Meshfree Approximation with MATLAB

Lecture V: "Optimal" Shape Parameters for RBF Approximation Methods

Greg Fasshauer

Department of Applied Mathematics
Illinois Institute of Technology

Dolomites Research Week on Approximation
September 8–11, 2008

Outline

Motivation

We saw earlier that the "correct" shape parameter ε plays a number of important roles:

- it determines the accuracy of the fit,
- it is important for numerical stability,
- it determines the speed of convergence,
- it is related to the saturation error of stationary approximation.

Motivation

We saw earlier that the "correct" shape parameter ε plays a number of important roles:

- it determines the accuracy of the fit,
- it is important for numerical stability,
- it determines the speed of convergence,
- it is related to the saturation error of stationary approximation.

In many applications the "best" ε is determined by "trial-and-error".

Motivation

We saw earlier that the "correct" shape parameter ε plays a number of important roles:

- it determines the accuracy of the fit,
- it is important for numerical stability,
- it determines the speed of convergence,
- it is related to the saturation error of stationary approximation.

In many applications the "best" ε is determined by "trial-and-error".

We now consider the use of cross validation.

Leave-One-Out Cross-Validation (LOOCV)

Proposed by [Rippa (1999)] (and already [Wahba (1990)] and [Dubrule '83]) for RBF interpolation systems $A\boldsymbol{c} = \boldsymbol{f}$

For a fixed $k = 1, \ldots, N$ and fixed ε, let

$$\mathcal{P}_f^{[k]}(\boldsymbol{x}, \varepsilon) = \sum_{\substack{j=1 \\ j \neq k}}^{N} c_j^{[k]} \Phi_\varepsilon(\boldsymbol{x}, \boldsymbol{x}_j)$$

be the RBF interpolant to training data $\{f_1, \ldots, f_{k-1}, f_{k+1}, \ldots, f_N\}$, i.e.,

$$\mathcal{P}_f^{[k]}(\boldsymbol{x}_i) = f_i, \qquad i = 1, \ldots, k-1, k+1, \ldots, N.$$

Leave-One-Out Cross-Validation (LOOCV)

Proposed by [Rippa (1999)] (and already [Wahba (1990)] and [Dubrule '83]) for RBF interpolation systems $A\boldsymbol{c} = \boldsymbol{f}$
For a fixed $k = 1, \ldots, N$ and fixed ε, let

$$\mathcal{P}_f^{[k]}(\boldsymbol{x}, \varepsilon) = \sum_{\substack{j=1 \\ j\neq k}}^{N} c_j^{[k]} \Phi_\varepsilon(\boldsymbol{x}, \boldsymbol{x}_j)$$

be the RBF interpolant to training data $\{f_1, \ldots, f_{k-1}, f_{k+1}, \ldots, f_N\}$, i.e.,

$$\mathcal{P}_f^{[k]}(\boldsymbol{x}_i) = f_i, \qquad i = 1, \ldots, k-1, k+1, \ldots, N.$$

Let

$$e_k(\varepsilon) = f_k - \mathcal{P}_f^{[k]}(\boldsymbol{x}_k, \varepsilon)$$

be the error at the one validation point $\boldsymbol{x}_k$ not used to determine the interpolant.

Leave-One-Out Cross-Validation (LOOCV)

Proposed by [Rippa (1999)] (and already [Wahba (1990)] and [Dubrule '83]) for RBF interpolation systems $A\boldsymbol{c} = \boldsymbol{f}$
For a fixed $k = 1, \ldots, N$ and fixed ε, let

$$\mathcal{P}_f^{[k]}(\boldsymbol{x}, \varepsilon) = \sum_{\substack{j=1 \\ j \neq k}}^{N} c_j^{[k]} \Phi_\varepsilon(\boldsymbol{x}, \boldsymbol{x}_j)$$

be the RBF interpolant to training data $\{f_1, \ldots, f_{k-1}, f_{k+1}, \ldots, f_N\}$, i.e.,

$$\mathcal{P}_f^{[k]}(\boldsymbol{x}_i) = f_i, \qquad i = 1, \ldots, k-1, k+1, \ldots, N.$$

Let

$$e_k(\varepsilon) = f_k - \mathcal{P}_f^{[k]}(\boldsymbol{x}_k, \varepsilon)$$

be the error at the one validation point $\boldsymbol{x}_k$ not used to determine the interpolant.
Find

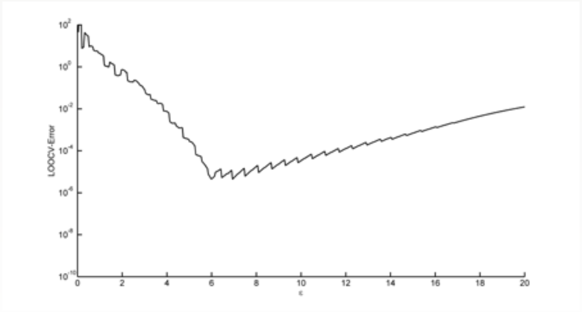
$$\varepsilon_{opt} = \underset{\varepsilon}{\operatorname{argmin}} \|\boldsymbol{e}(\varepsilon)\|, \qquad \boldsymbol{e} = [e_1, \ldots, e_N]^T$$

Naive approach

- Add a loop over ε
- Compare the error norms for different values of the shape parameter
- ε_{opt} is the one which yields the minimal error norm

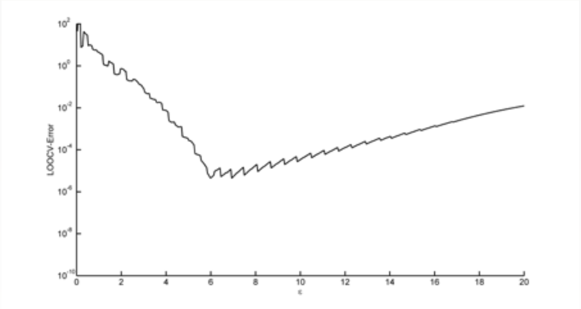
Naive approach

- Add a loop over ε
- Compare the error norms for different values of the shape parameter
- ε_{opt} is the one which yields the minimal error norm



Naive approach

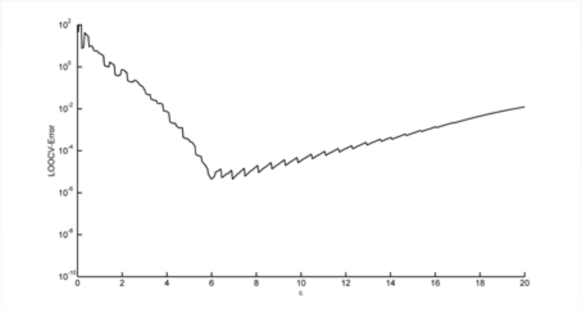
- Add a loop over ε
- Compare the error norms for different values of the shape parameter
- ε_{opt} is the one which yields the minimal error norm



Problem: computationally very expensive, i.e., $\mathcal{O}(N^4)$ operations

Naive approach

- Add a loop over ε
- Compare the error norms for different values of the shape parameter
- ε_{opt} is the one which yields the minimal error norm



Problem: computationally very expensive, i.e., $\mathcal{O}(N^4)$ operations
Advantage: does not require knowledge of the solution

Does this work?

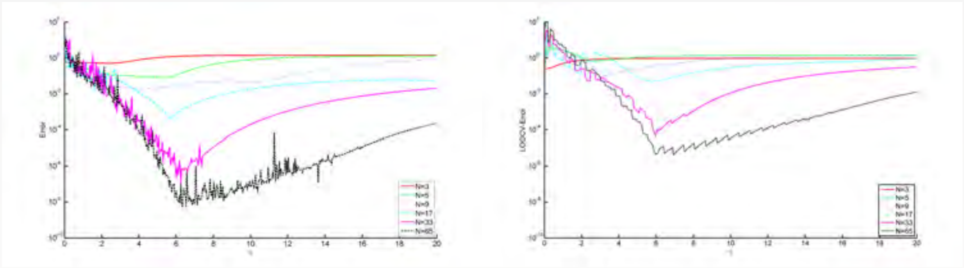


Figure: Optimal ε curves for trial and error (left) and for LOOCV (right) for 1D Gaussian interpolation.

N	3	5	9	17	33	65
trial-error	2.3246	5.1703	4.7695	5.6513	6.2525	6.5331
LOOCV	0.0401	0.0401	3.2064	5.7715	5.9319	6.9339

Table: Values of "optimal" ε.

A more efficient formula

Rippa (and also Wahba and Dubrule) showed that computation of the error components can be simplified to a single formula

$$e_k = \frac{c_k}{A_{kk}^{-1}},$$

where

- c_k: kth coefficient of full interpolant $\mathcal{P}_f$
- A_{kk}^{-1}: kth diagonal element of inverse of corresponding interpolation matrix

A more efficient formula

Rippa (and also Wahba and Dubrule) showed that computation of the error components can be simplified to a single formula

$$e_k = \frac{c_k}{A^{-1}_{kk}},$$

where

- c_k: kth coefficient of full interpolant $\mathcal{P}_f$
- A^{-1}_{kk}: kth diagonal element of inverse of corresponding interpolation matrix

Remark

- *c_k and A^{-1} need to be computed only once for each value of ε, so we still have $\mathcal{O}(N^3)$ computational complexity.*

A more efficient formula

Rippa (and also Wahba and Dubrule) showed that computation of the error components can be simplified to a single formula

$$e_k = \frac{c_k}{A^{-1}_{kk}},$$

where

- c_k: kth coefficient of full interpolant $\mathcal{P}_f$
- A^{-1}_{kk}: kth diagonal element of inverse of corresponding interpolation matrix

Remark

- *c_k and A^{-1} need to be computed only once for each value of ε, so we still have $\mathcal{O}(N^3)$ computational complexity.*
- *Can be vectorized in* MATLAB*:* `e = c./diag(inv(A))`*.*

LOOCV in MATLAB

- We can again use a naive approach and run a loop over many different values of ε.

LOOCV in MATLAB

- We can again use a naive approach and run a loop over many different values of ε.
- To be more efficient, we implement a "cost function", and then apply a minimization algorithm.

LOOCV in MATLAB

- We can again use a naive approach and run a loop over many different values of ε.
- To be more efficient, we implement a "cost function", and then apply a minimization algorithm.

Program (CostEps.m)

```
function ceps = CostEps(ep,r,rbf,rhs)
A = rbf(ep,r);
invA = pinv(A);
errorvector = (invA*rhs)./diag(invA);
ceps = norm(errorvector);
```

LOOCV in MATLAB

- We can again use a naive approach and run a loop over many different values of ε.
- To be more efficient, we implement a "cost function", and then apply a minimization algorithm.

Program (`CostEps.m`)

```
function ceps = CostEps(ep,r,rbf,rhs)
A = rbf(ep,r);
invA = pinv(A);
errorvector = (invA*rhs)./diag(invA);
ceps = norm(errorvector);
```

Possible calling sequence for the cost function:

```
ep = fminbnd(@(ep) CostEps(ep,DM,rbf,rhs),mine,maxe);
```

Program (`RBFInterpolation_sDLOOCV.m`)

```
s = 2;  N = 289;  gridtype = 'h';  M = 500;
global rbf;  rbf_definition;  mine = 0; maxe = 20;
[dsites, N] = CreatePoints(N,s,gridtype);
ctrs = dsites;
epoints = CreatePoints(M,s,'r');
rhs = testfunctionsD(dsites);
DM = DistanceMatrix(dsites,ctrs);
ep = fminbnd(@(ep) CostEps(ep,DM,rbf,rhs),mine,maxe);
IM = rbf(ep,DM);
DM = DistanceMatrix(epoints,ctrs);
EM = rbf(ep,DM);
Pf = EM * (IM\rhs);
exact = testfunctionsD(epoints);
maxerr = norm(Pf-exact,inf)
rms_err = norm(Pf-exact)/sqrt(M)
```

Combining Riley's Algorithm with LOOCV

We showed earlier that Riley's algorithm is an efficient solver for ill-conditioned symmetric positive definite linear systems.

Combining Riley's Algorithm with LOOCV

We showed earlier that Riley's algorithm is an efficient solver for ill-conditioned symmetric positive definite linear systems.
That is exactly what we need to do LOOCV.

Combining Riley's Algorithm with LOOCV

We showed earlier that Riley's algorithm is an efficient solver for ill-conditioned symmetric positive definite linear systems.
That is exactly what we need to do LOOCV.
Since we need to compute

$$e_k = \frac{c_k}{A^{-1}_{kk}},$$

we need to adapt Riley to find both $\boldsymbol{c}$ and A^{-1}.

Combining Riley's Algorithm with LOOCV

We showed earlier that Riley's algorithm is an efficient solver for ill-conditioned symmetric positive definite linear systems.
That is exactly what we need to do LOOCV.
Since we need to compute

$$e_k = \frac{c_k}{A^{-1}_{kk}},$$

we need to adapt Riley to find both $\boldsymbol{c}$ and A^{-1}.
Simple (and cheap):
Vectorize Riley's algorithm so that it can handle multiple right-hand sides, i.e., solve

$$A\boldsymbol{c} = [\boldsymbol{f} \;\; I]\,.$$

Combining Riley's Algorithm with LOOCV

We showed earlier that Riley's algorithm is an efficient solver for ill-conditioned symmetric positive definite linear systems.
That is exactly what we need to do LOOCV.
Since we need to compute

$$e_k = \frac{c_k}{A^{-1}_{kk}},$$

we need to adapt Riley to find both $\boldsymbol{c}$ and A^{-1}.
Simple (and cheap):
Vectorize Riley's algorithm so that it can handle multiple right-hand sides, i.e., solve

$$A\boldsymbol{c} = [\boldsymbol{f}\ \ I]\,.$$

Still need $\mathcal{O}(N^3)$ operations (Cholesky factorization unchanged; now matrix forward and back subs).

Combining Riley's Algorithm with LOOCV

We showed earlier that Riley's algorithm is an efficient solver for ill-conditioned symmetric positive definite linear systems.
That is exactly what we need to do LOOCV.
Since we need to compute

$$e_k = \frac{c_k}{A^{-1}_{kk}},$$

we need to adapt Riley to find both $\boldsymbol{c}$ and A^{-1}.
Simple (and cheap):
Vectorize Riley's algorithm so that it can handle multiple right-hand sides, i.e., solve

$$A\boldsymbol{c} = [\boldsymbol{f} \;\; I].$$

Still need $\mathcal{O}(N^3)$ operations (Cholesky factorization unchanged; now matrix forward and back subs).
In fact, the beauty of MATLAB is that the code for `Riley.m` does not change at all.

MATLAB Algorithm for Cost Function using Riley

```
function ceps = CostEps(ep,r,rbf,rhs)
A = rbf(ep,r);
invA = pinv(A);
errorvector = (invA*rhs)./diag(invA);
ceps = norm(errorvector);
```

Program (CostEpsRiley.m)

```
function ceps = CostEpsRiley(ep,r,rbf,rhs)
A = rbf(ep,r);
mu = 1e-11;
% find solution of Ax=b and A^-1
D = Riley(A,[rhs eye(size(A))],mu);
errorvector = D(:,1)./diag(D(:,2:end));
ceps = norm(errorvector);
```

Program (RBFInterpolation_sDLOOCVRiley.m)

```
s = 2;  N = 289;  gridtype = 'h';  M = 500;
global rbf;  rbf_definition;
mine = 0; maxe = 20; mu = 1e-11;
[dsites, N] = CreatePoints(N,s,gridtype);
ctrs = dsites;
epoints = CreatePoints(M,s,'r');
rhs = testfunctionsD(dsites);
DM = DistanceMatrix(dsites,ctrs);
ep = fminbnd(@(ep) CostEpsRiley(ep,DM,rbf,rhs,mu),...
     mine,maxe);
IM = rbf(ep,DM);
coef = Riley(IM,rhs,mu);
DM = DistanceMatrix(epoints,ctrs);
EM = rbf(ep,DM);
Pf = EM * coef;
exact = testfunctionsD(epoints);
maxerr = norm(Pf-exact,inf)
rms_err = norm(Pf-exact)/sqrt(M)
```

Data	900 uniform	900 Halton	2500 uniform	2500 Halton
$\varepsilon_{opt,pinv}$	5.9725	7.0165	6.6974	6.1277
RMS-err	1.6918e-06	4.6326e-07	1.9132e-08	2.0648e-07
cond(A)	8.813772e+19	1.433673e+19	2.835625e+21	7.387674e+20
$\varepsilon_{opt,Riley}$	5.6536	5.9678	6.0635	5.6205
RMS-err	4.1367e-07	7.7984e-07	1.9433e-08	4.9515e-08
cond(A)	1.737214e+21	1.465283e+20	1.079238e+22	5.811212e+20

Table: Interpolation with Gaussians to 2D Franke function.

Data	900 uniform	900 Halton	2500 uniform	2500 Halton
$\varepsilon_{opt,pinv}$	5.9725	7.0165	6.6974	6.1277
RMS-err	1.6918e-06	4.6326e-07	1.9132e-08	2.0648e-07
cond(A)	8.813772e+19	1.433673e+19	2.835625e+21	7.387674e+20
$\varepsilon_{opt,Riley}$	5.6536	5.9678	6.0635	5.6205
RMS-err	4.1367e-07	7.7984e-07	1.9433e-08	4.9515e-08
cond(A)	1.737214e+21	1.465283e+20	1.079238e+22	5.811212e+20

Table: Interpolation with Gaussians to 2D Franke function.

Remark

- *LOOCV with Riley is much faster than with* `pinv` *and of similar accuracy.*
- *If we use backslash in* `CostEps`*, then results are less accurate than with* `pinv`

  e.g., $N = 900$ *uniform: RMS-err*$= 5.1521e\text{-}06$ *with* $\varepsilon = 7.5587$.

LOOCV for Iterated AMLS

Recall

$$\mathcal{Q}_f^{(n)}(\boldsymbol{x}) = \boldsymbol{\Phi}_\varepsilon^T(\boldsymbol{x}) \sum_{\ell=0}^{n} (I - A_\varepsilon)^\ell \boldsymbol{f}$$

Now we find both

- a good value of the shape parameter ε,
- and a good stopping criterion that results in an optimal number, n, of iterations.

LOOCV for Iterated AMLS

Recall

$$\mathcal{Q}_f^{(n)}(\boldsymbol{x}) = \boldsymbol{\Phi}_\varepsilon^T(\boldsymbol{x}) \sum_{\ell=0}^{n} (I - A_\varepsilon)^\ell \boldsymbol{f}$$

Now we find both

- a good value of the shape parameter ε,
- and a good stopping criterion that results in an optimal number, n, of iterations.

Remark

For the latter to make sense we note that for noisy data the iteration acts like a noise filter. However, after a certain number of iterations the noise will begin to feed on itself and the quality of the approximant will degrade.

LOOCV for Iterated AMLS

Recall

$$\mathcal{Q}_f^{(n)}(\boldsymbol{x}) = \boldsymbol{\Phi}_\varepsilon^T(\boldsymbol{x}) \sum_{\ell=0}^{n} (I - A_\varepsilon)^\ell \boldsymbol{f}$$

Now we find both

- a good value of the shape parameter ε,
- and a good stopping criterion that results in an optimal number, n, of iterations.

Remark

For the latter to make sense we note that for noisy data the iteration acts like a noise filter. However, after a certain number of iterations the noise will begin to feed on itself and the quality of the approximant will degrade.

In [F. & Zhang (2007b)] two algorithms were presented.
We discuss one of them.

Rippa's algorithm was designed for LOOCV of interpolation problems. Therefore, convert IAMLS approximation to similar formulation.

Rippa's algorithm was designed for LOOCV of interpolation problems. Therefore, convert IAMLS approximation to similar formulation.
We showed earlier that

$$A\sum_{\ell=0}^{n}(I-A)^{\ell}\boldsymbol{f}=\mathcal{Q}_{\boldsymbol{f}}^{(n)},$$

where $\mathcal{Q}_{\boldsymbol{f}}^{(n)}$ is the IAMLS approximant evaluated on the data sites.
This is a linear system with system matrix A, but right-hand side vector $\mathcal{Q}_{\boldsymbol{f}}^{(n)}$. We want $\boldsymbol{f}$ on the right-hand side.

Rippa's algorithm was designed for LOOCV of interpolation problems.
Therefore, convert IAMLS approximation to similar formulation.
We showed earlier that

$$A \sum_{\ell=0}^{n} (I - A)^{\ell} \boldsymbol{f} = \mathcal{Q}_{\boldsymbol{f}}^{(n)},$$

where $\mathcal{Q}_{\boldsymbol{f}}^{(n)}$ is the IAMLS approximant evaluated on the data sites.
This is a linear system with system matrix A, but right-hand side vector $\mathcal{Q}_{\boldsymbol{f}}^{(n)}$. We want $\boldsymbol{f}$ on the right-hand side.
Therefore, multiply both sides by

$$\left[\sum_{\ell=0}^{n} (I - A)^{\ell}\right]^{-1} A^{-1}$$

and obtain

$$\left[\sum_{\ell=0}^{n} (I - A)^{\ell}\right]^{-1} \left(\sum_{\ell=0}^{n} (I - A)^{\ell} \boldsymbol{f}\right) = \boldsymbol{f}.$$

Now

$$\left[\sum_{\ell=0}^{n}(I-A)^{\ell}\right]^{-1}\left(\sum_{\ell=0}^{n}(I-A)^{\ell}\boldsymbol{f}\right)=\boldsymbol{f}$$

is in the form of a standard interpolation system with

- system matrix $\left[\sum_{\ell=0}^{n}(I-A)^{\ell}\right]^{-1}$,
- coefficient vector $\sum_{\ell=0}^{n}(I-A)^{\ell}\boldsymbol{f}$,
- and the usual right-hand side $\boldsymbol{f}$.

Remark

The matrix $\sum_{\ell=0}^{n}(I-A)^{\ell}$ *is a truncated Neumann series approximation of* A^{-1}.

Do LOOCV for the system

$$\left[\sum_{\ell=0}^{n}(I-A)^{\ell}\right]^{-1}\left(\sum_{\ell=0}^{n}(I-A)^{\ell}\boldsymbol{f}\right)=\boldsymbol{f}$$

Do LOOCV for the system

$$\left[\sum_{\ell=0}^{n}(I-A)^{\ell}\right]^{-1}\left(\sum_{\ell=0}^{n}(I-A)^{\ell}\boldsymbol{f}\right)=\boldsymbol{f}$$

Now formula for components of the error vector becomes

$$e_k=\frac{c_k}{(\text{sytem matrix})^{-1}_{kk}}=\frac{\left[\sum_{\ell=0}^{n}(I-A)^{\ell}\boldsymbol{f}\right]_k}{\left[\sum_{\ell=0}^{n}(I-A)^{\ell}\right]_{kk}}$$

Do LOOCV for the system

$$\left[\sum_{\ell=0}^{n}(I-A)^{\ell}\right]^{-1}\left(\sum_{\ell=0}^{n}(I-A)^{\ell}\boldsymbol{f}\right)=\boldsymbol{f}$$

Now formula for components of the error vector becomes

$$e_k=\frac{c_k}{(\text{sytem matrix})_{kk}^{-1}}=\frac{\left[\sum_{\ell=0}^{n}(I-A)^{\ell}\boldsymbol{f}\right]_k}{\left[\sum_{\ell=0}^{n}(I-A)^{\ell}\right]_{kk}}$$

No matrix inverse required!

Do LOOCV for the system

$$\left[\sum_{\ell=0}^{n}(I-A)^{\ell}\right]^{-1}\left(\sum_{\ell=0}^{n}(I-A)^{\ell}\boldsymbol{f}\right)=\boldsymbol{f}$$

Now formula for components of the error vector becomes

$$e_k=\frac{c_k}{(\text{sytem matrix})^{-1}_{kk}}=\frac{\left[\sum_{\ell=0}^{n}(I-A)^{\ell}\boldsymbol{f}\right]_k}{\left[\sum_{\ell=0}^{n}(I-A)^{\ell}\right]_{kk}}$$

No matrix inverse required!

Numerator and denominator can be accumulated iteratively.

Numerator: take k^{th} component of

$$\boldsymbol{v}^{(0)}=\boldsymbol{f},\qquad \boldsymbol{v}^{(n)}=\boldsymbol{f}+(I-A)\,\boldsymbol{v}^{(n-1)}$$

Denominator: take k^{th} diagonal element of

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

Complexity of matrix powers in denominator can be reduced by using an eigen-decomposition.

Complexity of matrix powers in denominator can be reduced by using an eigen-decomposition.
First compute

$$I - A = X \Lambda X^{-1},$$

where

- Λ: diagonal matrix of eigenvalues of $I - A$,
- X: columns are eigenvectors.

Complexity of matrix powers in denominator can be reduced by using an eigen-decomposition.
First compute

$$I - A = X \Lambda X^{-1},$$

where

- Λ: diagonal matrix of eigenvalues of $I - A$,
- X: columns are eigenvectors.

Then, iterate

$$M^{(0)} = I, \qquad M^{(n)} = I + \Lambda M^{(n-1)}$$

so that, for any fixed n,

$$\left[\sum_{\ell=0}^{n} (I - A)^{\ell}\right] = X M^{(n)} X^{-1}.$$

Complexity of matrix powers in denominator can be reduced by using an eigen-decomposition.
First compute

$$I - A = X \Lambda X^{-1},$$

where

- Λ: diagonal matrix of eigenvalues of $I - A$,
- X: columns are eigenvectors.

Then, iterate

$$M^{(0)} = I, \qquad M^{(n)} = I + \Lambda M^{(n-1)}$$

so that, for any fixed n,

$$\left[\sum_{\ell=0}^{n} (I - A)^{\ell} \right] = X M^{(n)} X^{-1}.$$

Need only diagonal elements of this. Since $M^{(n)}$ is diagonal this can be done efficiently as well.

Algorithm (for iterated AMLS with LOOCV)

```
Fix $\varepsilon$. Perform an eigen-decomposition
    $$I - A = X\Lambda X^{-1}$$
Initialize $\boldsymbol{v}^{(0)} = \boldsymbol{f}$ and $M^{(0)} = I$
For $n = 1, 2, \dots$
    Perform the updates
        $$\begin{aligned} \boldsymbol{v}^{(n)} &= \boldsymbol{f} + (I - A)\,\boldsymbol{v}^{(n-1)} \\ M^{(n)} &= I + \Lambda M^{(n-1)} \end{aligned}$$
    Compute the cost vector (using MATLAB notation)
        $$\boldsymbol{e}^{(n)} = \boldsymbol{v}^{(n)}./\mathrm{diag}(X * M^{(n)}/X)$$
    If $\left\|\boldsymbol{e}^{(n)}\right\| - \left\|\boldsymbol{e}^{(n-1)}\right\| < tol$
        Stop the iteration
    end
end
```

Also finds optimal stopping value for n

Ridge regression or smoothing splines

(see, e.g., [Kimeldorf & Wahba (1971)])

$$\min_{\boldsymbol{c}} \left\{ \boldsymbol{c}^T A \boldsymbol{c} + \gamma \sum_{j=1}^{N} \left(\mathcal{P}_f(\boldsymbol{x}_j) - f_j \right)^2 \right\},$$

Ridge regression or smoothing splines

(see, e.g., [Kimeldorf & Wahba (1971)])

$$\min_{\boldsymbol{c}} \left\{ \boldsymbol{c}^T A \boldsymbol{c} + \gamma \sum_{j=1}^{N} \left(\mathcal{P}_f(\boldsymbol{x}_j) - f_j \right)^2 \right\},$$

Equivalent to solving

$$\left(A + \frac{1}{\gamma} I \right) \boldsymbol{c} = \boldsymbol{f}.$$

Ridge regression or smoothing splines

(see, e.g., [Kimeldorf & Wahba (1971)])

$$\min_{\boldsymbol{c}} \left\{ \boldsymbol{c}^T A \boldsymbol{c} + \gamma \sum_{j=1}^{N} \left(\mathcal{P}_f(\boldsymbol{x}_j) - f_j \right)^2 \right\},$$

Equivalent to solving

$$\left(A + \frac{1}{\gamma} I \right) \boldsymbol{c} = \boldsymbol{f}.$$

Just like before, so LOOCV error components given by

$$e_k = \frac{\left[\left(A + \frac{1}{\gamma} I \right)^{-1} \boldsymbol{f} \right]_k}{\left(A + \frac{1}{\gamma} I \right)^{-1}_{kk}}.$$

The "optimal" values of the shape parameter ε and the smoothing parameter γ are determined in a nested manner.
We now use a new cost function `CostEpsGamma`

Program (`CostEpsGamma.m`)

```
function ceg = CostEpsGamma(ep,gamma,r,rbf,rhs,ep)
A = rbf(ep,r);
A = A + eye(size(A))/gamma;
invA = pinv(A);
errorvector = (invA*rhs)./diag(invA);
ceg = norm(errorvector);
```

For a fixed initial ε we find the "optimal" γ followed by an optimization of `CostEpsGamma` over ε.
The algorithm terminates when the difference between to successive optimization runs is sufficiently small.

$N =$		9	25	81	289	1089
AMLS	RMSerr	4.80e-3	1.53e-3	6.42e-4	4.39e-4	2.48e-4
	ε	1.479865	1.268158	0.911530	0.652600	0.468866
	no. iter.	7	6	6	4	3
	time	0.2	0.4	1.0	5.7	254
Ridge	RMSerr	3.54e-3	1.62e-3	7.20e-4	4.57e-4	2.50e-4
	ε	2.083918	0.930143	0.704802	0.382683	0.181895
	γ	100.0	100.0	47.324909	26.614484	29.753487
	time	0.3	1.2	1.1	21.3	672

Table: Comparison of IAMLS and ridge regression using Gaussians for noisy data sampled at Halton points.

See [F. & Zhang (2007a)]

RBF-PS methods

Adapt Rippa's LOOCV algorithm for RBF-PS methods

RBF-PS methods

Adapt Rippa's LOOCV algorithm for RBF-PS methods
Instead of $A\boldsymbol{c} = \boldsymbol{f}$ with components of the cost vector determined by

$$e_k = \frac{c_k}{A^{-1}_{kk}}$$

we now have (due to the symmetry of A)

$$D = A_{\mathcal{L}} A^{-1} \quad \Longleftrightarrow \quad A D^T = (A_{\mathcal{L}})^T$$

RBF-PS methods

Adapt Rippa's LOOCV algorithm for RBF-PS methods
Instead of $A\boldsymbol{c} = \boldsymbol{f}$ with components of the cost vector determined by

$$e_k = \frac{c_k}{A^{-1}_{kk}}$$

we now have (due to the symmetry of A)

$$D = A_{\mathcal{L}} A^{-1} \quad \Longleftrightarrow \quad AD^T = (A_{\mathcal{L}})^T$$

so that the components of the cost matrix are given by

$$E_{k\ell} = \frac{(D^T)_{k\ell}}{A^{-1}_{kk}}.$$

In MATLAB this can again be vectorized:

Program (CostEpsLRBF.m)

```
function ceps = CostEpsLRBF(ep,DM,rbf,Lrbf)
N = size(DM,2);
A = rbf(ep,DM);
rhs = Lrbf(ep,DM)';
invA = pinv(A);
errormatrix = (invA*rhs)./repmat(diag(invA),1,N);
ceps = norm(errormatrix(:));
```

In MATLAB this can again be vectorized:

Program (CostEpsLRBF.m)

```
function ceps = CostEpsLRBF(ep,DM,rbf,Lrbf)
N = size(DM,2);
A = rbf(ep,DM);
rhs = Lrbf(ep,DM)';
invA = pinv(A);
errormatrix = (invA*rhs)./repmat(diag(invA),1,N);
ceps = norm(errormatrix(:));
```

The function `Lrbf` creates the matrix $A_{\mathcal{L}}$. For the Gaussian RBF and the Laplacian differential operator this could look like

```
Lrbf = @(ep,r) 4*ep^2*exp(-(ep*r).^2).*((ep*r).^2-1);
```

In MATLAB this can again be vectorized:

Program (CostEpsLRBF.m)

```
function ceps = CostEpsLRBF(ep,DM,rbf,Lrbf)
N = size(DM,2);
A = rbf(ep,DM);
rhs = Lrbf(ep,DM)';
invA = pinv(A);
errormatrix = (invA*rhs)./repmat(diag(invA),1,N);
ceps = norm(errormatrix(:));
```

The function `Lrbf` creates the matrix $A_{\mathcal{L}}$. For the Gaussian RBF and the Laplacian differential operator this could look like

```
Lrbf = @(ep,r) 4*ep^2*exp(-(ep*r).^2).*((ep*r).^2-1);
```

Remark

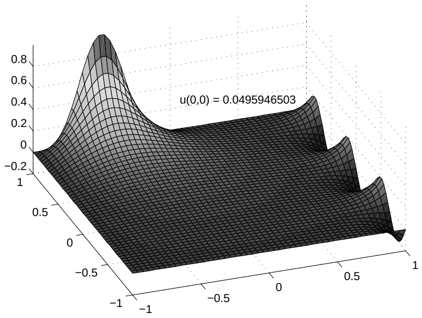
For differential operators of odd order one also needs difference matrices.

Example (2D Laplace equation, Program 36 of [Trefethen (2000)])

$$u_{xx} + u_{yy} = 0, \quad x, y \in (-1, 1)^2,$$

with piecewise defined boundary conditions

$$u(x,y) = \begin{cases} \sin^4(\pi x), & y = 1 \text{ and } -1 < x < 0, \\ \frac{1}{5}\sin(3\pi y), & x = 1, \\ 0, & \text{otherwise.} \end{cases}$$



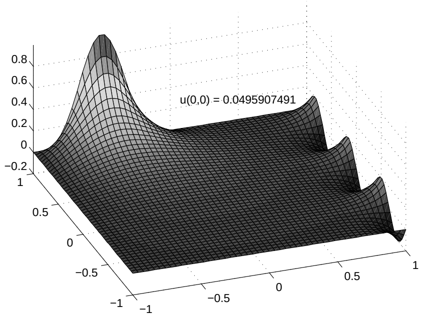


Figure: Solution of the Laplace equation using a Chebyshev PS approach (left) and cubic Matérn RBFs with $\varepsilon = 0.362752$ (right) with 625 collocation points.

Example (2D Helmholtz equation, Program 17 in [Trefethen (2000)])

$$u_{xx} + u_{yy} + k^2 u = f(x,y), \quad x, y \in (-1,1)^2,$$

with boundary condition $u = 0$ and

$$f(x,y) = \exp\left(-10\left[(y-1)^2 + (x-\frac{1}{2})^2\right]\right).$$

Example (2D Helmholtz equation, Program 17 in [Trefethen (2000)])

$$u_{xx} + u_{yy} + k^2 u = f(x,y), \quad x, y \in (-1,1)^2,$$

with boundary condition $u = 0$ and

$$f(x,y) = \exp\left(-10\left[(y-1)^2 + (x-\frac{1}{2})^2\right]\right).$$

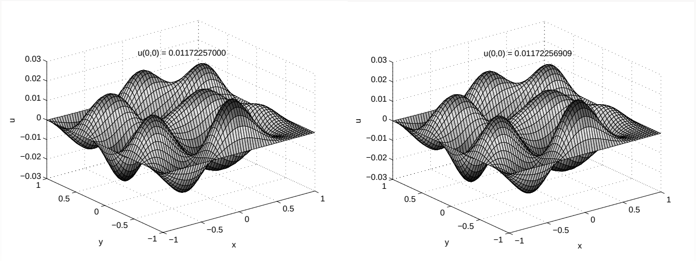


Figure: Solution of 2-D Helmholtz equation with 625 collocation points using the Chebyshev pseudospectral method (left) and Gaussians with $\varepsilon = 2.549243$ (right).

Example (Allen-Cahn equation, Program 35 in [Trefethen (2000)])

Most challenging for the RBF-PS method.

$$u_t = \mu u_{xx} + u - u^3, \qquad x \in (-1,1),\ t \geq 0,$$

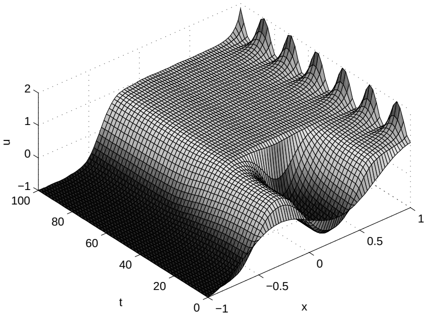
with parameter $\mu = 0.01$, initial condition

$$u(x,0) = 0.53x + 0.47\sin\left(-\frac{3}{2}\pi x\right), \qquad x \in [-1,1],$$

and non-homogeneous (time-dependent) boundary conditions

$$u(-1,t) = -1 \text{ and } u(1,t) = \sin^2(t/5).$$

The solution to this equation has three steady states ($u = -1, 0, 1$) with the two nonzero solutions being stable. The transition between these states is governed by the parameter μ.
The unstable state should vanish around $t = 30$.



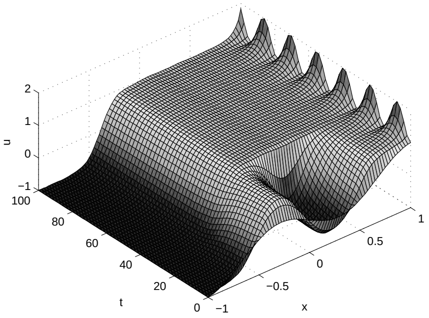


Figure: Solution of the Allen-Cahn equation using the Chebyshev pseudospectral method (left) and a cubic Matérn functions with $\varepsilon = 0.350920$ (right) with 21 Chebyshev points.

Summary

- Several applications of LOOCV:
  - RBF interpolation (with and without Riley),
  - IAMLS,
  - ridge regression,
  - RBF-PS
- Riley more efficient than `pinv`
- IAMLS method performs favorably when compared to ridge regression for noisy data (no dense linear systems solved)
- LOOCV algorithm for finding an “optimal” shape parameter for Kansa’s method in [Ferreira *et al.* (2007)]

Future work or work in progress:

- variable shape parameters (e.g., [Kansa & Carlson (1992), Fornberg and Zuev (2007)])
    - potential for improved accuracy and stability
    - challenging at the theoretical level
    - difficult multivariate optimization problem
- other criteria for “optimal” ε
    - compare Fourier transforms of kernels with data
    - maximum likelihood

References I

Buhmann, M. D. (2003).
Radial Basis Functions: Theory and Implementations.
Cambridge University Press.

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.

Trefethen, L. N. (2000).
Spectral Methods in MATLAB.
SIAM (Philadelphia, PA).

Wahba, G. (1990).
Spline Models for Observational Data.
CBMS-NSF Regional Conference Series in Applied Mathematics 59, SIAM (Philadelphia).

References II

Wendland, H. (2005).
Scattered Data Approximation.
Cambridge University Press.

O. Dubrule.
Cross validation of Kriging in a unique neighborhood.
J. Internat. Assoc. Math. Geol. **15**/6 (1983), 687–699.

Fasshauer, G. E. and Zhang, J. G. (2007).
Scattered data approximation of noisy data via iterated moving least squares.
in *Curve and Surface Fitting: Avignon 2006*, T. Lyche, J. L. Merrien and L. L. Schumaker (eds.), Nashboro Press, pp. 150–159.

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

References III

Ferreira, A. J. M., Fasshauer, G. E., Roque, C. M. C., Jorge, R. M. N. and Batra, R. C. (2007).
Analysis of functionally graded plates by a robust meshless method.
J. Mech. Adv. Mater. & Struct. **14**/8, pp. 577–587.

Fornberg, B. and Zuev, J. (2007).
The Runge phenomenon and spatially variable shape parameters in RBF interpolation,
Comput. Math. Appl. **54**, pp. 379–398.

Kansa, E. J. (1990).
Multiquadrics — A scattered data approximation scheme with applications to computational fluid-dynamics — II: Solutions to parabolic, hyperbolic and elliptic partial differential equations.
Comput. Math. Applic. **19**, pp. 147–161.

Kansa, E. J. and Carlson, R. E. (1992).
Improved accuracy of multiquadric interpolation using variable shape parameters.
Comput. Math. Applic. **24**, pp. 99–120.

References IV

Kimeldorf, G. and Wahba, G. (1971).
Some results on Tchebycheffian spline functions.
J. Math. Anal. Applic. **33**, pp. 82–95.

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.