{\infty} \left(\mu C^{-1}\right)^k \boldsymbol{y}$$

converges.
For $\mu \ll \lambda_{\min}$ we have fast convergence.
The matrix C is better conditioned than A since

$$\operatorname{cond}(C) = \frac{\lambda_{\max} + \mu}{\lambda_{\min} + \mu} \ll \operatorname{cond}(A) = \frac{\lambda_{\max}}{\lambda_{\min}}$$

provided $\mu > \lambda_{\min}$.

Summary:

μ needs to be

- large enough to improve conditioning
- small enough to provide fast convergence

Recommended to use $\mu \approx \lambda_{\min}$ (we use $\mu = 10^{-11}$ when $\lambda_{\min} \approx 10^{-17}$)

SVD solution of $A\boldsymbol{x} = \boldsymbol{b}$:

$$\boldsymbol{x} = \sum_{j=1}^{r} \frac{\beta_j}{\sigma_j} \boldsymbol{v}_j,$$

where $A = U\Sigma V^T$ and $\beta = U^T\boldsymbol{b}$

SVD solution of $A\boldsymbol{x} = \boldsymbol{b}$:

$$\boldsymbol{x} = \sum_{j=1}^{r} \frac{\beta_j}{\sigma_j} \boldsymbol{v}_j,$$

where $A = U\Sigma V^T$ and $\beta = U^T \boldsymbol{b}$

- regularize by truncation of components associated with small singular values

SVD solution of $Ax = \boldsymbol{b}$:

$$\boldsymbol{x} = \sum_{j=1}^{r} \frac{\beta_j}{\sigma_j} \boldsymbol{v}_j,$$

where $A = U\Sigma V^T$ and $\beta = U^T\boldsymbol{b}$

- regularize by truncation of components associated with small singular values

Riley:

$$\boldsymbol{x}_k = \sum_{j=1}^{r} \left[1 - \left(\frac{\mu}{\mu + \sigma_j}\right)^k\right] \frac{\beta_j}{\sigma_j} \boldsymbol{v}_j,$$

where $C = A + \mu I$

SVD solution of $A\boldsymbol{x} = \boldsymbol{b}$:

$$\boldsymbol{x} = \sum_{j=1}^{r} \frac{\beta_j}{\sigma_j} \boldsymbol{v}_j,$$

where $A = U\Sigma V^T$ and $\beta = U^T\boldsymbol{b}$

- regularize by truncation of components associated with small singular values

Riley:

$$\boldsymbol{x}_k = \sum_{j=1}^{r} \left[1 - \left(\frac{\mu}{\mu + \sigma_j} \right)^k \right] \frac{\beta_j}{\sigma_j} \boldsymbol{v}_j,$$

where $C = A + \mu I$

- regularize by iteration (i.e., damping each mode)

Another look at the regularization parameters

Riley iteration: $$\boldsymbol{x}_k = \sum_{j=1}^{r} \left[1 - \left(\frac{\mu}{\mu + \sigma_j} \right)^k \right] \frac{\beta_j}{\sigma_j} \boldsymbol{v}_j$$

Assume $\mu \approx \sigma_r$ (balancing regularization and convergence)
Then
$$\left(\frac{\mu}{\mu + \sigma_r} \right)^k \approx \frac{1}{2^k}$$

Another look at the regularization parameters

Riley iteration: $$\boldsymbol{x}_k = \sum_{j=1}^{r} \left[1 - \left(\frac{\mu}{\mu + \sigma_j} \right)^k \right] \frac{\beta_j}{\sigma_j} \boldsymbol{v}_j$$

Assume $\mu \approx \sigma_r$ (balancing regularization and convergence)

Then

$$\left(\frac{\mu}{\mu + \sigma_r} \right)^k \approx \frac{1}{2^k}$$

So

$$1 - \left(\frac{\mu}{\mu + \sigma_r} \right)^k \approx 0.9 \qquad \text{if } k = 3 \text{ or } k = 4$$

and

$$1 - \left(\frac{\mu}{\mu + \sigma_r} \right)^k \approx 0.99 \qquad \text{for } k \approx 7$$

Another look at the regularization parameters

Riley iteration: $$\boldsymbol{x}_k = \sum_{j=1}^{r} \left[1 - \left(\frac{\mu}{\mu+\sigma_j}\right)^k\right] \frac{\beta_j}{\sigma_j} \boldsymbol{v}_j$$

Assume $\mu \approx \sigma_r$ (balancing regularization and convergence)

Then

$$\left(\frac{\mu}{\mu+\sigma_r}\right)^k \approx \frac{1}{2^k}$$

So

$$1 - \left(\frac{\mu}{\mu+\sigma_r}\right)^k \approx 0.9 \qquad \text{if } k = 3 \text{ or } k = 4$$

and

$$1 - \left(\frac{\mu}{\mu+\sigma_r}\right)^k \approx 0.99 \qquad \text{for } k \approx 7$$

Note: $1 - \left(\frac{\mu}{\mu+\sigma_j}\right)^k$, $j < r$ is even closer to 1

Obtaining a Stable Basis via SVD

Consider a stacked interpolation and evaluation system:

$$\begin{bmatrix} A \\ B \end{bmatrix} \boldsymbol{c} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{s} \end{bmatrix}$$

with $A_{ij} = \Phi(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and $B_{kj} = \Phi(\boldsymbol{y}_k, \boldsymbol{x}_j)$, where $\boldsymbol{x}_i$ are centers = data sites, $\boldsymbol{y}_k$ are evaluation points

Obtaining a Stable Basis via SVD

Consider a stacked interpolation and evaluation system:

$$\begin{bmatrix} A \\ B \end{bmatrix} \boldsymbol{c} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{s} \end{bmatrix}$$

with $A_{ij} = \Phi(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and $B_{kj} = \Phi(\boldsymbol{y}_k, \boldsymbol{x}_j)$, where $\boldsymbol{x}_i$ are centers = data sites, $\boldsymbol{y}_k$ are evaluation points

Apply SVD:

$$\begin{bmatrix} A \\ B \end{bmatrix} \boldsymbol{c} = U \underbrace{\Sigma V^T \boldsymbol{c}}_{=\boldsymbol{d}} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{s} \end{bmatrix}$$

Obtaining a Stable Basis via SVD

Consider a stacked interpolation and evaluation system:

$$\begin{bmatrix} A \\ B \end{bmatrix} \boldsymbol{c} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{s} \end{bmatrix}$$

with $A_{ij} = \Phi(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and $B_{kj} = \Phi(\boldsymbol{y}_k, \boldsymbol{x}_j)$, where $\boldsymbol{x}_i$ are centers = data sites, $\boldsymbol{y}_k$ are evaluation points

Apply SVD:

$$\begin{bmatrix} A \\ B \end{bmatrix} \boldsymbol{c} = U \underbrace{\Sigma V^T \boldsymbol{c}}_{=\boldsymbol{d}} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{s} \end{bmatrix}$$

Now:

$$A\boldsymbol{c} = U(1{:}\mathrm{N}, 1{:}\mathrm{r})\boldsymbol{d} = \boldsymbol{f} \quad \Longrightarrow \quad \boldsymbol{d} = U^{\dagger}(1{:}\mathrm{N}, 1{:}\mathrm{r})\boldsymbol{f}$$

Obtaining a Stable Basis via SVD

Consider a stacked interpolation and evaluation system:

$$\begin{bmatrix} A \\ B \end{bmatrix} \boldsymbol{c} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{s} \end{bmatrix}$$

with $A_{ij} = \Phi(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and $B_{kj} = \Phi(\boldsymbol{y}_k, \boldsymbol{x}_j)$, where $\boldsymbol{x}_i$ are centers = data sites, $\boldsymbol{y}_k$ are evaluation points

Apply SVD:

$$\begin{bmatrix} A \\ B \end{bmatrix} \boldsymbol{c} = U \underbrace{\Sigma V^T \boldsymbol{c}}_{=\boldsymbol{d}} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{s} \end{bmatrix}$$

Now:

$$A\boldsymbol{c} = U(1{:}\mathrm{N}, 1{:}\mathrm{r})\boldsymbol{d} = \boldsymbol{f} \quad \Longrightarrow \quad \boldsymbol{d} = U^{\dagger}(1{:}\mathrm{N}, 1{:}\mathrm{r})\boldsymbol{f}$$

$$\begin{aligned} \boldsymbol{s} &= B\boldsymbol{c} = U(\mathrm{N}{+}1{:}\mathrm{end}, 1{:}\mathrm{r})\boldsymbol{d} \\ &= U(\mathrm{N}{+}1{:}\mathrm{end}, 1{:}\mathrm{r})U^{\dagger}(1{:}\mathrm{N}, 1{:}\mathrm{r})\boldsymbol{f} \end{aligned}$$

The following quotes from [Golub (1965)] lead up to Riley's algorithm:

> . . . *results of an extensive calculation. The matrix consists of the first 5 columns of the inverse of the* 6×6 *Hilbert matrix.*

The following quotes from [Golub (1965)] lead up to Riley's algorithm:

> *. . . results of an extensive calculation. The matrix consists of the first 5 columns of the inverse of the 6×6 Hilbert matrix.*
>
> *For many problems, even with the use of orthogonal transformations it may be impossible to obtain an accurate solution.*

The following quotes from [Golub (1965)] lead up to Riley's algorithm:

> *. . . results of an extensive calculation. The matrix consists of the first 5 columns of the inverse of the* 6×6 *Hilbert matrix.*
>
> *For many problems, even with the use of orthogonal transformations it may be impossible to obtain an accurate solution.*

From the MathSciNet review of [Golub (1965)]:

> *This essentially consists in a more effective implementation of J. D. Riley's algorithm.*

In our experiments with $N = 900$ (either equally spaced or Halton) and $\mu = 10^{-11}$ we get

$$\frac{\mathrm{cond}(A)}{\mathrm{cond}(C)} \approx 10^6 - 10^8$$

for our ill-conditioned problems (i.e., with $\mathrm{cond}(A) \approx 10^{20}$).

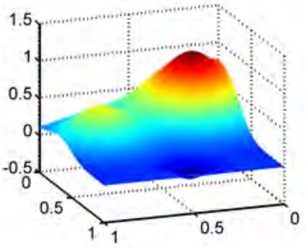
In our experiments with $N = 900$ (either equally spaced or Halton) and $\mu = 10^{-11}$ we get

$$\frac{\mathrm{cond}(A)}{\mathrm{cond}(C)} \approx 10^6 - 10^8$$

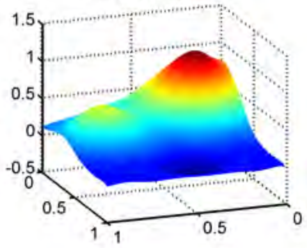
for our ill-conditioned problems (i.e., with $\mathrm{cond}(A) \approx 10^{20}$).

The timing for the Riley algorithm is only about 5% slower than standard `backslash` solver — and two orders of magnitude more accurate!

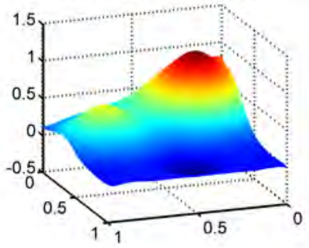
Franke test function



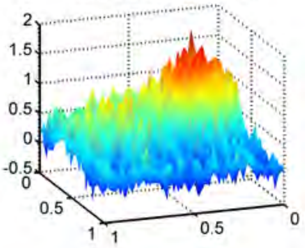
Riley approximation



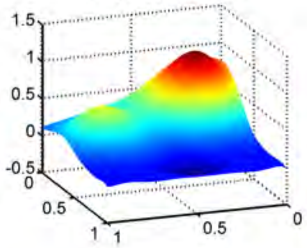
POD-SVD approximation



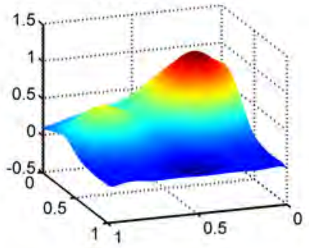
Standard approximation

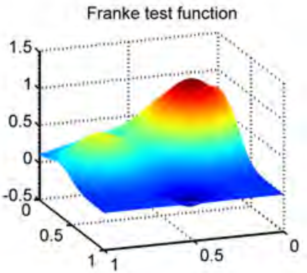


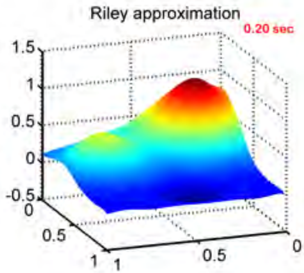
Riley2 approximation

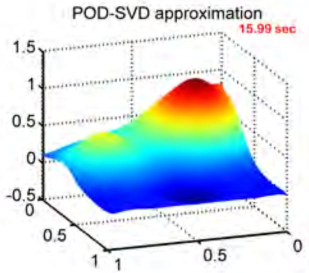


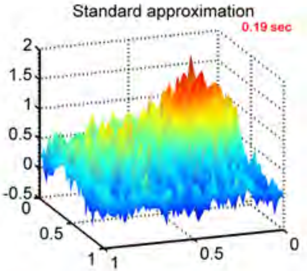
SVD approximation

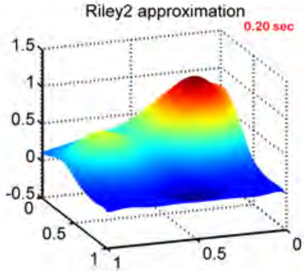


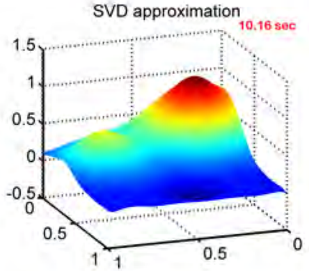

Franke test function
1.5
1
0.5
0
-0.5
0
0.5
1
1
0.5
0

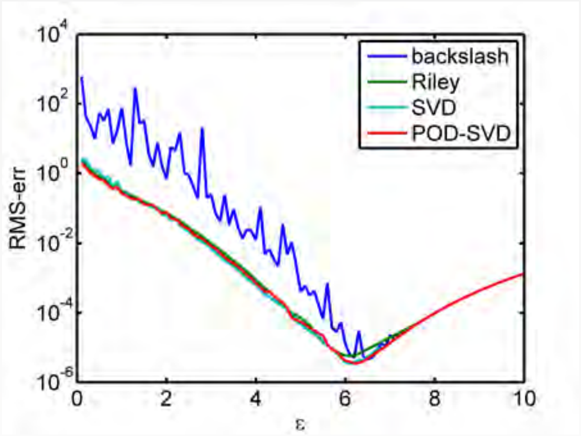

Riley approximation
0.20 sec
1.5
1
0.5
0
-0.5
0
0.5
1
1
0.5
0


POD-SVD approximation
15.99 sec
1.5
1
0.5
0
-0.5
0
0.5
1
1
0.5
0


Standard approximation
0.19 sec
2
1.5
1
0.5
0
-0.5
0
0.5
1
1
0.5
0

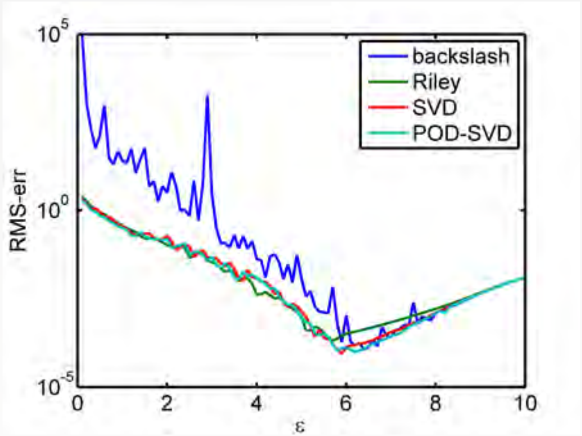

Riley2 approximation
0.20 sec
1.5
1
0.5
0
-0.5
0
0.5
1
1
0.5
0


SVD approximation
10.16 sec
1.5
1
0.5
0
-0.5
0
0.5
1
1
0.5
0



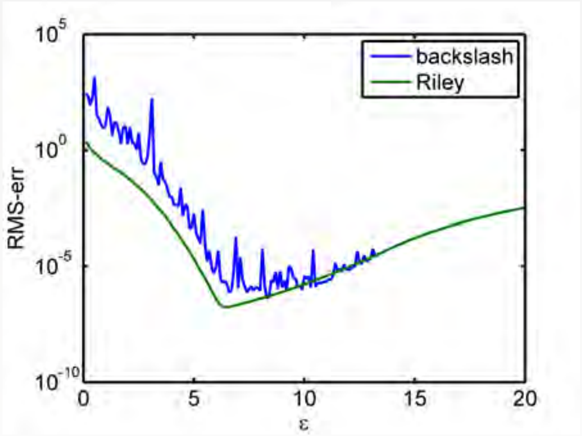
Method	backslash	Riley	SVD	POD-SVD
Time (sec)	24.1	26.0	710.0	1417.8

Table: Execution times for solution of 100 linear systems of size 900×900 (gridded data).



Method	backslash	Riley	SVD	POD-SVD
Time (sec)	24.0	26.7	739.1	1693.1

Table: Execution times for solution of 100 linear systems of size 900 $\times$ 900 (irregular data).



Method	backslash	Riley
Time (sec)	401.2	429.8

Table: Execution times for solution of 200 linear systems of size 2500×2500 (gridded data).

Summary

- Riley's paper seems largely forgotten/ignored
- Riley's algorithm seems to be superior to truncated SVD for ill-conditioned symmetric positive definite systems
- Riley's algorithm can be extended to
  - arbitrary (indefinite) square linear systems (use QR factorization of $C = A + \mu I$)
  - arbitrary non-square linear systems (use normal equations, and QR factorization of $C = A^T A + \mu I$)
  - sparse systems together with sparse Cholesky factorization (fast approximate sparse SVD)
- Riley's algorithm can be accelerated (e.g., by Aitken's method) — not yet done
- Need a strategy to find "optimal" regularization parameter μ

References I

- Buhmann, M. D. (2003).
  Radial Basis Functions: Theory and Implementations.
  Cambridge University Press.
- Kincaid, D. and Cheney, W. (2002).
  Numerical Analysis: Mathematics of Scientific Computing (3rd ed.).
  Brooks/Cole (Pacific Grove, CA).
- Fasshauer, G. E. (2007).
  Meshfree Approximation Methods with MATLAB.
  World Scientific Publishers.
- Higham, D. J. and Higham, N. J. (2005).
  MATLAB *Guide*.
  SIAM (2nd ed.), Philadelphia.
- Wendland, H. (2005).
  Scattered Data Approximation.
  Cambridge University Press.

References II

Ashby, S. F., Manteuffel, T. A. and Otto, J. S. (1992).
A comparison of adaptive Chebyshev and least squares polynomial preconditioning for Hermitian positive definite linear systems.
SIAM J. Sci. Statist. Comput. **13**, pp. 1–29.

Barba, L. A. and Rossi, Louis F. (2008).
Global field interpolation for particle methods.
preprint.

Baxter, B. J. C. (2002).
Preconditioned conjugate gradients, radial basis functions, and Toeplitz matrices.
Comput. Math. Appl. **43**, pp. 305–318.

Beatson, R. K., Cherrie, J. B. and Mouat, C. T. (1999).
Fast fitting of radial basis functions: methods based on preconditioned GMRES iteration.
Adv. Comput. Math. **11**, pp. 253–270.

References III

Beatson, R. K., Light, W. A. and Billings, S. (2000).
Fast solution of the radial basis function interpolation equations: domain decomposition methods.
SIAM J. Sci. Comput. **22**, pp. 1717–1740.

Benzi, M. (2002).
Preconditioning techniques for large linear systems: a survey.
J. Comput. Phys. **182**, pp. 418–477.

Brown, D., Ling, L., Kansa, E. and Levesley, J. (2005).
On approximate cardinal preconditioning methods for solving PDEs with radial basis functions.
Engineering Analysis with Boundary Elements **29**, pp. 343–353.

Cesari, L. (1937).
Sulla risoluzione dei sistemi di equazioni lineari per approssimazioni successive.
Ricerca sci., Roma **2** 8_{I}, pp. 512–522.

References IV

Chan, T. F. and Foulser, D. E. (1988).
Effectively well-conditioned linear systems.
SIAM J. Sci. Statist. Comput. **9**, pp. 963–969.

Dubois, P. F., Greenbaum, A. and Rodrigue, G. H. (1979).
Approximating the inverse of a matrix for use in iterative algorithms on vector processors.
Computing **22**, pp. 257–268.

Dyn, N. (1987).
Interpolation of scattered data by radial functions.
in *Topics in Multivariate Approximation*, C. K. Chui, L. L. Schumaker and F. Utreras (eds.), Academic Press (New York), pp. 47–61.

Dyn, N., Levin, D. and Rippa, S. (1986).
Numerical procedures for surface fitting of scattered data by radial functions.
SIAM J. Sci. Statist. Comput. **7**, pp. 639–659.

References V

Fasshauer, G. E. and Zhang, J. G. (2007).
On choosing "optimal" shape parameters for RBF approximation.
Numerical Algorithms **45**, pp. 345–368.

Fasshauer, G. E. and Zhang, J. G. (2008).
Preconditioning of radial basis function interpolation systems via accelerated iterated approximate moving least squares approximation.
in *Progress on Meshless Methods*, A. J. M. Ferreira, E. J. Kansa, G. E. Fasshauer and V. M. A. Leitão (eds.), Springer, pp. 57–76.

Golub, G. H. (1965).
Numerical methods for solving linear least-squares problems.
Numer. Math. **7**, pp. 206–216.

Ling, L. and Kansa, E. J. (2005).
A least-squares preconditioner for radial basis functions collocation methods.
Adv. Comp. Math. **23**, pp. 31–54.

References VI

Neumaier, A. (1998)
Solving ill-conditioned and singular linear systems: a tutorial on regularization.
SIAM Review **40**/3, pp. 636–666.

Riley, J. D. (1955).
Solving systems of linear equations with a positive definite, symmetric, but possibly ill-conditioned matrix.
Mathematical Tables and Other Aids to Computation **9**/51, pp. 96–101.

Rippa, S. (1999).
An algorithm for selecting a good value for the parameter *c* in radial basis function interpolation.
Adv. in Comput. Math. **11**, pp. 193–210.

Zhang, J. G. (2007).
Iterated Approximate Moving Least-Squares: Theory and Applications.
Ph.D. Dissertation, Illinois Institute of Technology.