PADOVA UNIVERSITY PRESS

Dolomites Research Notes on Approximation

Special Issue SA2022 – *Software for Approximation 2022*, Volume 15 · 2022 · Pages 101–108

A note on the usage of the `pinv` MATLAB function*

Francesco Marchetti[a]

Abstract

In MATLAB, the function `pinv` is the default choice when computing the Moore-Penrose inverse of a matrix. In this note, we discuss possible alternative strategies for its calculation and we show that such built-in function might not represent an optimal choice in some cases. The presentation is supported by a detailed *step-by-step* MATLAB implementation.

1 Introduction

The Moore-Penrose inverse [6] (also referred to as pseudoinverse) is a well-known generalisation of the classical matrix inverse. It is employed in the computation of the least squares solution of overdetermined linear systems, as well as in calculating the minimum norm solution in undetermined ones [7]. Therefore, it is a fundamental tool in many fields and applications, and its properties have been investigated in depth [8, 9]. As a consequence, dedicated built-in functions are common in various programming languages, including e.g. MATLAB (or Octave), where the function `pinv` is at the disposal.

In this note, our purpose is to show that such a default function might not be an *optimal* choice for computing the pseudoinverse, under certain circumstances where the underlying problem is not that gravely ill-conditioned. In this case, computing the pseudoinverse by directly solving the so-called *normal equations* may provide a saving in computational time, along with a satisfactory accuracy in the process. In order to experiment on this topic, here we restrict to the kernel-based approximation setting, since its native flexibility allows a straightforward construction of approximation problems characterised by different levels of conditioning. Throughout the paper, the discussion is supported by related MATLAB codes.

The outline of this work is the following. In Section 2, we introduce the kernel-based approximation setting. In Section 3 various strategies for the computation of the pseudoinverse in MATLAB are discussed, and then tested in Section 4. Some final comments are then provided in Section 5.

2 Kernel-based approximation in MATLAB

In the following, we mainly refer to [2, §19.3]. For a more general formulation of the approximant, and for a complete overview concerning kernel-based approximation, we refer the reader e.g. to [3, 10].

Let $\Omega \subseteq \mathbb{R}^d$ and let $\mathcal{X} = \{\boldsymbol{x}_i,\ i = 1,\dots,n\} \subset \Omega$ be a set of distinct nodes, $n \in \mathbb{N}$, with $\boldsymbol{x}_i = (x_{i,1},\dots,x_{i,d})^\intercal$, $i = 1,\dots,n$. Given $f : \Omega \longrightarrow \mathbb{R}$, suppose that the set of function values $\mathcal{F} = \{f(\boldsymbol{x}_i),\ i = 1,\dots,n\}$ is at the disposal. Furthermore, let $\Xi = \{\boldsymbol{\xi}_i,\ i = 1,\dots,m\} \subset \Omega$ be a set of distinct centres, $m \in \mathbb{N}$, $m \leq n$. In order to recover the underlying function f on Ω, we consider the kernel-based approximant

$$S_f(\boldsymbol{x}) = \sum_{i=1}^{m} c_i \kappa_\varepsilon(\boldsymbol{x}, \boldsymbol{\xi}_i), \quad \boldsymbol{x} \in \Omega,$$

where and $\kappa_\varepsilon : \Omega \times \Omega \longrightarrow \mathbb{R}$ is a strictly positive definite kernel depending on a *shape parameter* $\varepsilon > 0$. We suppose κ_ε to be *radial*, i.e., there exists a univariate Radial Basis Function (RBF) $\varphi_\varepsilon : \mathbb{R}_{\geq 0} \longrightarrow \mathbb{R}$ such that $\kappa_\varepsilon(\boldsymbol{x}, \boldsymbol{y}) = \widetilde{\varphi}(\varepsilon r) = \varphi_\varepsilon(r)$, with $r = \|\boldsymbol{x} - \boldsymbol{y}\|_2$.

Letting $\Lambda = (\Lambda_{i,j}) = \kappa_\varepsilon(\boldsymbol{x}_i, \boldsymbol{\xi}_j)$, $i = 1,\dots,n$, $j = 1,\dots,m$ be the *collocation* matrix and given the vector of function values $\boldsymbol{f} = (f_1,\dots,f_n)^\intercal$, a vector of coefficients $\boldsymbol{c} = (c_1,\dots,c_m)^\intercal \in \mathbb{R}^m$ is determined by solving in the least squares sense the linear system

$$\Lambda \boldsymbol{c} = \boldsymbol{f}, \tag{1}$$

i.e., by minimizing

$$\sum_{i=1}^{n} (S_f(\boldsymbol{x}_i) - f(\boldsymbol{x}_i))^2.$$

The vector $\boldsymbol{c}$ is uniquely determined as long as Λ has full rank. We observe that the classical interpolation framework is recovered by setting $\Xi = \mathcal{X}$. In this paper, we take $\Xi \subseteq \mathcal{X}$, which guarantees $\mathrm{rk}(\Lambda) = m$ and thus the uniqueness of the approximant S_f.

In the following, we display a MATLAB function that constructs the collocation matrix Λ in the presented setting. The `DistanceMatrix.m` function is available at [2, Program 1.1].

*The preface of this special issue to which the article belongs is given in [1].

[a]Department of Mathematics "Tullio Levi-Civita", University of Padova, Italy

MATLAB code

```
function CM = collocation_builder(dsites,ctrs,rbf,ep)

%---------------------------------------------------------------------------%
%
% Goal:     Build the collocation matrix
%
% Inputs:   dsites:   n x d array, data points
%           ctrs:     m x d array, centres
%           rbf:      function handle, considered RBF
%           ep:       positive scalar, shape parameter
%
% Outputs:  CM:       n x m array, the collocation matrix
%
%
% Calls on: DistanceMatrix (by G. Fasshauer)
%
%---------------------------------------------------------------------------%

DM = DistanceMatrix(dsites,ctrs);
CM = rbf(ep,DM);

end
```

It is well known that $\boldsymbol{c} = \Lambda^+ \boldsymbol{f}$, where

$$\Lambda^+ = (\Lambda^\intercal \Lambda)^{-1} \Lambda^\intercal \tag{2}$$

is the Moore-Penrose inverse of the collocation matrix. In the next section, we present different manners in which Λ^+ can be computed in MATLAB.

3 Computing the pseudoinverse

Here, we refer to the documentation in [5].

To calculate the matrix Λ^+ in MATLAB one usually employs the built-in function `pinv`, which makes use of the singular value decomposition to form the pseudoinverse. In particular, it allows treating as zeros the singular values that are smaller than a certain tolerance `tol`, whose default value is $\max\{m,n\}\text{eps}(\|\Lambda\|_2)$, being eps the function that computes the floating-point relative accuracy of its argument. By virtue of this regularising effect, `pinv` is sometimes preferred to `inv` when facing severe ill-conditioning. Then, (1) is solved by computing the matrix-vector product between the pseudoinverse and $\boldsymbol{f}$.

Besides such **pinv method**, we recall the following alternative strategies, which we test and compare in Section 4. More details are included in the MATLAB code below.

1. The **\(backslash) method**. The matrix Λ^+ can be computed by using the backslash MATLAB operator in combination with an $n \times n$ identity matrix. The MATLAB QR solver is employed. Moreover, such method can be applied to solve (1) without making Λ^+ explicit.
2. The **mult method**. We calculate Λ^+ by using its definition (2). This strategy is equivalent to solving the normal equations and it should be avoided, usually, because of the possible ill-conditioning related to the problem would be amplified. Nevertheless, in Section 4 we will show its competitiveness under certain circumstances. Then, (1) is solved by computing the matrix-vector product between Λ^+ and $\boldsymbol{f}$.
3. The **mult+Tik method**. This approach is similar to the **mult**. Here, the matrix inversion is regularised by using the *Tikhonov regularisation*, where a L_2-norm penalty term is added to the problem [11].

MATLAB code

```
function [PICM,coeffs] = pseudo_compute(CM,rhs,mode_t,mode_p)

%---------------------------------------------------------------------------%
%
% Goal:     Pseudoinversion of the collocation matrix and/or solution
%           of the linear system
%
% Inputs:   CM:     n x m array, the collocation matrix
%           rhs:    n x 1 array, vector of function values
%           mode_t: string, 'i' to compute the pseudoinverse, 's' to
%                   solve the linear system directly
%           mode_p: string, the computational method: 'pinv', 'backslash',
%                   mult' or 'multi+Tik'.
%
% Outputs:  PICM:   m x n array, the pseudoinverse of CM. In
%                   ('s','backslash') mode, PICM is set to the zero scalar
%           coeffs: m x 1 array, the solution of the linear system. In
%                   'i' mode, coeffs is set to the zero scalar
%
```

```
%--------------------------------------------------------------------------%

if strcmp(mode_t,'i')
    if strcmp(mode_p,'pinv')
        PICM = pinv(CM);
    end
    if strcmp(mode_p,'backslash')
        PICM = CM\eye(size(CM,1));
    end
    if strcmp(mode_p,'mult')
        PICM = ((CM'*CM)\eye(size(CM,2)))*CM';
    end
    if strcmp(mode_p,'mult+Tik')
        PICM = ((CM'*CM+10^(-8)*eye(size(CM,2)))\eye(size(CM,2)))*CM';
    end

    coeffs = 0;
end

if strcmp(mode_t,'s')
    if strcmp(mode_p,'pinv')
        PICM = pinv(CM);
        coeffs = PICM*rhs;
    end
    if strcmp(mode_p,'backslash')
        coeffs = CM\rhs;
        PICM = 0;
    end
    if strcmp(mode_p,'mult')
        PICM = ((CM'*CM)\eye(size(CM,2)))*CM';
        coeffs = PICM*rhs;
    end
    if strcmp(mode_p,'mult+Tik')
        PICM = ((CM'*CM+10^(-8)*eye(size(CM,2)))\eye(size(CM,2)))*CM';
        coeffs = PICM*rhs;
    end
end

end
```

Remark 1. Very often, the aim is solving (1) with no need of making Λ^+ explicit. Nevertheless, expressing Λ^+ or some of its submatrices may be of interest, such as, e.g., in [4].

4 Numerics

Let $\Omega = [-1,1]^2$ and let $f_1, f_2 : \Omega \longrightarrow \mathbb{R}$ be defined as

$$f_1(\boldsymbol{x}) = |0.5 - x_1 - x_2| + 3x_1 - x_2^2,$$
$$f_2(\boldsymbol{x}) = \left(1 + (x_1 - 0.5)^2 + (x_2 + 0.2)^2\right)^{-1},$$

with $\boldsymbol{x} = (x_1, x_2)$. As evaluation set, we take an equispaced $p \times p$ grid Υ in Ω, with $p = 60$. Moreover, an equispaced $q \times q$ grid $\mathcal{X}$ in Ω is employed as set of $n = q^2$ nodes, with $q = 40$. Then, for an underlying function f_i, $i = 1, 2$, and a RBF φ_ε fixed, we perform the following test.

1. We take a random subset $\Xi \subset \mathcal{X}$ of m centres, with $m = \lfloor n/10 \rfloor$.
2. We construct the collocation matrix Λ upon φ_ε, Ξ and $\mathcal{X}$, as detailed in Section 2.
3. We compute both
   - the pseudoinverse Λ^+ and
   - the solution of (1)

   by applying the strategies outlined in Section 3, also reporting the required CPU times.
4. We evaluate the obtained approximant S_f by computing on Υ the Root Mean Squared Error (RMSE) with respect to the underlying function $f = f_1, f_2$, i.e., by denoting the elements of Υ as $\boldsymbol{t}_i$, $i = 1, \ldots, p^2$,

$$RMSE = \sqrt{\frac{1}{p^2} \sum_{i=1}^{p^2} |f(\boldsymbol{t}_i) - S_f(\boldsymbol{t}_i)|^2}.$$

In our experiments, this scheme is repeated 10 times for each approximation setting in order to achieve some statistical robustness: the computational times and errors depicted in the figures below are indeed the mean values related to these repetitions. Moreover,

we display the results obtained varying the value of the shape parameter $\varepsilon \in [2^{-7}, 2^7]$. Precisely, such interval is discretized in log-form. We experiment with the globally supported RBFs (see Subsection 4.1)

$$\varphi_{\varepsilon,M0}(r) = e^{-\varepsilon r}, \quad \text{MatÃ©rn } C^0,$$
$$\varphi_{\varepsilon,M6}(r) = e^{-\varepsilon r}(15 + 15\varepsilon r + 6(\varepsilon r)^2 + (\varepsilon r)^3), \quad \text{MatÃ©rn } C^6,$$

and with the compactly supported RBFs (see Subsection 4.2)

$$\varphi_{\varepsilon,W0}(r) = \max(0, 1-\varepsilon r)^2, \quad \text{Wendland } C^0,$$
$$\varphi_{\varepsilon,W6}(r) = \max(0, 1-\varepsilon r)^8(32(\varepsilon r)^3 + 25(\varepsilon r)^2 + 8(\varepsilon r) + 1), \quad \text{Wendland } C^6.$$

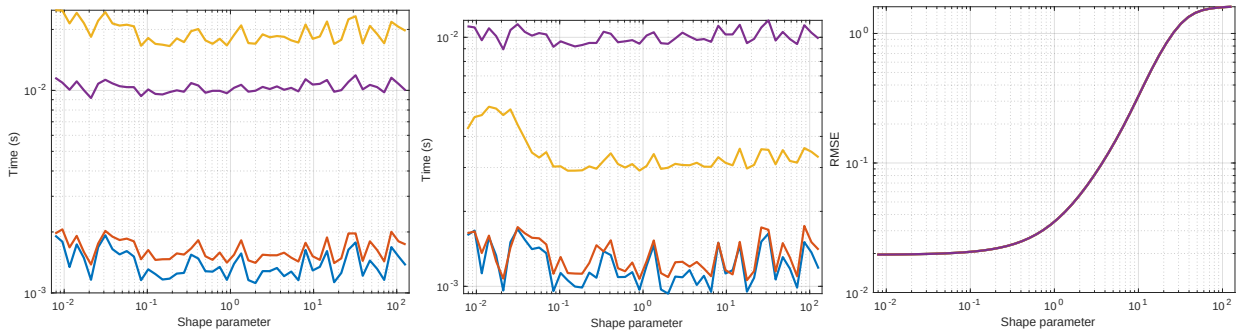
We detail an instantiation of the presented procedure in the following MATLAB code.

MATLAB code

```
%% SETTINGS

% Choose the modes

mode_t = 's';
mode_p = 'mult';

% Choose the function
f = @(x,y) 1./(1+(x-0.5).^2+(y+0.2).^2);
% f = @(x,y) abs(0.5-x-y)+3*x-y.^2;

% Choose the RBF
ep = 8;
% rbf = @(e,r) exp(-e.*r);
rbf =@(e,r) exp(-e*r).*(15+15*e*r+6*(e*r).^2+(e*r).^3);

%% COMPUTATIONS

[xx,yy] = meshgrid(linspace(-1,1,60));
epoints = [xx(:) yy(:)];
[xx,yy] = meshgrid(linspace(-1,1,40));
dsites = [xx(:) yy(:)];
rhs = f(dsites(:,1),dsites(:,2));
f_true = f(epoints(:,1),epoints(:,2));

rng(92)
indices = randperm(size(dsites,1),floor(size(dsites,1)*0.1));
ctrs = dsites(indices,:);
CM = collocation_builder(dsites,ctrs,rbf,ep);
EM = collocation_builder(epoints,ctrs,rbf,ep);

cputime = tic;
[PICM,coeffs] = pseudo_compute(CM,rhs,mode_t,mode_p);
cputime = toc(cputime);

if strcmp(mode_t,'s')
    approx_eval = EM*coeffs;
    rmse = sqrt(mean(((approx_eval-f_true)).^2));
end

%% RESULTS

fprintf('\n')
if strcmp(mode_t,'i')
    fprintf('CPU time for the pseudoinversion: %e\n',cputime)
else
    fprintf('CPU time for solving the linear system: %e\n',cputime)
    fprintf('RMSE test error: %e\n',rmse)
end
```

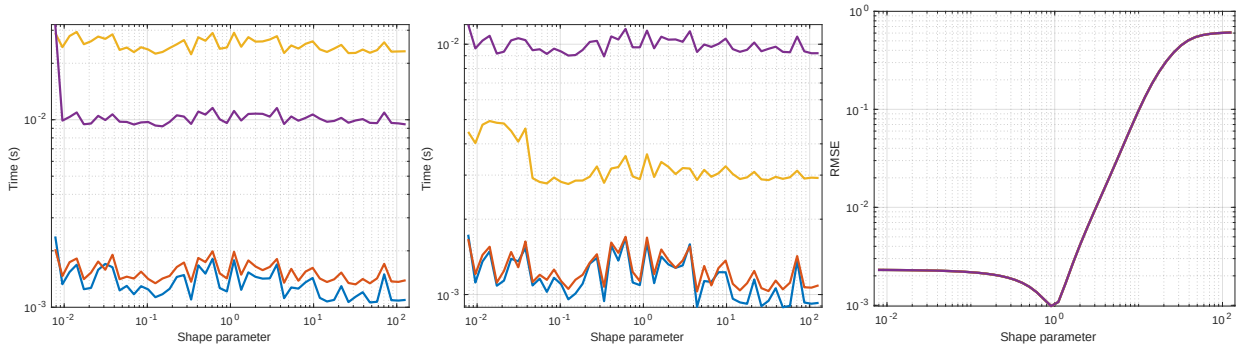
4.1 RBFs with global support

The results obtained by taking $\varphi_{\varepsilon,M0}$ and $\varphi_{\varepsilon,M6}$ are displayed in Figures 1, 2, 3 and 4.



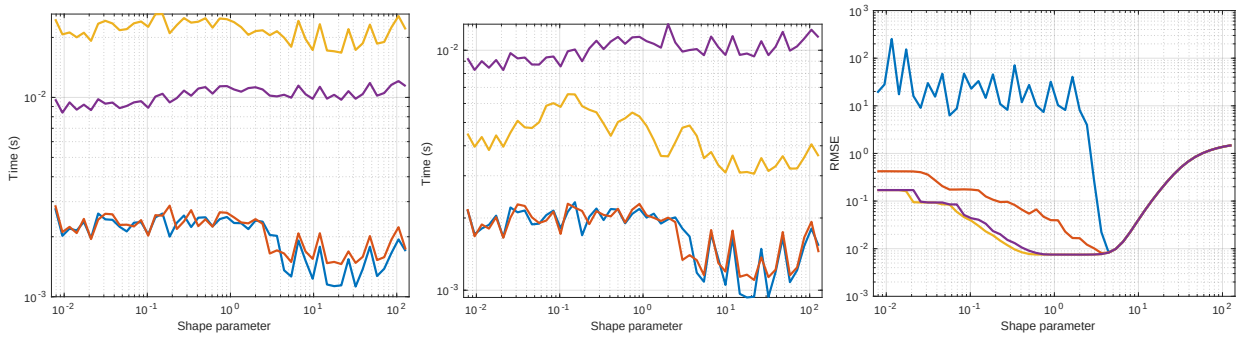
(a) CPU time for computing Λ^+. **(b)** CPU time for solving (1) directly. **(c)** RMSE on Υ.

Figure 1: Approximation of f_1 via $\varphi_{\varepsilon,M0}$. Legend: • backslash, • mult, • mult+Tik, • pinv.



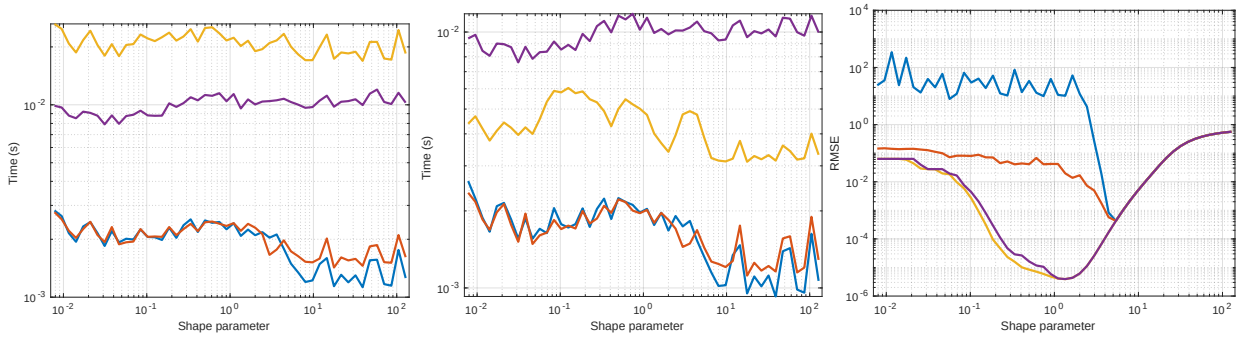
(a) CPU time for computing Λ^+. **(b)** CPU time for solving (1) directly. **(c)** RMSE on Υ.

Figure 2: Approximation of f_2 via $\varphi_{\varepsilon,M0}$. Legend: • backslash, • mult, • mult+Tik, • pinv.



(a) CPU time for computing Λ^+. **(b)** CPU time for solving (1) directly. **(c)** RMSE on Υ.

Figure 3: Approximation of f_1 via $\varphi_{\varepsilon,M6}$. Legend: • backslash, • mult, • mult+Tik, • pinv.



(a) CPU time for computing Λ^+. **(b)** CPU time for solving (1) directly. **(c)** RMSE on Υ.

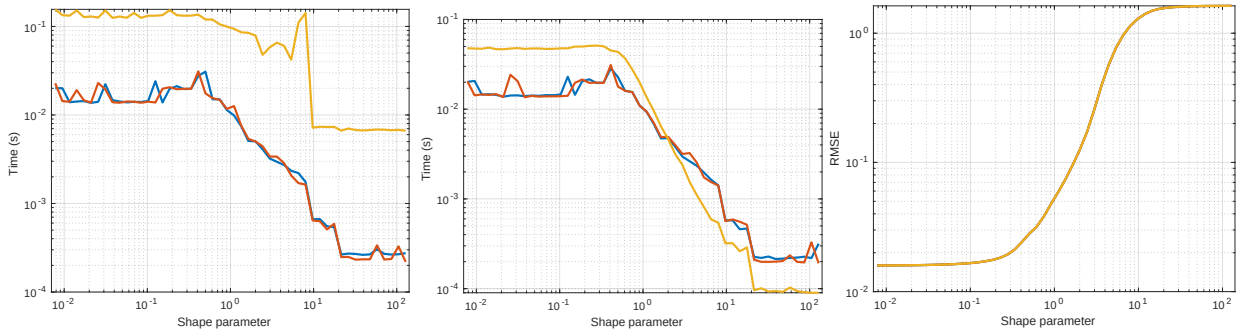
Figure 4: Approximation of f_2 via $\varphi_{\varepsilon,M6}$. Legend: backslash, mult, mult+Tik, pinv.

We observe the following.

- The **pinv method** is faster with respect to the \ **method** in computing Λ^+, but the \ **method** provides a faster direct resolution of (1) and slightly more accurate approximants.
- When using the MatÃ©rn C^0 RBF in Figures 1 and 2, which provides interpolation processes that are not ill-conditioned, the **mult method** definitely outperforms both the **pinv** and the \ **method** in terms of required CPU time, also providing approximants of the same accuracy.
- When using the MatÃ©rn C^6 RBF in Figures 3 and 4, which provides severely ill-conditioned approximation processes, on the one hand the **mult method** is the fastest strategy, but on the other hand the accuracy of its related approximants is deteriorated as the shape parameter gets *small enough*. This is related to the well-known accuracy/stability *trade-off principles*; we refer the interested reader to [2, §16]. When dealing with f_1 (Figure 3), its best performance varying ε is still competitive with the other strategies. When approximating f_2, the **pinv** and the \ **method** are more precise than the **mult method** (Figure 4). Nevertheless, especially when employing the Tikhonov regularisation, we obtained precise approximants with lower computational costs with respect to the other methods.
- The value of the shape parameter does not influence the required CPU time. Indeed, the computed collocation matrices are dense as we are using globally supported RBFs.

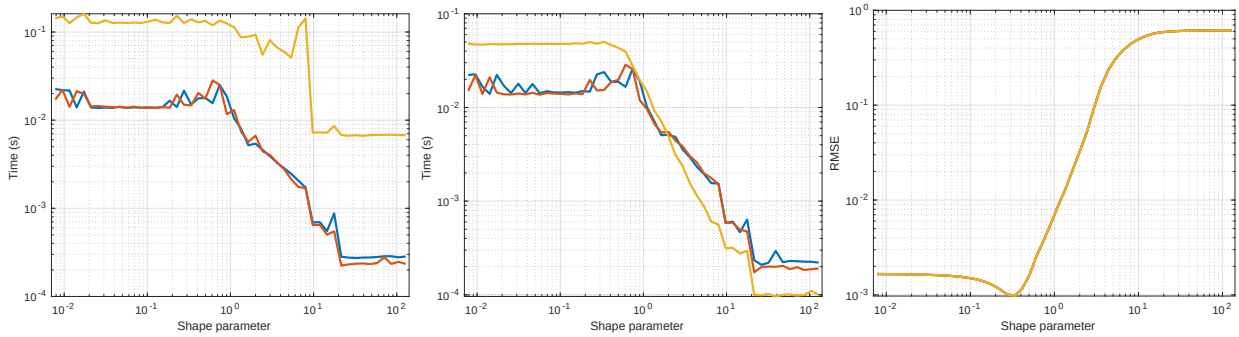
4.2 RBFs with compact support

Here, in performing the numerical tests, we carry out some slight modifications of the code in order to employ the `sparse` MATLAB framework [5]. Although the usage of MATLAB functions designed for dealing with sparsity is partly justified in our experiments, since the involved collocation matrices attain some sparsity for certain values of ε only, we point out that our purpose is not to compare the sparse to the dense setting, but instead to assess whether the observations provided in Subsection 4.1 are valid in this different MATLAB framework too. In Figures 5, 6, 7 and 8 we depict the obtained results.



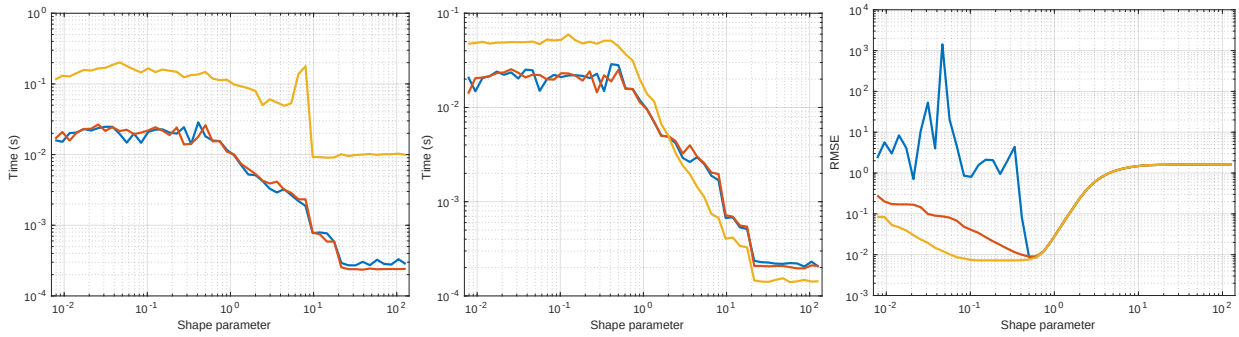
(a) CPU time for computing Λ^+. **(b)** CPU time for solving (1) directly. **(c)** RMSE on Υ.

Figure 5: Approximation of f_1 via $\varphi_{\varepsilon,W0}$. Legend: backslash, mult, mult+Tik.



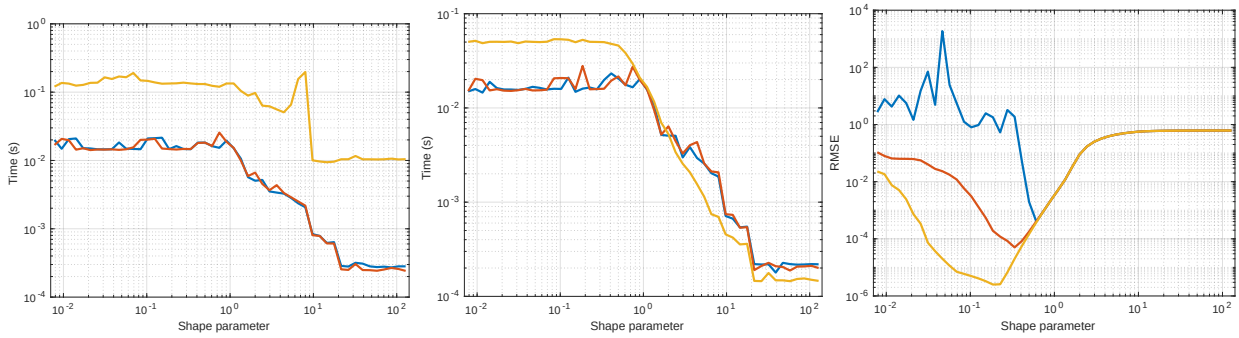
(a) CPU time for computing Λ^+. **(b)** CPU time for solving (1) directly. **(c)** RMSE on Υ.

Figure 6: Approximation of f_2 via $\varphi_{\varepsilon,W0}$. Legend: backslash, mult, mult+Tik.



(a) CPU time for computing Λ^+. **(b)** CPU time for solving (1) directly. **(c)** RMSE on Υ.

Figure 7: Approximation of f_1 via $\varphi_{\varepsilon,W6}$. Legend: backslash, mult, mult+Tik.



(a) CPU time for computing Λ^+. **(b)** CPU time for solving (1) directly. **(c)** RMSE on Υ.

Figure 8: Approximation of f_2 via $\varphi_{\varepsilon,W6}$. Legend: backslash, mult, mult+Tik.

Overall, the observations made in the previous subsection still hold true. Nevertheless, a couple of peculiarities can be highlighted.

- The **pinv method** is not taken into account here, as the MATLAB function `pinv` can not be employed with matrices of sparse type.
- There is no time advantage in using the **mult method** when solving (1) directly, while its usage leads to a faster computation of Λ^+ with respect to the **\ method**.

5 Conclusions

In this note, we discussed different strategies for the computation of the Moore-Penrose inverse in MATLAB. If on the one hand the usage of the built-in `pinv` function still represents a *safe* and default choice for concrete applications if no related information

is available, on the other hand various numerical experiments show that it does not represent the best choice for computing the pseudoinverse in certain approximation settings. Therefore, when possible, what is observed in this work may be taken into account in pursuing both a *fast* and an accurate reconstruction process.

Acknowledgements

This research has been accomplished within the Rete ITaliana di Approssimazione (RITA) and the thematic group on Approximation Theory and Applications of the Italian Mathematical Union, with the support of GNCS-INδAM too.

References

[1] R. Cavoretto, A. De Rossi. Software for Approximation 2022 (SA2022). *Dolomites Res. Notes Approx.*, Special Issue SA2022, 15:i–ii, 2022.

[2] G. E. Fasshauer. Meshfree Approximations Methods with MATLAB. *World Scientific*, Singapore, 2007.

[3] G. E. Fasshauer, M.J. McCourt. Kernel-based Approximation Methods Using MATLAB. *World Scientific*, Singapore, 2015.

[4] L. Ling, F. Marchetti. A stochastic extended Rippa's algorithm for LpOCV. *Appl. Math. Lett.*, 129:107955, 2022.

[5] Mathworks. Help center. *https://mathworks.com/help/*.

[6] R. Penrose. A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51:3, 406–413, 1955.

[7] R. Penrose. On best approximate solutions of linear matrix equations. *Mathematical Proceedings of the Cambridge Philosophical Society*, 52:1, 17–19, 1956.

[8] P. Petersen. Linear algebra. *Springer-Verlag*, New York, 2012.

[9] G. Wang, Y. Wei, S. Qiao. Generalized Inverses: Theory and Computations. *Springer, 2nd ed., Developments in Mathematics, Vol. 53*, Singapore, Science Press, Beijing, 2018.

[10] H. Wendland. Scattered Data Approximation. *Cambridge Monogr. Appl. Comput. Math.*, 17, Cambridge Univ. Press, Cambridge, 2005.

[11] R. A. Willoughby. Solutions of Ill-Posed Problems (A. N. Tikhonov and V. Y. Arsenin). *SIAM Review*, 21:2, 266–267, 1979.