Linearly scaling direct method for accurately inverting sparse banded matrices

Pablo García-Risueño[a,b,c], Pablo Echenique[a,b,c]

[a] *Instituto de Química Física "Rocasolano", CSIC, Serrano 119, E-28006 Madrid, Spain*
[b] *Departamento de Física Teórica, Universidad de Zaragoza, Pedro Cerbuna 12, E-50009 Zaragoza, Spain*
[c] *Instituto de Biocomputación y Física de Sistemas Complejos (BIFI), Universidad de Zaragoza, Mariano Esquillor s/n, Edificio I+D, E-50018 Zaragoza, Spain*

Abstract

In many problems in Computational Physics and Chemistry, one finds a special kind of sparse matrices, termed *banded matrices*. These matrices, which are defined as having non-zero entries only within a given distance from the main diagonal, need often to be inverted in order to solve the associated linear system of equations. In this work, we introduce a new $\mathcal{O}(n)$ algorithm for solving such a system, being $n \times n$ the size of the matrix. We produce the analytical recursive expressions that allow to directly obtain the solution, as well as the pseudocode for its computer implementation. Moreover, we review the different options for possibly parallelizing the method, we describe the extension to deal with matrices that are banded plus a small number of non-zero entries outside the band, and we use the same ideas to produce a method for obtaining the full inverse matrix. Finally, we show that the New Algorithm is competitive, both in accuracy and in numerical efficiency, when compared to a standard method based in Gaussian elimination. We do this using sets of large random banded matrices, as well as the ones that appear when one tries to solve the 1D Poisson equation by finite differences.

Keywords: banded matrix, sparse matrix, inversion, Gaussian elimination

1. Introduction

In this article we present efficient formulae and subsequent algorithms to solve the system of linear equations

$$Ax = b \ , \tag{1}$$

where A is a $n \times n$ matrix, x is the $n \times 1$ vector of the unknowns, b is a given $n \times 1$ vector and A satisfies that for known $m_u, m_l < n$

$$A_{I,I+K} = 0 \quad \forall\ K > m_u\ , \forall I\ , \tag{2}$$

$$A_{I+L,I} = 0 \quad \forall\ L > m_l\ , \forall I\ , \tag{3}$$

i. e., A is a *banded matrix* and (1) is a *banded system*. We also investigate how to solve a similar systems where there are some non-zero entries not lying in the diagonal band.

Banded systems like this are abundant in the computational physics and computational chemistry literature, specially because the discretization of differential equations, transforming them into fininte-differences equations, often results in banded matrices [1, 2]. Many examples of this can be found in boundary value problems in general [3, 4, 5], in fluid mechanics [6, 7, 8], thermodynamics [9], classical wave mechanics [3], structure mechanics [10], nanoelectronics [11], circuit analysis [12], or diffusion equations and Maxwell's first-order curl equations [2]. In quantum chemistry, finite difference methods using banded matrices are used both in the wavefunction formalism [13, 14, 15] and in density functional theory [16, 17]. In addition to finite-differences problems, banded systems not arising from discretization are also present in several areas, like constrained molecular dynamics [18] or the calculation of Lagrange multipliers in classical mechanics [19]. Moreover, banded matrix techniques are not only useful in linear systems, but also in linearized ones, which also appear frequently in the literature [4, 6, 7, 8, 9, 18].

The resolution of a linear system with A a $n \times n$ dense matrix requires $\mathcal{O}(n^3)$ floating point operations (or *flops*)[1] (A flop is a floating point operation, like addition, substraction, multiplication and quotient [21].). However, banded systems can be solved in $\mathcal{O}(nm_u m_l)$ flops using very simple recursive formulae, while the explicit form of A^{-1} can be obtained in $\mathcal{O}(n^2)$ flops. As mentioned before, there exist a number of physical problems whose behaviour is described with banded systems where $m_u, m_l \ll n$, which provides large computational savings. This is even more important in cases in which the calculation of relevant quantities requires many iterations. This is the case of molecular dynamics [22, 14], Monte Carlo simulations [23], quantum properties calculations via self-consistent field equations [24, 25], etc.

[1] As stated in [20], this can be reduced to $\mathcal{O}(n^{log_2 7 \approx 2.807})$.

In this work, we introduce a new algorithm for solving banded systems and inverting banded matrices which presents very competitive numerical properties, outperforming in many cases other commonly used techniques. Additionally, we provide the explicit recursive expressions in which the algorithm is based, thus facilitating further analytical developments stemming from it.

The article is structured as follows: in section 2, we derive simple recursive formulae for the efficient solution of a linear banded system. They enable to solve (1) in $\mathcal{O}(nm_u m_l)$ flops, and are suitable to be used in a serial machine. In section 3, we extend these formulae to the case of systems where some entries outside the band are also non-zero and provide some clues about the numerical cost of such extensions. In section 5, we briefly discuss the main issues related to the numerical performance of computer algorithms, and we comment on the schemes used for parallelizing linear banded solvers which can be combined with the technique presented here. In section 4, we cast the mathematical relations introduced in 2 into algorithmic form, providing the pseudocode of the basic algorithm and discussing possible alternatives. In section 6, we compare our new scheme with another popular banded solver in terms of both accuracy and numerical cost, using randomly generated banded systems; while, in section 7, we apply the method to a simple physical problem: the solution of the 1D Poisson equation. Finally, in section 8, we state the most important conclusions of the work. As extra material, in the Appendix, we provide equations for the explicit expression of the entries of A^{-1}.

2. Analytical solution of banded systems

One of the most common ways of solving the linear system in eq. (1) consists in gradually turning to zero the different entries of the matrix A in a process called *Gaussian elmination* [26, 27, 28]. This process is based on the possibility of writing A as $A = LU$, with L a lower triangular matrix and U an upper triangular one. This way of writing A, called *LU-decomposition*, is possible (i.e., L and U exist), if and only if A is invertible and all its leading principal minors are non-zero [29]. If one of the two matrices L and U is chosen to be *unit triangular*, i.e., with 1's on its diagonal, the matrices not only exist but they are also unique.

The analytical calculations and algorithms introduced in this work are based on a different but closely related property of A, namely, the possibility of finding Q a lower triangular matrix and P an upper triangular one, such

that we have

$$QAP = \mathbb{I} \quad \Rightarrow \quad A^{-1} = PQ \ , \tag{4}$$

where $\mathbb{I}$ is the identity matrix.

The requirements for these two matrices to exist are the same as those in the LU-decomposition, because, in fact, the two propositions are equivalent. That the existence of a '*QP-decomposition*' follows from the existence of the LU one is trivially proved if we make $Q = L^{-1}$ and $P = U^{-1}$. The converse implication follows from the fact that, for an invertible matrix A (it has to be invertible if we want that eq. (1) has a unique solution), the fact of its determinant, $\det A$, being different from zero and the relation $\det Q \det A \det P = \det \mathbb{I} = 1$ force both Q and P to have non-zero determinant and be, therefore, invertible. This allows to write $A = Q^{-1}P^{-1}$ and, since the inverse of a triangular matrix is also a triangular matrix of the same kind, we can identify $L = Q^{-1}$ and $U = P^{-1}$ thus proving the existence of the LU-decomposition. This equivalence also allows us to say that, as long as one of the two matrices Q and P is unit triangular, the QP-decomposition is unique.

An important qualification to this situation is that, in order to solve the system in eq. (1), the only options are not to QP (or LU) decompose A; we can also solve the system by performing a Gaussian elimination process that is based on the QP (or LU) decomposition of a matrix $\tilde{A}$, which is obtained from A by permuting its rows and/or columns. If these permutations, also called *pivoting* in the context of Gaussian elimination, are performed, the condition for $Q\tilde{A}P = \mathbb{I}$ (or $\tilde{A} = LU$) to hold is just that A is invertible, and the algorithms obtained from the pivoting case are tipically more stable. In what follows, we shall deal only with the non-pivoting case for simplicity, but the reader should notice that pivoting can be included in the discussion with minor adjustments. The algorithms derived in sec. 4 and implemented in the computer contain both the pivoting and the non-pivoting cases.

Therefore, let us now explicitly build the matrices P and Q that satisfy (4) for a given matrix A. When we know them, they can be used to compute the inverse A^{-1}, and then we will be able to solve (1). However, in this section (see also ref. [30]), we will see that there is no need to explicitly build A^{-1}, and the information needed to calculate P and Q can be used in a different way to solve (1).

We begin by writing P and Q as follows

$$P := P_1 P_2 \dots P_n = \prod_{K=1}^{n} P_K \ , \tag{5a}$$

$$Q := Q_n Q_{n-1} \dots Q_1 = \prod_{K=n}^{1} Q_K \ , \tag{5b}$$

being

$$P_K := \begin{pmatrix} 1 & & & & & & & & & \\ & \ddots & & & & & & & & \\ & & 1 & & & & & & & \\ & & & \xi_{KK} & \xi_{K,K+1} & \cdots & \xi_{K,K+m_u} & & & \\ & & & & 1 & & & & & \\ & & & & & \ddots & & & & \\ & & & & & & 1 & & & \\ & & & & & & & \ddots & & \\ & & & & & & & & 1 & \\ & & & & & & & & & 1 \end{pmatrix} , \tag{6}$$

and

$$Q_K := \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & \xi_{K+1,K} & 1 & & & & & \\ & & \vdots & & \ddots & & & & \\ & & \xi_{K+m_l,K} & & & 1 & & & \\ & & & & & & \ddots & & \\ & & & & & & & 1 & \\ & & & & & & & & 1 \end{pmatrix} , \tag{7}$$

where $K = 1, \dots, n$, and all the non-specified entries are zero. Note that P_K equals the identity matrix except in its K-th row, and Q_K equals the identity matrix except in its K-th column.

Now, the trick is to choose all coefficients ξ_{IJ} in the preceding matrices so that we have $QAP = \mathbb{I}$ in (4) (whenever the conditions for this to be possible are satisfied; see the beginning of this section).

First, we must notice that, given (6), multiplying an generic matrix G by its right by P_K is equivalent to adding the K-th column of G multiplied

by the corresponding ξ coefficients to several columns of G, at the same time that the K-th column of the original matrix is multiplied by ξ_{KK}:

$$(GP_K)_{IJ} = G_{IJ} \qquad \text{for } J < K \text{ and } J > K + m_u \ , \tag{8a}$$

$$(GP_K)_{IK} = G_{IK}\xi_{KK} \ , \tag{8b}$$

$$(GP_K)_{IJ} = G_{IJ} + G_{IK}\xi_{KJ} \qquad \text{for } K < J \leq K + m_u \ . \tag{8c}$$

If we take this into account, we can choose ξ_{11} so that $(AP_1)_{11} = 1$, and $(AP_1)_{1J} = 0$ for $J = 2, \ldots, n$. Given the fact that A is banded by hypothesis (see, in particular (2, 3)), we have that

$$(AP_1)_{11} = A_{11}\xi_{11} = 1 \ \Rightarrow \ \xi_{11} = 1/A_{11} \ , \tag{9a}$$

$$(AP_1)_{1J} = A_{1J} + A_{11}\xi_{1J} = 0 \ \Rightarrow \ \xi_{1J} = -\frac{A_{1J}}{A_{11}} \ , \quad 1 < J \leq 1 + m_u \ . \tag{9b}$$

Operating in this way, we have 'erased' (i.e., turned into zeros) the superdiagonal entries of A that lie on its first row, and we have done this by multiplying A by its right by P_1 with the appropriate ξ_{1J}. Then, if we multiply AP_1 by its right by P_2 and choose the coefficients ξ_{2J} in the analogous way, we can erase all the superdiagonal entries in the second row and turn to 1 its diagonal entry. In general, multiplying $AP_1 \cdots P_{K-1}$ by P_K erases the superdiagonal entries of the K-th row, and turns to 1 the diagonal entry. This way to proceed is called *Gaussian elimination* [29], and after n steps, the resulting matrix is the unit lower triangular matrix $A\prod_{K=1}^{n} P_K = AP$.

The expression for the coefficients ξ_{IJ}, with $I \leq J$ and $I > 1$ is more complex than (9) because, as a consequence of (8), whenever we multiply a matrix by its right by P_K, not only its K-th row (the one we are erasing) is affected, but also all the rows below (the m_l rows below in the case of a banded matrix like (4)). However, the matrix $A\prod_{L=1}^{K-1} P_L$ is 0 in all its superdiagonal entries belonging to the first $K-1$ rows, and multiplying it by its right by P_K has no influence on these rows. Hence, the fact that we have chosen to erase the superdiagonal entries of A from the first row to the last row allows us to express the general conditions that the ξ coefficients belonging to different P_K's must satisfy in the following way:

$$\left(A \prod_{K=1}^{I} P_K \right)_{II} = 1 \ , \tag{10a}$$

$$\left(A \prod_{K=1}^{I} P_K \right)_{IJ} = 0 \quad \text{for } I < J \ . \tag{10b}$$

Now, using (10a) together with (8b), we can derive the following expression for the coefficient ξ_{II} in terms of the previous steps of the process:

$$\left(A\prod_{K=1}^{I}P_K\right)_{II} = \left(A\prod_{K=1}^{I-1}P_KP_I\right)_{II} = \left(A\prod_{K=1}^{I-1}P_K\right)_{II}\xi_{II} = 1$$
$$\Longrightarrow \quad \xi_{II} = \frac{1}{\left(A\prod_{K=1}^{I-1}P_K\right)_{II}} \ . \tag{11}$$

Analogously, using (10b) and (8c), we can write an explicit expression for ξ_{IJ} with $I < J$:

$$\left(A\prod_{K=1}^{I-1}P_KP_I\right)_{IJ} = \left(A\prod_{K=1}^{I-1}P_K\right)_{IJ} + \left(A\prod_{K=1}^{I-1}P_K\right)_{II}\xi_{IJ} = 0$$
$$\Longrightarrow \quad \xi_{IJ} = -\frac{\left(A\prod_{K=1}^{I-1}P_K\right)_{IJ}}{\left(A\prod_{K=1}^{I-1}P_K\right)_{II}} = -\left(A\prod_{K=1}^{I-1}P_K\right)_{IJ}\xi_{II} \ . \tag{12}$$

Also according to (8), for $I \leq J$

$$\left(A\prod_{K=1}^{L}P_K\right)_{IJ} = A_{IJ} + \sum_{M=J-m_u}^{L}\left(A\prod_{K=1}^{M-1}P_K\right)_{IM}\xi_{MJ} \ , \quad I > L \ . \tag{13}$$

Note that, in this equation we have $I \leq M$. The reason is that, due to (5a) and (6), all the terms that are multiplied by coefficients ξ and added to the I, J entry (with $I < J$) correspond to subdiagonal entries of $A\prod_{L=1}^{K-1}P_L$. This allows us to calculate the coefficients ξ_{IJ} with $I > J$, i.e., those that correspond to the matrices Q_K, once all the coefficients in the matrices P_K have been already evaluated. Since we know that AP is a unit lower triangular matrix, this means that its subdiagonal I, M entry (with $I > M$) equals ξ_{IJ}, because no other changes affect this entry when multiplying AP by the different Q_K's. Hence,

$$\left(A\prod_{L=1}^{M-1}P_L\right)_{IM} = -\xi_{IM} \quad \text{for } I > M \ . \tag{14}$$

If we apply this on the right hand side of (13), and we insert the resulting expression with $J = I$ and $L = I - 1$ into (11), and also insert it with

$L = I - 1$ into (12), we get the recursive equations we were looking for:

$$\xi_{II} = \left(A_{II} - \sum_{M=I-m_u}^{I-1} \xi_{IM}\xi_{MI} \right)^{-1} , \tag{15a}$$

$$\xi_{IJ} = \xi_{II} \left(-A_{IJ} + \sum_{M=I-m_u}^{I-1} \xi_{IM}\xi_{MJ} \right) \quad \text{for } I < J . \tag{15b}$$

Since eq. (13) also holds for $I > J$, if we join it to (14), we get

$$\xi_{IJ} = -A_{IJ} + \sum_{M=I-m_u}^{I-1} \xi_{IM}\xi_{MJ} \quad \text{for } I > J . \tag{16}$$

These results can be further modified with the aim of improving the numerical efficiency of the algorithms derived from them. The starting point for the summations in (15) must be the value of M such that both ξ_{IM} and ξ_{MJ} are non-zero. We must take into account that in a banded matrix the number of nonzero entries above and on the left of the I, J entry depends on the values of I, J:

- There are $m_u + (I - J)$ non-zero entries immediately above A_{IJ}.
- There are $m_l - (I - J)$ non-zero entries immediately on the left of A_{IJ}.

These properties are also satisfied in $A \prod_{L=1}^{K} P_L$ for all K. Therefore, if we define

$$\begin{aligned} \mu_{IJ} &:= \min\{m_u + (I - J), m_l - (I - J)\} , \\ \mu' &:= \min\{m_u, m_l\} , \end{aligned}$$

we can re-express (15) as

$$\xi_{II} = \left(A_{II} - \sum_{M=\max\{1,I-\mu'\}}^{I-1} \xi_{IM}\xi_{MI} \right)^{-1} , \tag{17a}$$

$$\xi_{IJ} = \xi_{II} \left(-A_{IJ} + \sum_{M=\max\{1,I-\mu_{IJ}\}}^{I-1} \xi_{IM}\xi_{MJ} \right) \qquad \text{for } I < J , \tag{17b}$$

$$\xi_{IJ} = -A_{IJ} + \sum_{M=\max\{1,J-\mu_{IJ}\}}^{J-1} \xi_{IM}\xi_{MJ} \qquad \text{for } I > J . \tag{17c}$$

In the restricted but very common case in which $m_l = m_u =: m$, the previous equations become

$$\xi_{II} = \left(A_{II} - \sum_{M=max(1,I-m)}^{I-1} \xi_{IM}\xi_{MI} \right)^{-1} , \tag{18a}$$

$$\xi_{IJ} = \xi_{II} \left(-A_{IJ} + \sum_{M=\max\{1,J-m\}}^{I-1} \xi_{IM}\xi_{MJ} \right) \qquad \text{for } I < J , \tag{18b}$$

$$\xi_{IJ} = -A_{IJ} + \sum_{M=\max\{1,I-m\}}^{J-1} \xi_{IM}\xi_{MJ} \qquad \text{for } I > J . \tag{18c}$$

The reader must also note that, although the coefficients ξ_{IJ} have been obtained performing the products $\prod_{K=n}^{1} Q_K A \prod_{L=1}^{n} P_L$ in a certain order, they are independent of this choice. Indeed, if we take a look to expressions (5a), (5b), (6), and (7), we can see that the K-th row (or column) is always erased before the $(K+1)$-th one. It does not matter if we apply first Q_K or P_K to erase the K row (or column). In both cases the result of the operation will be the same: to add $-G_{IK}G_{KJ}/G_{KK}$ (where $G := \prod_{M=K-1}^{1} Q_M A \prod_{L=1}^{K-1} P_L$) to all the entries of G_{IJ} such that $I \in \{K+1, \ldots, K+m_l\}$ and $J \in \{K+1, \ldots, K+m_u\}$. This is valid when both the K-th row and the K-th column are not erased yet. If one of them is already erased, erasing the other has no influence on G_{IJ} with $I \in \{K+1, \ldots, K+m_l\}$ and $J \in \{K+1, \ldots, K+m_u\}$. In both cases $\xi_{IK} = -G_{IK}/G_{KK}$ for $I > K$, and $\xi_{KJ} = -G_{KJ}/G_{KK}$ for $J > K$. This is because all the previous rows (or columns) are erased before, and then adding columns (or rows) has no influence on the K-th one.

Now, the algorithm to solve (1) can be divided into three stages (in our implementation we join together the first and second ones). Since $A^{-1} = PQ$ (4), these steps are:

1. Get the coefficients ξ.
2. Obtain the intermediate vector $c := Qb$.
3. Obtain the final vector $x = Pc$.

Using the results derived above, let us calculate the expressions for the second and third steps:

Whenever we multiply a generic $n \times 1$ vector v by the left by Q_K (see (7)), we modify its K-th to $(K + m_l)$-th rows in the following way:

$$(Q_K v)_I = v_I \qquad \text{for } I \leq K, \quad I > K + m_l \ , \tag{19a}$$

$$(Q_K v)_I = v_I + \xi_{IK} v_K \qquad \text{for } K < I \leq K + m_l \ . \tag{19b}$$

Since $Q := Q_n Q_{n-1} \dots Q_1$, using the expression for each of the Q_K in (7), and the fact that $\xi_{IJ} = 0$ for $I > J + m_l$, we have

$$Q_{IJ} = 0 \qquad \text{for } I < J \ , \tag{20a}$$

$$Q_{II} = 1 \ , \tag{20b}$$

$$Q_{IJ} = \sum_{M=\max\{I-m_l,1\}}^{I-1} \xi_{IM} Q_{MJ} \qquad \text{for } I > J \ . \tag{20c}$$

where the maximum in the lower limit of the sum accounts for border effects and ensures that M is never smaller than 1.

From these relations among the entries of Q we can get the components c_I of the intermediate vector c in the second step above:

$$\begin{aligned} c_I &= \sum_{J=1}^{n} Q_{IJ} b_J = \sum_{J=1}^{I} Q_{IJ} b_J = b_I + \sum_{J=1}^{I-1} \left(\sum_{M=\max\{I-m_l,1\}}^{I-1} \xi_{IM} Q_{MJ} \right) b_J \\ &= b_I + \sum_{M=\max\{I-m_l,1\}}^{I-1} \xi_{IM} \sum_{J=1}^{I-1} Q_{MJ} b_J \\ &= b_I + \sum_{M=\max\{I-m_l,1\}}^{I-1} \xi_{IM} c_M \ , \end{aligned} \tag{21}$$

where we have used that Q is a lower triangular matrix (20a), then (20c) and finally a feedback in the equation.

We will now turn to the third and final step of the process, which consists of calculating the final vector $x = Pc$.

Whenever we multiply a generic $n \times n$ matrix G by the left by P_K (see eq. (6)), the resulting matrix is the same as G in all its rows except for the K-th one, which is equal to a linear combination of the first $m_u + 1$ rows below it:

$$(P_K G)_{IJ} = G_{IJ} \qquad \text{for } I \neq K \ , \tag{22a}$$

$$(P_K G)_{KJ} = \sum_{L=K}^{\min\{K+m_u,n\}} \xi_{KL} G_{LJ} \ , \tag{22b}$$

where the minimum in the upper limit of the sum accounts for border effects and ensures that $K + L$ is never larger than n.

If we now use the relations above to construct P as in (5a), i.e., we first take P_n and multiply it by the left by P_{n-1}, then we multiply the result, $P_{n-1}P_n$, by the left by P_{n-2}, etc., we arrive to:

$$P_{II} = \xi_{II} \ , \tag{23a}$$

$$P_{IJ} = \sum_{K=I+1}^{\min\{I+m_u,n\}} \xi_{IK} P_{KJ} \qquad \text{for } I < J \ , \tag{23b}$$

$$P_{IJ} = 0 \qquad \text{for } I > J \ , \tag{23c}$$

meaning that every row of P is a linear combination of the following rows, plus a term in the diagonal.

These expressions allow us to obtain the last equation that is necessary to solve the linear system in (1):

$$\begin{aligned} x_I &= \sum_{J=1}^{n} P_{IJ} c_J = \sum_{J=I}^{n} P_{IJ} c_J = \xi_{II} c_I + \sum_{J=I+1}^{n} \left(\sum_{K=I+1}^{\min\{I+m_u,n\}} \xi_{IK} P_{KJ} \right) c_J \\ &= \xi_{II} c_I + \sum_{K=I+1}^{\min\{I+m_u,n\}} \xi_{IK} \sum_{J=I+1}^{n} P_{KJ} c_J \\ &= \xi_{II} c_I + \sum_{K=I+1}^{\min\{I+m_u,n\}} \xi_{IK} x_K \ . \end{aligned} \tag{24}$$

Now, we can use expressions (17a), (17b), and (17c) in order to obtain the coefficients ξ, and them plug them into (21) and (24) to finally solve (1).

To close this section, let us focus on the computational cost of this procedure: From (17a), (17b), and (17c), it follows that obtaining the coefficients ξ requires $\mathcal{O}(n)$ flops. Being more precise, the summations in (17a), (17b), and (17c), require μ_{IJ} products and $\mu_{IJ} - 1$ additions ($2\mu_{IJ} - 1$ flops). If, without loss of generality, we consider $m_u \geq m_l$, it is easy to check that the following computational costs hold:

- Obtaining one diagonal ξ_{II} takes $2\mu' + 2 \simeq 2m_u$ flops.

- Obtaining one superdiagonal coefficient ξ_{IJ} (where $I < J$) takes about $2\min\{m_l, m_u - (J-I)\}$ flops. Hence, in order to obtain all the coefficients in a column above the diagonal, there are two sets of ξ's that require a different number of operations. The lower one requires $2m_l$ flops; the upper one $2(m_u - (J-I))$ flops. All in all, obtaining $\xi_{J-I,J}$ for $I = 1, \ldots, m_u$ takes about $m_l(2m_u - m_l)$ flops.

- In order to obtain the ξ_{IJ} coefficient in a subdiagonal row ($I > J$), the number of flops to be performed is $\min\{m_u, m_l - (I-J)\} = m_l - (I-J)$; in such a way that the total number of flops related to this row is approximately m_l^2.

Finally, obtaining all coefficients ξ would take slightly less (due to the border effects) than $2nm_um_l$ flops. Once they are known (or partly during the process to get them), we can obtain the solution vector x using the simple recursive relationships presented in this section at a cost of $4n(m_u + m_l)$ flops.

3. Banded plus sparse systems

A slight modification of the calculations presented in the previous section is required to tackle systems where not all the non-zero entries are within the band. The resulting modified procedure is described in this section.

If we have

$$A' := A + \sum_{T=1}^{T_{max}} A'_{R_T S_T} \Delta^{R_T S_T} \ , \tag{25}$$

with A banded (see eqs. (2) and (3)) and the matrix $\Delta^{R_T S_T}$ consisting of entries $(\Delta^{R_T S_T})_{IJ} = \delta_{I,R_T}\delta_{J,S_T}$, being δ_{IJ} the Kroenecker delta, we shall say that A' is a *banded plus sparse* matrix, and

$$A'x = b \tag{26}$$

a *banded plus sparse* system.

In the pure banded system (section 2) only $\xi_{K,K+J}$ and $\xi_{K+I,K}$ with $K = 1, \ldots, n$; $J = 1, \ldots, m_u$; $I = 1, \ldots, m_l$ had to be calculated. In this case, we also need to get

$$\begin{aligned} &\xi_{IS_T} \qquad \text{if } R_T < S_T, \text{ for } I = R_T, R_T + 1, \ldots, S_T - m_u - 1 \ , \\ &\xi_{R_T J} \qquad \text{if } R_T > S_T, \text{ for } J = S_T, S_T + 1, \ldots, R_T - m_l - 1 \ , \end{aligned}$$

with $T = 1, \ldots, T_{max}$.

This is so because, when erasing an extra-band entry above the diagonal $\xi_{R_T S_T}$ (through Gaussian elimination, i.e., multiplying by a P_{R_T} matrix) new non-zero entries are created below. When erasing (eliminating) an extra-band entry below the diagonal, new non-zero entries are created on its right.

If we define

$$\nu_{R_T I} := \max\{R_T, I - m_l\} \ , \tag{27a}$$

$$\rho_{S_T J} := \max\{S_T, J + m_u\} \ , \tag{27b}$$

then, if $R_T < S_T$, we have

$$\xi_{R_T S_T} = -A'_{R_T S_T} \xi_{R_T R_T} \ , \tag{28a}$$

$$\xi_{IJ} = \sum_{M=\nu}^{I-1} \xi_{IM} \xi_{MJ} \qquad \text{for } R_T < I < J - m_u \ ; \tag{28b}$$

and if $R_T > S_T$,

$$\xi_{R_T S_T} = -A'_{R_T S_T} \ , \tag{29a}$$

$$\xi_{IJ} = \sum_{M=\rho}^{J-1} \xi_{IM} \xi_{MJ} \qquad \text{for } S_T < J < I - m_l \ . \tag{29b}$$

These equations have to be modified for $I < J$ if there exist $A'_{R_x S_T}$ with $R_x < R_T$. This is so because erasing the upper non-zero entries by adding columns creates new non-zero entries below them, and the new relations must take this into account. Analogous corrections must be done for $I > J$ if there exist $A'_{R_T S_x}$ with $S_x > S_T$. The way to build the correct formula for ξ_{IJ} in each case is to use eqs. (17a), (17b), and (17c) allowing summations to start by $M = 1$ and avoiding to include the ξ_{KL} that are known a priori to be zero.

The computational cost of solving banded plus systems scales with n, as long as the number of columns above the band and rows below it containing non-zero entries $A'_{R_T S_T}$ is small ($\ll n$).

4. Algorithmic implementation

Based on the expressions derived in the previous sections (17, 21, 24), we have coded a number of different algorithms that efficiently solve the linear

system in (1). The difference between the method in this work and the most commonly used implementation of Gaussian elimination techniques, such as the ones included in LAPACK [26], Numerical Recipes in C [27], or those discussed in ref. [28] is that these methods perform an LU factorization of the matrix A, and the coefficients ξ for the Gaussian elimination are obtained in several steps, whereas the method introduced here does not perform such a LU factorization, and it obtains the coefficients ξ in a single step.

In order to obtain the solution of (1) we need to get the coefficients ξ for Gaussian elimination as explained in section 4. That is, one diagonal coefficient for each row/column, m_u coefficients in each row and m_l coefficients in each column (except for the last ones, where less coefficients have to be calculated). Higher accuracies in the solution are obtained by pivoting, i.e., altering the order of the rows and columns in the process of Gaussian elimination so that the *pivot* (the element temporarily in the diagonal and by which we are going to divide) is never too close to zero. Double pivoting (in rows *and* columns) usually gives more accurate results than partial pivoting (in rows *or* columns). However, the former is seldom preferred for banded systems, since it requires $\mathcal{O}(n^2)$ operations, while the latter requires only $\mathcal{O}(n)$. In the implementations described in this section, we choose to perform partial pivoting on rows, as they do in refs. [27, 26]. In the same spirit, and in order to save as much memory as possible, we store matrices by diagonals.

We proceed as follows: For each given I, we obtain ξ_{II} (using (17a)), and then ξ_{JI} (using (17c)) for $J = I + 1, \ldots, I + m_l$. If $|\xi_{JI}| > |\xi_{II}|$, we exchange rows I and J in the matrix A and in the vector b. This is called partial pivoting in rows, and it usually gives greater numerical stability to the solutions; in our tests of section 6 the error was lowered in two orders of magnitude by partial pivoting. Next, we calculate ξ_{IJ} (using (17b)) for $J = I+1, \ldots, I+m_u$. When we have calculated all the relevant coefficients ξ for a given I, we calculate c_I using (21). We repeat these steps for all rows I, starting by $I = 1$ and moving one row at a time up to $I = n$. This ordering enables us to solve the system using eqs. (17), (21), (24), because the superdiagonal ξ_{IJ} (i.e., those with $I < J$) only require the knowledge of the coefficients with a lower row index I, while the subdiagonal coefficients ξ_{IJ} with $I > J$ only require the knowledge of coefficients with a lower column index. We have additionally implemented a procedure to avoid performing dummy summations (i.e., those where the term to add is null),

which eliminates the need for evaluating μ_{IJ} in every step. According to the pivotings performed before starting to calculate a given ξ, a different number of terms will appear in the summation to obtain it. This procedure uses the previous pivoting (i. e., row exchanging) information and determines how many ξ coefficients have to be obtained in any row or column, and how many terms the summation to obtain them will consist of (this procedure is not indicated in the pseudo-code below for the sake of simplicity). The final step consists of obtaining x from b using (24).

The pseudo-code of the algorithm can be summarized as follows:

```
// Steps 1 and 2: Calculating the coefficients ξ and the vector c
for (K = 1, K ≤ n, K++) do
  // Calculating the diagonal ξ's:
  ξ_KK = 1/(A_KK + Σ_{M=K−μ'}^{K−1} ξ_KM ξ_MK)
  // Calculating the subdiagonal ξ's:
  for (I = K + 1, I ≤ I + m_l, I++) do
    ξ_IK = −A_IK + Σ_{M=K−μ_IK}^{K−1} ξ_IM ξ_MK      for I > J
  end for

  // Pivoting:
  if ∃ |ξ_JI| > |ξ_II| for J = I + 1, ..., I + m_l then
    for (K = 1, K ≤ n, K++) do
      A_IK ↔ A_JK
    end for
    b_I ↔ b_J
  end if

  // Calculating the superdiagonal ξ's:
  for (J = K + 1, J ≤ K + m_u, J++) do
    ξ_KJ = −ξ_KK A_KJ + Σ_{M=K−μ_KJ}^{K−1} ξ_KM ξ_MJ
  end for

  // Calculating c_K = (Qb)_K:
  c_K = b_K
  for (L = K − m_l, L ≤ K − 1, L++) do
    c_K += ξ_{K,L} c_L
  end for
end for
```

// Step 3: Calculating $x_K = (Pc)_K = (PQb)_K$:
for $(K = n,\ K \geq 1,\ K--)$ **do**
  $x_K = \xi_{KK} c_K$
  for $(L = K + 1,\ L \leq K + m_u,\ L++)$ **do**
    $x_K += \xi_{K,L} x_L$
  end for
end for

In the actual computer implementation we split the most external loop into three loops ($I = 1, \ldots, 2m$, $I = 2m + 1, \ldots, n - 2m$, and $I = n - 2m + 1, \ldots, n$), because the summations to obtain the coefficients ξ lack some terms in the initial and final rows. We store A by diagonals in a $n \times (2m_u + m_l + 1)$ matrix in order to save memory and, with the same objective, we overwrite the original entries A_{IJ} with the calculated ξ_{IJ} for $I \leq J$, and we store the ξ_{IJ} with $I > J$ in another $n \times (2m_l)$ matrix.

One possible modification to the algorithm presented above is to omit the pivoting. This usually implies larger errors in the solution, but results in important computational savings. It can be used in problems where computational cost is more important that achieving a very high accuracy. In any case, one must note that the accuracy of the algorithm is typically acceptable without pivoting, so in many cases no pivoting will be necessary.

In (24) we can see that no subdiagonal coefficients (ξ_{IJ} with $I > J$) are needed to obtain x from c. In (21), we can see that only ξ_{IK} are necessary in order to obtain c_I, thus making it unnecessary to know ξ_{LK} for $L < I$. Therefore, we can get rid of them once c_I is known. Since we calculate c_I immediately after calculating all ξ_{IK}, we can overwrite $\xi_{I+1,K}$ on the memory position of ξ_{IK}. If we do so, about one third of the memory is saved, since less coefficients must be stored, however, according to some preliminary tests, this option is also 20% slower than the simpler one in which all coefficients are stored independently.

5. Parallelization

There exist many works in the literature aimed to parallelize the calculations needed to solve a banded system [1, 10, 21, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41]. The decision about which one to choose, and, in particular, which one to apply to the algorithms presented in this work depends, of course, on the architecture of the machine in which the calculations are

going to be performed. The choice is additionally complicated by the fact that, normally, only the number of floating point operations (flops) required by each scheme is reported in the articles. However, the number of flops is known to be a poor measure of the real wall-clock performance of computer algorithms and, specially, parallel ones, for a number of reasons:

- Not all the flops require the same time. For example, in currently common architectures, a quotient takes 4 times as much cycles as an addition or a product.
- A flop usually requires to access several positions of memory. Each access is much slower than the flop itself [42]. Moreover, the number of memory accesses does not need to be proportional to the number of flops.
- Transferring information among nodes in a cluster is commonly much slower than accessing a memory position or performing a floating point operation [42].

Despite this unavoidable complexities and the fact that rigorous tests should be made in any particular architecture, two parallelizing schemes seem well suited for the method presented in this work: the one in ref. [32] for shared-memory machines and the one in ref. [10] for distributed-memory machines. The former is faster if the communication time among nodes tends to zero, whereas the latter tackles the communication time problem by significantly reducing the number of messages that need to be passed.

6. Results and discussion

In the previous sections, we derived explicit expressions to directly solve banded systems, and we described their algorithmic implementation. Numerical tests have been performed to check the validity of the new scheme.

In order to assess its performance, we both measure its absolute accuracy and numerical efficiency and compare them with those of the banded solver described in the well-known book *Numerical Recipes in C* [27]. This solver, like the one in ref. [28], belongs to a popular family of algorithms which work by calculating the ξ coefficients for the Gaussian elimination procedure in different iterations. Since the coefficients result from the summation of several terms, they first obtain the first term of the summation of several ξ's, then all the second terms, and so on. In contrast, our method gets first

the final value of a given ξ by calculating all the terms in the corresponding summation; and then, once a given ξ_{IJ}, is known, it computes $\xi_{I,J+1}$.

In the comparisons in this section, we consider $m_u = m_l = m$ for simplicity. We took $n = 10^3, 10^4$ and 10^5 and $m = 3, 10, 30, 100$ and 300 for all them in our tests. In addition to this, we took $n = 10^6$ with $m = 3, 10$ and 30. For each given pair of values of n and m, we generated a set of 1000 random banded matrices. They are $n \times n$, and all their entries are null, except the diagonal ones and their first m neighbours on the right and on the left. The value of these entries is a random number between 500 and -500 with 6 figures. The components of the independent term (the vector b in (4)) are random numbers between 0 and 1000, also with 6 figures. We tested both algorithms (that from [27] and ours) with and without pivoting. We used PowerPC 970FX 2.2 GHz machines, and no specific optimization flags have been given to the compiler. Every point in our performance plots corresponds to the mean of 1000 tests and, in each point, we use the same random system as the input for both algorithms. With respect to the measure of the wall-clock time, we consider only the computation time, not the time for initializing the system and displaying the results.

The errors E are measured using the normalized deviation of Ax from b if we use as vector x the solution provided by the numerical method:

$$\text{Error} := \frac{\sum_{I=1}^{n} |\sum_{J=1}^{n} A_{IJ} x_J - b_I|}{\sum_{I=1}^{n} |x_I|} \ . \tag{30}$$

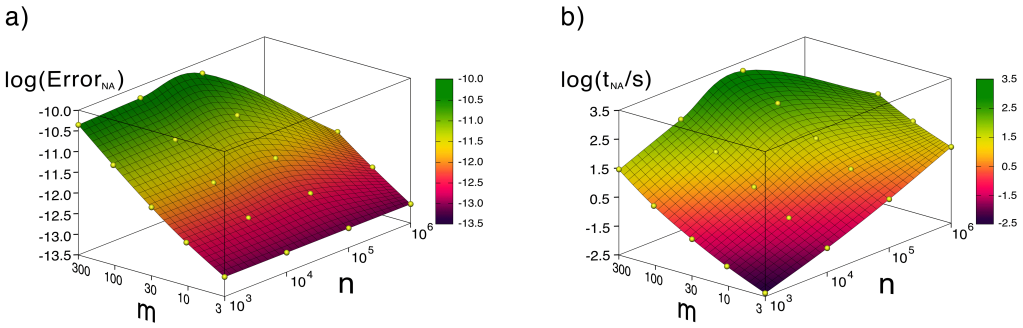

Figure 1: Properties of the New Algorithm introduced in this work, with pivoting, as a function of the size of the matrix n and the width of the band m in random banded test systems. **a)** Its accuracy, as measured by the error defined in eq. (30). **b)** Its numerical efficiency, measured by the execution time.

In the plots in this section we present the absolute and relative accuracy and efficiency of the New Algorithm (NA) introduced in this work to the aforementioned banded solver appearing in Numerical Recipes in C (NRC), [27] (NRC), for the cases with and without pivoting. The yellow spheres in the plots represent the calculated points, which correspond to the average of 1000 tests with different input random matrices and vectors. For the sake of visual confort, interpolating surfaces have been produced with cubic splines, also, the x and y axes are in logarithmic scale. In the figures in which we compare quantities between the two algorithms, a blue plane at $z = 1$ divides the n, m regions where one algorithm or the other is to be preferred.

In figure 1a, we can see that our algorithm with pivoting has a very good accuracy, with the error being approximately constant with n, and increasing with m as a power law, $\log(\text{Error}) \propto \log(m)$, with a small exponent (~ 1.4). The execution time in the tested region (see fig. 1b) is proportional to n, and also approximately proportional to $m^{1.7}$, not to m^2 as one could expect from the number of flops ($\propto nm^2$). This suggests that memory access is an important time-consuming factor, in addition to floating point operations.

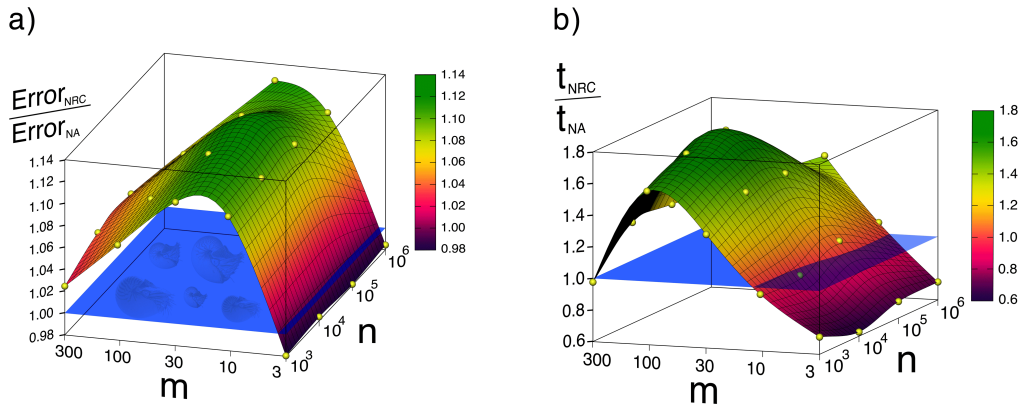

Figure 2: Comparison between the properties of the New Algorithm introduced in this work (NA) and the one in ref. [27] (NRC), both with pivoting, as a function of the size of the matrix n and the width of the band m in random banded test systems. **a)** Relative accuracy, as measured by the quotient of the errors defined in eq. (30). **b)** Relative numerical efficiency, measured by the quotient of the execution times.

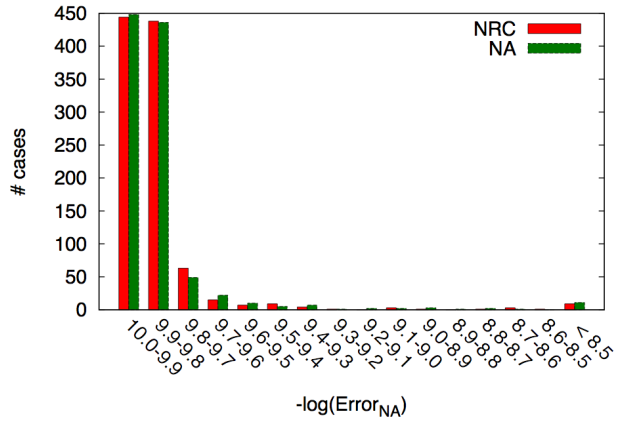

Figure 3: Histogram of the errors made by the algorithms NA and NRC for solving random banded systems without pivoting. The data corresponds to 1000 random inputs with $n = 10^5$ and $m = 10$.

In fig. 2a, we can see that, if pivoting is performed, our new algorithm is always more accurate than NRC, except for a narrow range of m between 1 and 4; the typical increase in accuracy is around a 5%, reaching almost a 15% for some values of n and m. In fig. 2b, we can see that, if pivoting is present, the New Algorithm is also faster than NRC for most of the studied values of n and m; being typical speedups of around a 40% and the largest ones of almost a 80%.

If pivoting is not performed, the accuracy decreases typically by two or three orders of magnitude (but errors still remain very low, usually around 10^{-10}). We also see that, in the non-pivoting case, a few of the calculations (around 1 in 500) present errors significantly larger than the average, probably suggesting that the random procedure has produced a matrix that is close to singular with respect to the hypotheses introduced in sec. 2. We show a typical example of the distribution of errors for the non-pivoting banded solvers NA and NRC in fig. 3. The data corresponds to the errors of 1000 random input matrices with $n = 10^5$ and $m = 10$. In such a case, the average of the error is less representative. In this example test, the highest error in NRC is $1.57 \cdot 10^{-9}$, and in NA is $2.53 \cdot 10^{-9}$, although these numbers are probably anecdotical. One must also note that 99% of the errors are $\mathcal{O}(10^{-10})$ or smaller. A comparison of the red and green bars in the histogram suggests that there are not big differences in the errors of both algorithms (NA and NRC) without pivoting.

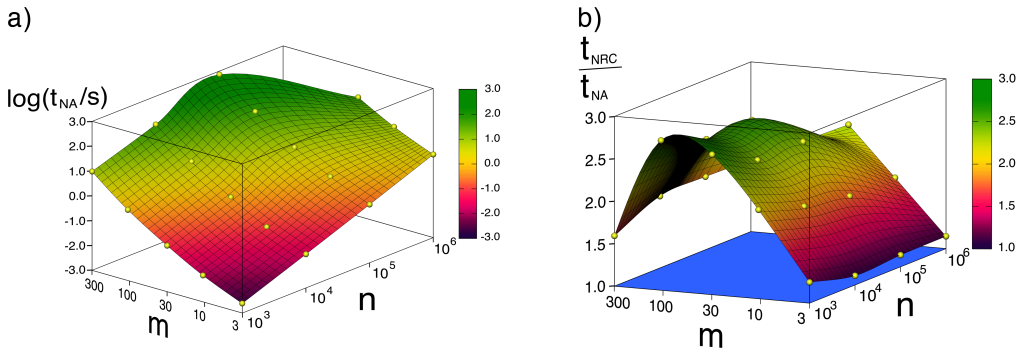

Figure 4: **a)** Numerical complexity of the New Algorithm introduced in this work, without pivoting, when solving random banded test systems. Execution time as a function of the size of the matrix n and the width of the band. **b)** Comparison between the numerical complexity of NA and NRC.

Despite these problems in dealing with almost singular matrices, not pivoting has an important advantage regarding computational cost, and it may be useful for problems in which the matrices are a priori known to be well behaved. These computational savings are noticed if we compare figs. 1b, and 4a. In fig. 4b, we can additionally see that the New Algorithm introduced in this work is always faster than NRC for the explored values of n and m if no pivoting is performed; the increase in efficiency reaching almost a factor 3 for some values of n and m, and being typically around a factor 2.

To sum up, we can conclude that, for the matrices tested in this section, the new algortithm introduced in this work is competitive both in accuracy and in computational efficiency when compared with a standard method for inverting banded matrices. This holds true both with and without pivoting.

7. An application: Efficient solution of a 1D finite-differences problem

In order to test the New Algorithm in a real (albeit simple) physical problem, we used it, as well as the NRC algorithm studied in the previous section, to solve the Poisson equation (31) in one dimension with a (known) independent term $b(x)$ chosen to be Gaussian.

$$\frac{d^2}{dx^2} f(x) = b(x) \ . \tag{31}$$

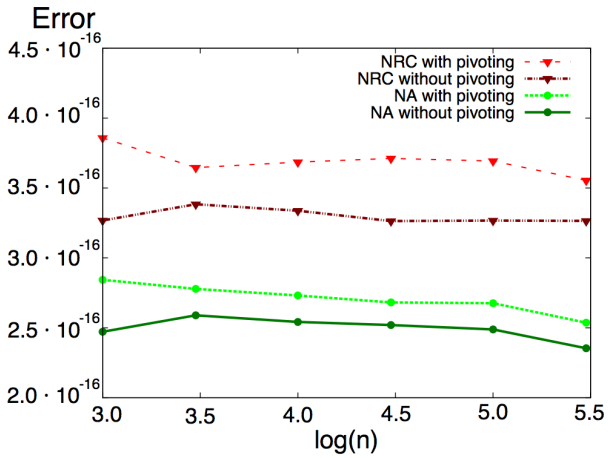

Figure 5: Comparison of the error of the New Algorithm (NA) and NRC for solving the fourth-order 1D Poisson equation in a mesh. The error is calculated using eq. (30).

If we discretize the Laplace operator of second derivatives, d^2/dx^2, into n points, an $n \times n$ matrix A appears that needs to be inverted in order to solve the problem. If the discretization is chosen to be fourth order, which is more accurate than the usual second order one, A is a banded matrix with $m_u = m_l = m = 4$ [17].

We solved the problem for different values of n in a computer with a 2.26 GHz Intel Core 2 Duo. We found that, in this particular problem, and although errors are very small in any case (see fig. 5), the algorithms with pivoting give results that are always less accurate than those of the algorithms without pivoting. Since pivoting requires additional time, it is clear that we should use algorithms without pivoting. In this case, the New Algorithm is better than NRC for all the tested values of n, despite the low value of m, as it can be observed in fig. 5. We can also notice that the error does not increase with the size of the system n, as it happened with the random matrices in the previous section. In the case without pivoting (which is the recommended case, as we mentioned) the difference in computational efficiency between the two algorithms is around 2-3%, with the New Algorithm outperforming NRC.

For this particular problem, the logarithm of the execution times in the tested range of n follows an almost perfect linear relationship $\log(t/s) = a\,\log(n) + b$, being the parameters:

- NRC with pivoting: $a = 1.001$, $b = -5.912$
- NA with pivoting: $a = 1.001$, $b = -5.779$
- NRC without pivoting: $a = 1.002$, $b = -6.005$
- NA without pivoting: $a = 1.000$, $b = -6.006$

8. Concluding remarks

We have introduced in this work a new method to invert the banded matrices that so often appear in Computational Physics and Chemistry, as well as in other disciplines. We have shown that this new algorithm is capable of being more accurate than standard methods based on Gaussian elimination at a lower computational cost, which opens the door to its use in many practical problems, such as the ones described in the introduction.

Moreover, we have produced the analytical expressions that allow to directly obtain, in a recursive manner, the solution to the associated linear system in terms of the entries of the original matrix. To have at our disposal these explicit formulae, which have also been presented for the calculation of the full inverse matrix in the Appendix, not only simplifies the task of coding the needed computer algorithms, but it may also be useful to facilitate analytical developments in the problems in which banded matrices appear.

Aknowledgments

The authors would like to thank J. L. Alonso, G. Ciccotti, J. M. Peña and Á. Rubio for illuminating discussions and useful advices, and to M. García-Risueño for helping with the plots. This work has been supported by the research projects E24/3 (DGA, Spain), FIS2009-13364-C02-01 (MICINN, Spain) and 200980I064 (CSIC, Spain). P. G.-R. is supported by a JAE PREDOC grant (CSIC, Spain).

Appendix A. Inverse of a banded matrix

In the previous sections, we proved that the banded linear system of n equations with n unknowns in (1) can be solved in order n operations. Sometimes, we can be interested on obtaining the inverse matrix A^{-1} itself. We can do it in order n^2 operations using the same kind of ideas discussed

in the main body of the article. It is worth to be stressed that the explicit inverse of an arbitrary banded matrix usually cannot be obtained in $\mathcal{O}(n)$ flops, since the inverse of a banded matrix has n^2 entries and it is not, in general, a banded matrix itself (an exception to this is a block diagonal matrix). In order to obtain an efficient way to invert A, in this appendix we will derive some recursive relations between the rows of P (and Q). To this end, we will first calculate the explicit expression of the entries of these matrices.

Using eqs. (5), (6), and (7), from which (8) and (19) follow, and after some straightforward but long calculations, one can show that the aforementioned entries satisfy

$$P_{IJ} = \xi_{JJ} \sum_{c \in C^{\uparrow}_{I,J,m_u}} \prod_{(K,L)\in c} \xi_{KL} \qquad \text{for } I < J \ , \tag{A.1a}$$

$$P_{II} = \xi_{II} \ , \tag{A.1b}$$

$$P_{IJ} = 0 \qquad \text{for } I > J \ , \tag{A.1c}$$

and

$$Q_{IJ} = \sum_{c \in C^{\downarrow}_{I,J,m_l}} \prod_{(K,L)\in c} \xi_{KL} \qquad \text{for } I > J \ , \tag{A.2a}$$

$$Q_{II} = 1 \ , \tag{A.2b}$$

$$Q_{IJ} = 0 \qquad \text{for } I < J \ . \tag{A.2c}$$

We call the summations appearing in the first line of each of this groups of expressions *jump summation* from I to J with increasing indices ($\uparrow$) and m_u neighbours, and *jump summation* from I to J with decreasing indices ($\downarrow$) and m_l neighbours, respectively. The jump summation provides us with an explicit expression for all entries in A^{-1}, without the need to recursively refer to other entries. This can be useful in order to parallelize its calculation.

As it can be seen in the expressions, each product in the sums contains a number of coefficients ξ_{KL}. The pairs of indices (K, L) which are included in a given product are taken from a set c. In turn, each term of the sum corresponds to a different set of pairs indices c drawn from a set of sets of pairs of indices $C^{\uparrow}_{I,J,m_u}$ (in the case of P_{IJ}) or $C^{\downarrow}_{I,J,m_l}$ (in the case of Q_{IJ}). Therefore, the only detail that remains to understand these 'jump summations' is to specify which are the elements of these latter sets.

A given element c of either $C^{\uparrow}_{I,J,m_u}$ or $C^{\downarrow}_{I,J,m_l}$ can be expressed as

$$c = \{(K_1, L_1), (K_2, L_2), \ldots, (K_S, L_S)\} \ , \tag{A.3}$$

in such a way that $C^{\uparrow}_{I,J,m_u}$ comprises all possible such c's that comply with a number of rules:

- $K_1 = I$, and $L_S = J$.
- $K_r < L_r$, for $r = 1, \ldots, S$.
- $K_{r+1} = L_r$, for $r = 1, \ldots, S$.
- $L_r - K_r \leq m_u$, for $r = 1, \ldots, S$.

Let us see an example:

$$\sum_{c \in C^{\uparrow}_{3,6,2}} \prod_{(K,L)\in c} \xi_{KL} \;=\; \xi_{3,4}\xi_{4,5}\xi_{5,6} + \xi_{3,4}\xi_{4,6} + \xi_{3,5}\xi_{5,6}\ . \tag{A.4}$$

The rules to determine the elements of $C^{\downarrow}_{I,J,m_l}$ are analogous to the ones above but taking into account that the indices decrease:

- $K_1 = I$, and $L_S = J$.
- $K_r > L_r$, for $r = 1, \ldots, S$.
- $K_{r+1} = L_r$, for $r = 1, \ldots, S$.
- $K_r - L_r \leq m_l$, for $r = 1, \ldots, S$.

An example would be:

$$\begin{aligned}\sum_{c \in C^{\downarrow}_{5,1,3}} \prod_{(K,L)\in c} \xi_{KL} \;=\; & \xi_{5,4}\xi_{4,3}\xi_{3,2}\xi_{2,1} + \xi_{5,3}\xi_{3,2}\xi_{2,1} + \xi_{5,4}\xi_{4,2}\xi_{2,1} \\ & + \xi_{5,4}\xi_{4,3}\xi_{3,1} + \xi_{5,3}\xi_{3,1} + \xi_{5,2}\xi_{2,1} + \xi_{5,4}\xi_{4,1}\ .\end{aligned} \tag{A.5}$$

If we first focus on P, it is easy to see that, according to the properties of the jump summation, we have

$$\begin{aligned}\sum_{c \in C^{\uparrow}_{I,J,m_u}} \prod_{(K,L)\in c} \xi_{KL} \;=\; & \xi_{I,I+1} \sum_{c \in C^{\uparrow}_{I+1,J,m_u}} \prod_{(K,L)\in c} \xi_{KL} \\ & + \xi_{I,I+2} \sum_{c \in C^{\uparrow}_{I+2,J,m_u}} \prod_{(K,L)\in c} \xi_{KL} \\ & + \ldots + \xi_{I,I+m_u} \sum_{c \in C^{\uparrow}_{I+m_u,J,m_u}} \prod_{(K,L)\in c} \xi_{KL}\ .\end{aligned} \tag{A.6}$$

If we insert a multiplicative factor ξ_{JJ} at both sides and use (A.1), this equation becomes eq. (23) obtained in sec. 2:

$$P_{IJ} = \sum_{K=I+1}^{\min\{I+m_u,n\}} \xi_{IK} P_{KJ} \qquad \text{for } I < J \ .$$

An analogous expression for Q can be obtained in a similar way:

$$Q_{IJ} = \sum_{L=J+1}^{\min\{J+m_l,n\}} Q_{IL} \xi_{LJ} \qquad \text{for } I > J \ .$$

Now, if we define $\mu_1 := \min\{m_u, J-I\}$, and $\mu_2 := \min\{m_l, I-J\}$, since $A^{-1} = PQ$ (see (4)), we have that

$$\begin{aligned}
(A^{-1})_{IJ} &= (PQ)_{IJ} = \sum_{K=1}^{n} P_{IK} Q_{KJ} = \sum_{K=J}^{n} P_{IK} Q_{KJ} = \\
&= \sum_{K=J}^{n} \left(\sum_{L=I+1}^{I+\mu_1} P_{LK} \xi_{IL} \right) Q_{KJ} = \sum_{L=I+1}^{I+\mu_1} \xi_{IL} \left(\sum_{K=J}^{n} P_{LK} Q_{KJ} \right) \\
&= \sum_{L=I+1}^{I+\mu_1} \xi_{IL} (A^{-1})_{LJ} \qquad \text{for } I < J \ , \qquad \text{(A.7)}
\end{aligned}$$

which is a recursive relationship for the superdiagonal entries of A^{-1}.

Peforming similar computations, we have

$$(A^{-1})_{II} = \xi_{II} + \sum_{L=I+1}^{I+\mu_1} \xi_{IL} (A^{-1})_{LJ} = \xi_{II} + \sum_{L=J+1}^{J+\mu_2} \xi_{LJ} (A^{-1})_{IL} \ , \qquad \text{(A.8)}$$

$$(A^{-1})_{IJ} = \sum_{L=J+1}^{J+\mu_2} \xi_{LJ} (A^{-1})_{IL} \qquad \text{for } I > J \ . \qquad \text{(A.9)}$$

Using the last three equations, we can easily construct an algorithm to compute A^{-1} in $\mathcal{O}(n^2)$ flops. This algorithm would first calculate $(A^{-1})_{nn} = \xi_{nn}$. Then it would use (A.7) to obtain, in this order, $(A^{-1})_{n-1,n}$, $(A^{-1})_{n-2,n}$, ..., $(A^{-1})_{1n}$. These are the superdiagonal ($I < J$) entries of the n-th column. Then, it would use (A.9) to obtain, in this order, $(A^{-1})_{n,n-1}$, $(A^{-1})_{n,n-2}$, ..., $(A^{-1})_{n1}$, i.e., the subdiagonal ($I > J$) entries of the n-th row. Once the n row and column of A^{-1} are known, $(A^{-1})_{n-1,n-1}$ can be

obtained with (A.8), and then (A.7) and (A.9) can be used to obtain the entries of this $(n-1)$ column and row, respectively. When calculating the entries of a column J, i.e., ξ_{KJ} with $K < J$, ξ_{LJ} is to be obtained always before $\xi_{L-1,J}$. When calculating the entries of a row I, i.e., ξ_{IK} with $I > K$, ξ_{IK} is to be obtained always before $\xi_{I,K-1}$. This procedure can be repeated for all rows and columns of A^{-1}, and the calculation of the K-th row and column can be performed in parallel.

References

[1] J. Dongarra, S. L. Johnson, Solving banded systems on a parallel processor, Parallel Computing 5 (1987) 219–246.

[2] J. Hyman, J. Morel, M. Shashkov, S. Steinberg, Mimetic finite difference methods for diffusion equations, Computational Geosciences 6 (2002) 333–352.

[3] R. E. Shaw, L. E. Garey, A fast method for solving second order boundary value volterra integro-differential equations, International Journal of Computer Mathematics 65, 1-2 (1997) 121–129.

[4] M. Paprzycki, I. Gladwell, Solving almost block diagonal systems on parallel computers, Parallel Computing 17 (1991) 133–153.

[5] S. J. Wright, Stable parallel algorithm for two-point boundary value problems, SIAM J. Sci. Stat. Comput. 13 (1992) 742–764.

[6] W. R. Briley, H. McDonald, Solution of the multidimensional compressible navier-stokes equations by a generalized implicit method, JCOP 24, 4 (1977) 372–397.

[7] P. D. Ariel, Hiemenz flow in hydromagnetics, Acta Mechanica 103 (1992) 31–43.

[8] O. M. Haddad, M. A. Al-Nimr, G. H. Shatnawi, Free convection gas flow in porous micro-channels, Selected Papers from the WSEAS Conferences in Spain, September 2008 Santander, Cantabria, Spain (2008).

[9] O. Haddad, M. Abuzaid, M. Al-Nimr, Entropy generation due to laminar incompressible forced convection flow through parallel-plates microchannel, Entropy 6, (5) (2004) 413–416.

[10] E. Polizzi, A. H. Shameh, A parallel hybrid banded system solver: the SPIKE algorithm, Parallel Computing 32 (2006) 177–194.

[11] E. Polizzi, N. Ben Abdallah, Subband decomposition approach for the simulation of quantum electron transport in nanostructures, Journal of Computational Physics 202, 1 (2004) 150–180.

[12] A. Lumsdaine, J. White, D. Webber, A. Sangiovanni-Vincentelli, A band relaxation algorithm for reliable and parallelizable circuit simulation, Research Laboratory of Electronics Dept. of Electrical Engineering and Computer Science Massachusetts Institute of Technology Cambridge, MA 02139, CH2657-5/88/0000/0308 01.000 1988IEEE (1988).

[13] J. M. Sanz-Serna, I. Christie, A simple adaptive technique for nonlinear wave problems, Journal of Computational Physics 67, 2 (1986) 348–360.

[14] R. Guantes, S. C. Farantos, High order finite difference algorithms for solving the schrdinger equation in molecular dynamics, Journal of Chemical Physics 111, 24 (1999) 10827–10835.

[15] R. Guardiola, J. Ros, On the numerical integration of the schrdinger equation in the finite-difference schemes, Journal of Computational Physics 111, 24 (1999) 374–389.
[16] A. Castro, H. Appel, M. Oliveira, C. A. Rozzi, X. Andrade, F. Lorenzen, M. A. L. Marques, E. K. U. Gross, A. Rubio, Octopus: a tool for the application of time-dependent density functional theory, Phys. Stat. Sol 243 (2006) 2465.
[17] M. A. L. Marques, A. Castro, G. F. Bertsch, A. Rubio, octopus: a first-principles tool for excited electron-ion dynamics, Computer Physics Communications 151 (2003) 60.
[18] J. P. Ryckaert, G. Ciccotti, H. J. C. Berendsen, Numerical integration of the Cartesian equations of motion of a system with constraints: Molecular dynamics of n-alkanes, J. Comput. Phys. 23 (1977) 327–341.
[19] M. Mazars, Holonomic constraints: An analytical result, Journal of Physics A: Mathematical and Theoretical 40, 8 (2007) 1747–1755.
[20] V. Strassen, Gaussian elimination is not optimal, Numerische Mathematik 13 (1969) 354–356.
[21] S. Chandrasekaran, M. Gu, Fast and stable algorithms for banded plus semiseparable systems of linear equations, SIAM Journal on Matrix Analysis and its Applications 25(2) (2003) 373–384.
[22] J. L. Alonso, X. Andrade, P. Echenique, F. Falceto, D. Prada-Gracia, A. Rubio, Efficient formalism for large-scale ab initio molecular dynamics based on time-dependent density functional theory, Phys. Rev. Lett. 101 (2008) 096403.
[23] W. K. Hastings, Monte Carlo sampling methods using markov chains and their applications, Biometrika 57, 1 (1970) 97–109.
[24] P. Echenique, J. L. Alonso, A mathematical and computational review of Hartree-Fock SCF methods in Quantum Chemistry, Mol. Phys. 105 (2007) 3057–3098.
[25] O. V. Gritsenko, A. Rubio, L. C. Balbs, J. A. Alonso, Self-consistent local-density approximation with model coulomb pair-correlation functions for electronic systems, Physical Review A 47 (1993) 1811–1816.
[26] E. Anderson, Z. Bai, C. e. a. Bischof, LAPACK User's Guide, SIAM, Philadelphia, release 3.0 edition, (1999).
[27] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Numerical recipes. The art of scientific computing, Cambridge University Press, New York, 3nd edition, (2007).
[28] D. S. Wakins, Fundamentals of Matrix Computations, Wiley Inter-Science, New York, 2nd edition, 1991.
[29] G. H. Golub, C. F. Van Loan (Eds.), Matrix Computations, The Johns Hopkins University Press, Baltimore and London, 2nd edition, 1993.
[30] A. Castro, M. A. L. Marques, A. Rubio, Propagators for the time-dependent Kohn-Sham equations, J. Chem. Phys. 121 (2004) 3425–3433.
[31] D. A. Bini, B. Meini, Effective methods for solving banded toeplitz systems, SIAM J. Matrix Anal. Appl. 20 (1999) 700–719.
[32] U. Meier, A parallel partition method for solving banded systems of linear equations, Parallel Computing 2 (1985) 33–23.
[33] H. Zhang, W. F. Moss, Using parallel banded linear system solvers in generalizad eigenvalue problems, Parallel Computing 20, 8 (1994) 1089–1105.
[34] D. H. Lawrie, A. H. Sameh, The computation and communications complexity of a

parallel banded system solver, ACM Transactions on Mathematical Software 10, 2 (1984) 185–195.
[35] S. L. Johnson, Solving narrow systems on ensemble architectures, ACM Transactions on Mathematical Software 11 (1985) 271–288.
[36] S. C. Chen, D. J. Kuck, A. H. Sameh, Practical parallel band triangular system solvers, ACM Transactions on Mathematical Software 4 (1978) 270–277.
[37] D. J. Evans, M. Hatzopoulos, The solution of certain banded systems of linear equations using the holding algorithm, The Computer Journal 19, 2 (1976) 184–187.
[38] L. E. Garey, R. E. Shaw, Solving banded and near symmetric systems, Applied Mathematics and Computation 5, 311 (2000) 133–143.
[39] P. Arbenz, W. Gander, A survey of direct parallel algorithms for banded linear systems, Technical Report TR 221, Inst. for Scientific Comp., ETH, Zrich (1994).
[40] G. H. Golub, A. H. Sameh, V. Sarin, A parallel balance scheme for banded linear systems, Numerical linear algebra with applications 8 (2001) 297–316.
[41] J. Dongarra, A. H. Sameh, On some parallel banded system solvers, Parallel Computing 1 (1984) 223–235.
[42] J. L. Hennessy, D. A. Patterson, Computer Architecture, A Quantitative Approach, Morgan Kaufmann - Elsevier, San Mateo, CA, 3rd edition, 2003.