

Improving NNMFPACK with heterogeneous and efficient kernels for β -divergence metrics

N. Díaz-Gracia · A. Cocaña-Fernández · M. Alonso-González ·
F. J. Martínez-Zaldívar · R. Cortina · V. M. García-Mollá ·
P. Alonso · J. Ranilla · A. M. Vidal

Published online: 27 December 2014
© Springer Science+Business Media New York 2014

Abstract NNMFPACK is a library for the nonnegative matrix factorization (NNMF) problem. Nowadays NNMF is an essential tool in many fields spanning machine learning, data analysis, image analysis or audio source separation, among others. NNMFPACK is an efficient numerical library conceived for shared memory heterogeneous

N. Díaz-Gracia · A. Cocaña-Fernández · M. Alonso-González · R. Cortina · J. Ranilla (✉)
Departamento de Informática, Universidad de Oviedo, Gijón, Spain
e-mail: ranilla@uniovi.es

N. Díaz-Gracia
e-mail: noeliadiazgracia@gmail.com

A. Cocaña-Fernández
e-mail: cocanaalberto@gmail.com

M. Alonso-González
e-mail: monica300876@gmail.com

R. Cortina
e-mail: raquel@uniovi.es

F. J. Martínez-Zaldívar
Departamento de Comunicaciones, Universitat Politècnica de València, Valencia, Spain
e-mail: fjmartin@dcom.upv.es

V. M. García-Mollá · A. M. Vidal
Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València, Valencia, Spain
e-mail: vmgarcia@dsic.upv.es

A. M. Vidal
e-mail: avidal@dsic.upv.es

P. Alonso
Departamento de Matemáticas, Universidad de Oviedo, Oviedo, Spain
e-mail: palonso@uniovi.es

parallel systems, and it supports, from its conception, both conventional multi-core processors and many-core coprocessors. In this article, NNMFPACK is extended to handle different metrics options (β -divergence), and some other parallel algorithms have been added and tested. The performance of the new functionalities of NNMFPACK is tested, and some precision results of the implementations are showed using an example borrowed from the image processing field.

Keywords NNMF · Parallel library · GPU · Intel MIC · Multi-core · Many-core

1 Introduction

The nonnegative matrix factorization (NNMF) has become a very important tool in fields such as document clustering, data mining, machine learning, data analysis, image analysis, audio source separation or bioinformatics [1–6]. NNMF consists on approximating a matrix $A \in \mathbb{R}^{m \times n}$ by the product of two matrices W and H , with some conditions: the matrix A has nonnegative elements, and $W \in \mathbb{R}^{m \times k}$ and $H \in \mathbb{R}^{k \times n}$ with $k \leq \min(m, n)$ are two lower rank matrices with nonnegative elements too, such that $A \approx WH$. The problem is addressed as the computation of two matrices W_0, H_0 such that

$$\|W_0 H_0 - A\|_F = \min_{W, H \geq 0} \|WH - A\|_F. \quad (1)$$

Other norms can be used instead of the Frobenius norm (see, for instance, [7], where the NNMF is also defined in terms of the Kullback–Leibler divergence). Also, many algorithms have been proposed for NNMF calculation (see [5, 7–11]).

The relevance of this factorization is the accomplished dimensionality reduction that effectively works as a compression tool for many applications, because the matrix A is approximated as the product of two lower rank matrices. Though useful, NNMF is also a computationally demanding task, what encourages to develop efficient routines capable of reducing its high execution time.

Besides, modern computer architectures have evolved from CPUs with a reasonable number of cores (multi-core) to heterogeneous systems where CPUs are aided by hardware accelerators with a huge number of computational cores (many-core). Therefore, these complex architectures (heterogeneous architectures) are an essential tool for tackling the NNMF when we cope with large scale matrices.

Previous results on the parallelization of some algorithms for calculating the NNMF can be found in [12]. In [13] a first approach to a library, called NNMFPACK, for NNMF calculation was presented. NNMFPACK is an efficient numerical library conceived for shared memory heterogeneous parallel systems, and it supports, from its conception, both conventional multi-core processors and (many-core) coprocessors such as Intel Xeon Phi and CUDA compatible Graphics Processing Units (GPUs). Its routines are also invocable from MATLAB/Octave through MEX interfaces, what increases NNMFPACK's usability. In [13] only some initial aspects of the library were depicted.

Although initially a wide variety of audio signal processing applications (see, for example, [3, 14, 15]) has inspired the development of the library, our interest is to provide a general tool (not constrained to audio problems) which can be used in

any other field where the NNMF decomposition is required. For example, if digital image processing (face recognition, optical character recognition, content based image retrieval, etc.) is considered, each monochrome digital image is a rectangular array of pixels and each pixel is represented by its light intensity. Since the light intensity is measured by a nonnegative value, we can represent each image as a nonnegative matrix, where each element is a pixel. Colour images can be coded in the same way but with several nonnegative matrices.

Therefore, in this article the library is extended to cope with different metrics options (β -divergence), and some other parallel algorithms have been added and tested. As a result, we include the evaluation of the overhead performance of the new routines on several parallel architectures (GPU, multi-core, etc.).

The remainder of the article is as follows. In Sect. 2 we define the different metrics used in the library, and the computational cost of the implemented algorithms is shown. The experimental advantages of the implemented algorithms are analysed on different architectures in Sect. 3, together with an example related to image reconstruction. The article ends with a section devoted to conclusions.

2 Approximation and cost functions

To find an approximate factorization $A \approx WH$, it is necessary to define cost functions that quantifies the quality of the approximation. The factorization is usually sought after through the minimization problem

$$\min_{W, H \geq 0} D(A|WH), \quad (2)$$

where $D(A|WH)$ is a cost function defined by

$$D(A|WH) = \sum_{i=1}^m \sum_{j=1}^n d([A]_{ij} |[WH]_{ij}), \quad (3)$$

and $d(x|y)$ is a scalar cost function.

Lee and Seung proposed in [7] the use of the Euclidean distance

$$d(x|y) = \frac{1}{2}(x - y)^2 \quad (4)$$

and the Kullback–Leibler divergence defined by

$$d(x|y) = x \log \frac{x}{y} - x + y. \quad (5)$$

Thus, for example, considering the Euclidean distance and using gradient descent algorithms to solve the minimization problem, they obtain the next update rules

$$H \leftarrow H \cdot \frac{W^T A}{(W^T W)H}, \quad W \leftarrow W \cdot \frac{AH^T}{W(HH^T)}, \quad (6)$$

where the symbol \cdot and the fraction bar denote entrywise matrix product and division, respectively.

In our previous work (see [12]), the multiplicative algorithms used exclusively the Frobenius norm as a measure of the goodness of the approximation of the matrices A and WH . In many applications of the NNMF it is more useful the utilization of another metrics to measure the closeness between these matrices. It can be empirically checked that the β -divergence metrics (introduced by Eguchi and Minami, see [16]) provide more accurate results for specific problems, while maintaining, under certain conditions, its convergence.

The new built-in algorithms are a generalization of that presented in [12], and they are parameterized and optimized to use different types of metrics based on the value of the parameter β : the Frobenius norm ($\beta = 2$), the Kullback–Leibler divergence ($\beta = 1$), the Itakura–Saito divergence ($\beta = 0$) or any other for different values of parameter β .

In the NNMFPACK library we have implemented subroutines that use as cost function the β -divergence, that can to be defined as (see, e.g. [17])

$$d_\beta(x|y) := \begin{cases} \frac{1}{\beta(\beta-1)}(x^\beta + (\beta - 1)y^\beta - \beta xy^{\beta-1}), & \text{if } \beta \in \mathbb{R} \setminus \{0, 1\}, \\ x(\log x - \log y) + (y - x), & \text{if } \beta = 1 \\ \frac{x}{y} - \log \frac{x}{y} - 1, & \text{if } \beta = 0. \end{cases} \tag{7}$$

The previous cost function is defined for all real number, but usually when is used in applications, β takes values between 0 and 2. Thus, in our experiments we tested only values for β varying in this range.

Taking into account [17] and using the gradient criterion, it is possible to obtain the following rules to update the matrices H and W :

$$H \leftarrow H \cdot \frac{W^T((WH)^{\beta-2} \cdot A)}{W^T(WH)^{\beta-1}}, \quad W \leftarrow W \cdot \frac{(WH)^{\beta-2} \cdot A H^T}{(WH)^{\beta-1} H^T}, \tag{8}$$

where $X.^n$ denotes the matrix with entries $([X]_{ij})^n$.

In the routines implemented in the library, the parameter β can be chosen by the user, thus allowing the use of different metrics.

2.1 Computational costs

The algorithm described in Sect. 2 is a multiplicative uniform cost algorithm with approximately

$$2(6mnk + 3mn + mk + nk) \tag{9}$$

flops per iteration if $\beta \in \mathbb{R} \setminus \{0, 1\}$, where n , m and k are the dimensions of the input matrix $A \in \mathbb{R}^{m \times n}$ and the two lower rank matrices $W \in \mathbb{R}^{m \times k}$ and $H \in \mathbb{R}^{k \times n}$, with $k \leq \min(m, n)$. If the value of the parameter β is 2 (corresponding with the Frobenius norm) the overall computational cost in flops per iteration will be

$$2(6mnk + mk + nk). \tag{10}$$

The cost of the NNMFPACK's MLSA algorithm (see [13]), an efficient implementation of the Lee and Seung algorithm [7], is

$$4kmn + 2k(m+n)(2k+1), \quad (11)$$

flops per iteration, which is lower than (10).

This cost can be obtained by assuming the flop definition given in [18] and considering that in expression (6) the cost of basic operations such as matrix–matrix multiplication ($2mnk$ flops for $X = YZ$, with $X \in \mathbb{R}^{m \times n}$, $Y \in \mathbb{R}^{m \times k}$, $Z \in \mathbb{R}^{k \times n}$) or point-wise multiplication/division ($2mn$ flops for $X = Y \cdot Z/T$, with $X, Y, Z \in \mathbb{R}^{m \times n}$) are known.

Thus, in the updating of (6) the following operations must be done:

$$(W^T W) \rightarrow 2mk^2 \text{ flops} \quad (12)$$

$$(W^T W)H \rightarrow 2k^2n \text{ flops} \quad (13)$$

$$W^T A \rightarrow 2mnk \text{ flops} \quad (14)$$

$$H \cdot (W^T A) / ((W^T W)H) \rightarrow 2kn \text{ flops} \quad (15)$$

$$(HH^T) \rightarrow 2nk^2 \text{ flops} \quad (16)$$

$$W(HH^T) \rightarrow 2k^2m \text{ flops} \quad (17)$$

$$AH^T \rightarrow 2mnk \text{ flops} \quad (18)$$

$$W \cdot (AH^T) / (W(HH^T)) \rightarrow 2mk \text{ flops} \quad (19)$$

The overall cost per iteration, (11), is obtained by adding all these partial costs.

Although MLSA and β -divergence with $\beta = 2$ are mathematically equivalent, MLSA's cost is lower due to a rearrangement of its operations. Therefore, when β is equal to 2, MLSA is used instead of the β -divergence version.

Finally, if β is 1 some matrix operations can be substituted by vector operations and, therefore, the computational cost can be approximated by

$$2(4mnk + 2mk + mn + nk) \quad (20)$$

flops per iteration.

3 Experimental results

In [13] NNMFPACK's design principles and general outline of its functionality are given. In addition, [13] shows the installation procedure and gives some examples of use. A complete description of NNMFPACK can be found in its website [19]. The algorithms presented in this work are included in NNMFPACK according to the specifications given in [13, 19].

Furthermore, for better comparison with previous empirical results, the experiments carried out in this article were obtained using the same test bed system. That is, an ASUS server with

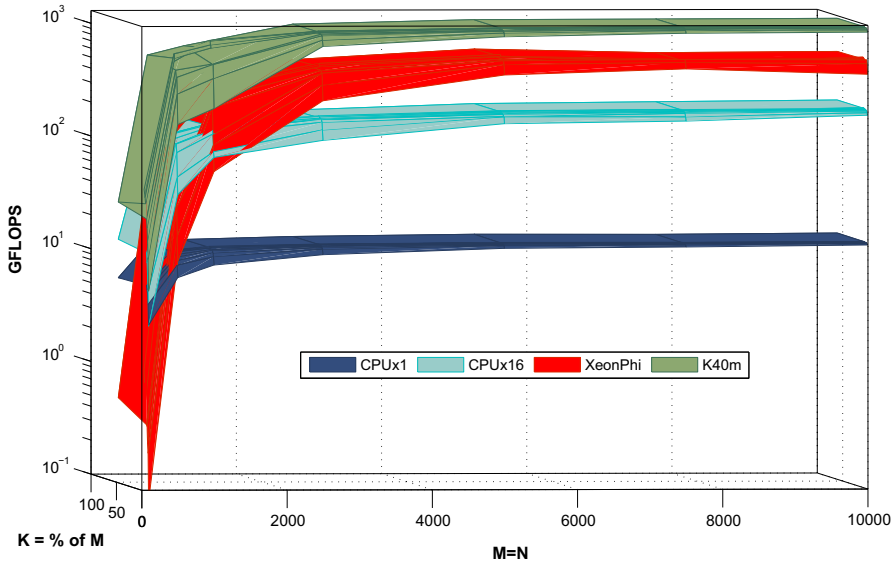


Fig. 1 NNMFPACK performance for β Div algorithm

- Two Intel Xeon E5-2650 CPUs @ 2.0 GHz with 64 GB of RAM. Their peak performance in double precision is 256 GFLOPS.
- One NVIDIA Tesla K40m GPU with 2880 cores @ 745 MHz, 12 GB of DDR5 RAM and a peak performance of 1.43 TFLOPS in double precision.
- One Intel Xeon Phi 5110P coprocessor with 60 cores @ 1.053 GHz, 8 GB RAM and a peak performance of 1.01 TFLOPS in double precision.
- CPU and Xeon Phi codes were compiled with the Intel C compiler, version 14.0.2.144, and Intel MKL, version 11.1.2.
- For the Tesla GPU, a NVIDIA NVCC compiler, version 6.0.37, and MAGMA, version 1.4.1 have been used.
- The Matlab version used is the 8.1.0.604 (R2013a), 64-bit, and the GNU Octave version is the 3.4.3 64-bit. Both libraries have been executed on the ASUS server cited above.

Since the aim of this work is to analyse the behaviour of the new algorithms and not to compare the efficiency of all NNMFPACK’s kernels, the experiments are focused on the performance of the general case ($\beta \in \mathbb{R} \setminus \{0, 1\}$) of the β -divergence algorithms (β Div in onwads).

The analysis performed has been carried out using as the input matrix $A \in \mathbb{R}^{m \times n}$ a uniformly random generated positive square matrix ($m = n$). For a more comprehensive study, for each m (or n), value k (the inner dimension of the multiplication approximation) ranges from 10 to 100 % of m (or n) with step 10 %.

Figure 1 shows the performance, in terms of GFlops, of the β Div algorithm, and Table 1 presents the theoretical percentage achieved by empirical results over the different architectures for some values of m , n and k .

Table 1 Theoretical performance percentage achieved by empirical results

| $n = m$ | k | CPUx1 (%) | CPUx16 (%) | Xeon Phi (%) | K40m (%) |
|---------|-------|-----------|------------|--------------|----------|
| 1,000 | 500 | 83.9 | 61.2 | 31.4 | 41.5 |
| 2,500 | 1,250 | 89.8 | 74.6 | 56.4 | 76.1 |
| 5,000 | 2,500 | 93.4 | 83.8 | 67.6 | 82.5 |
| 7,500 | 3,750 | 93.5 | 86.7 | 68.4 | 85.4 |
| 10,000 | 5,000 | 95.2 | 90.0 | 72.1 | 85.4 |

Table 2 MATLAB/Octave and NNMFPACK execution time for β Div algorithm (seconds)

| $n = m$ | k | CPUx1 | CPUx16 | Xeon Phi | K40m | Matlab | Octave |
|---------|-------|-------|--------|----------|------|--------|--------|
| 5,000 | 1,500 | 30.1 | 2.2 | 0.7 | 0.4 | 2.5 | 57.7 |
| 7,500 | 2,250 | 102.6 | 7.1 | 2.3 | 1.3 | 7.1 | 193.9 |
| 10,000 | 3,000 | 237.5 | 15.0 | 5.7 | 3.0 | 15.2 | 458.3 |

Figure 1 and Table 1 show that when using the 16 cores available in the testing machine (CPUx16 in Fig. 1), the CPU implementation achieves about the 90 % of its peak performance (i.e. with $m = n = k = 10,000$), while the sequential algorithm (CPUx1 legend in Fig. 1) achieves more than 95 % of it. Xeon Phi's performance overcome 70 % of its peak performance, reaching more than 700 GFLOPS. In a similar vein, Tesla K40m reaches up to 1240 GFLOPS, more than the 85 % of its peak performance. These percentages are consistent with those presented in [13], slightly lower due to the greater number of non Level 3 BLAS operations per iteration than in NNMFPACK's MLSA algorithm.

It is noteworthy that, when establishing a comparison between MATLAB/Octave functions and β Div, an initial adjustment of the input parameters of MATLAB/Octave is needed, so equity conditions are guaranteed. In this way, MATLAB/Octave's MULT NNMF driver (call `nnmf(A, k, 'alg', 'mult')`) is equivalent to NNMFPACK's MLSA kernel. Table 2 shows the results obtained for three test examples when the number of iterations for all platforms is the same. As can be seen, Octave times are extremely high. MATLAB is outperformed by NNMFPACK's CPU implementation (CPUx16), even using the intrinsic MKL parallelism and being the computational cost per iteration of MULT (MLSA) lower than the β Div (see Sect. 2.1).

Next we show some precision results of the implementations. The tests used a picture in grey scale as the problem matrix A with dimensions $1,536 \times 2,304$ pixels. The matrix was NNMF-factored using MATLAB and NNMFPACK and reconstructed later. Table 3 shows the factorization error computed as

$$err_F = \frac{\|A - WH\|_F}{\sqrt{mn}}, \quad err_\beta = \frac{\sqrt{2D_\beta(A|WH)}}{\sqrt{mn}}. \quad (21)$$

These two expressions measure the error in terms of Frobenius norm (err_F) of $A - WH$ and in terms of β Div (err_β) of A and WH , respectively.

Table 3 Error in MATLAB’s NNMF and NNMFPACK by using the Frobenius norm error (err_F) and the β Div error (err_β)

| | | | k | | | | | |
|-------------|---------------|-----|---------|------|------|-------------|-------|-------|
| | | | err_F | | | err_β | | |
| | | | 154 | 307 | 768 | 154 | 307 | 768 |
| <i>iter</i> | Matlab NNMF | 100 | 20.6 | 19.2 | 17.6 | 20.6 | 19.2 | 17.6 |
| | NNMFPACK | | | | | | | |
| | $\beta = 0$ | 50 | 23.3 | 22.1 | 20.8 | 0.226 | 0.214 | 0.203 |
| | | 100 | 17.2 | 15.4 | 13.2 | 0.159 | 0.144 | 0.125 |
| | | 150 | 14.5 | 12.4 | 10.1 | 0.133 | 0.112 | 0.093 |
| | | 200 | 13.1 | 10.9 | 8.5 | 0.118 | 0.099 | 0.078 |
| | $\beta = 1$ | 50 | 25.3 | 24.0 | 23.2 | 2.42 | 2.30 | 2.23 |
| | | 100 | 18.4 | 16.8 | 15.0 | 1.75 | 1.60 | 1.44 |
| | | 150 | 14.9 | 13.3 | 11.3 | 1.41 | 1.26 | 1.08 |
| | | 200 | 13.3 | 11.3 | 9.3 | 1.26 | 1.07 | 0.88 |
| | $\beta = 1.5$ | 50 | 26.3 | 25.6 | 24.5 | 8.10 | 7.88 | 7.56 |
| | | 100 | 19.4 | 17.7 | 16.2 | 5.94 | 5.45 | 5.01 |
| | | 150 | 15.8 | 14.1 | 12.3 | 4.84 | 4.34 | 3.80 |
| | | 200 | 14.0 | 12.1 | 10.1 | 4.32 | 3.73 | 3.13 |
| | $\beta = 2$ | 50 | 28.2 | 27.2 | 26.4 | 28.2 | 27.2 | 26.4 |
| | | 100 | 20.5 | 19.1 | 17.6 | 20.5 | 19.1 | 17.6 |
| | | 150 | 17.0 | 15.3 | 13.6 | 17.0 | 15.3 | 13.6 |
| | | 200 | 14.9 | 13.2 | 11.3 | 14.9 | 13.2 | 11.3 |

For the MATLAB’s MULT NNMF function and for NNMFPACK using the β Div with $\beta = 0, 1, 1.5$ and 2 . MATLAB’s MULT NNMF is adjusted in the same way as for the previous comparison (Table 2) and always executes 100 iterations. β Div is executed with several iteration values ($iter = 50, 100, 150$ and 200), and several factorization inner dimension values, $k = 154, 307$ and 768 , where $k = \text{round}(\min(m, n)/d)$, with $d = 10, 5$ and 2 .

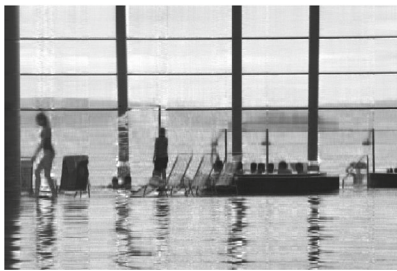
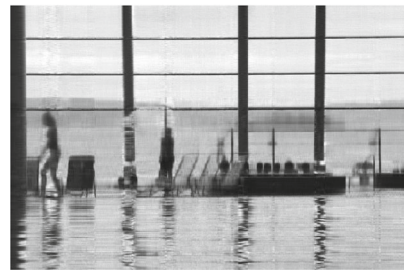
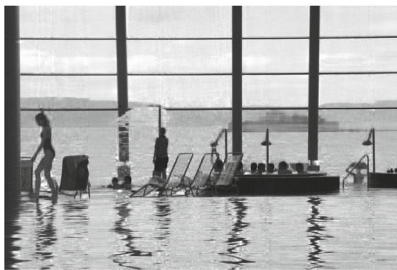
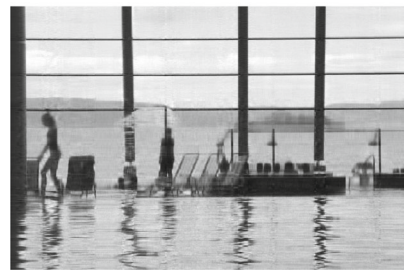
For the sake of completeness we offer results for the Frobenius norm error (err_F) and for β Div error (err_β). Obviously, measuring the error in terms of β Div error gives better results than in terms of Frobenius norm error, since the algorithm used is specific for minimizing the β Div of A and WH .

The results show that the error in NNMFPACK is lower than the MATLAB version when the number of iterations is higher than certain threshold independently of the inner dimension k . The difference in the error values between the MATLAB’s MULT NNMF and any NNMFPACK β -divergence variation is higher when the number of iterations increases, as expected.

Figure 2 shows the original picture (Fig. 2a) and the reconstructions from the factorization $A_{\text{reconstructed}} = WH \approx A_{\text{original}}$ using MATLAB (Fig. 2b) and NNMFPACK



(a) Original picture in gray scale

(b) MATLAB's MULT NMF with $k = 154$ (c) NNMF PACK, with $\beta = 0$, $k = 154$ and $iter = 100$ (d) NNMF PACK, with $\beta = 1.0$, $k = 154$ and $iter = 100$ (e) NNMF PACK, with $\beta = 1.5$, $k = 154$ and $iter = 100$ (f) NNMF PACK, with $\beta = 2.0$, $k = 154$ and $iter = 100$ (g) NNMF PACK, with $\beta = 0$, $k = 154$ and $iter = 200$ (h) NNMF PACK, with $\beta = 2.0$, $k = 154$ and $iter = 200$ **Fig. 2** Original picture and reconstructions using MATLAB's MULT NMF and NNMF PACK

(Fig. 2c–h), observing a better subjective reconstruction for NNMFPACK when β is lower.

The results for $\beta = 2$ using β Div with 100 iterations and MATLAB's MULT NNMF (100 iterations) are subjectively and objectively similar as shown in Fig. 2b, f and Table 3. This must be so because β Div with $\beta = 2$ is the same algorithm that NNMFPACK's MLSA and that MATLAB's MULT NNMF with the baseline adjustment made. Potential perceived differences come from the way of MATLAB and NNMFPACK initialize the matrices W and H . When the number of iterations doubles ($iter = 200$), a higher quality in the reconstruction is noticeable with better results for $\beta = 0$ (Fig. 2g) than for the other extreme, $\beta = 2$ (Fig. 2h).

Note that the optimal value of β in a concrete problem, represented by matrix A , is problem-dependent. For the case of image reconstruction, used in this paper, the β Div approach with $\beta = 0$ provides the best results since, as it can be in Table 3, the errors corresponding to $\beta = 0$ are lower than those corresponding to other values of β . This may be different for other type of problems represented for a different type of matrices, or even for random matrices.

4 Conclusions

We have presented an improved numerical library, NNMFPACK, that provides efficient algorithms to compute the NNMF. More specifically, different metrics (β -divergence) to assess the quality of the approximation of matrices are incorporated to NNMFPACK.

The design of algorithms presented in this work has been made according to the specifications of the NNMFPACK library, this is, guided by an efficiency target in current parallel computers. The features available make it an attractive alternative for the NNMF resolution in current multi-core and many-core architectures, providing some interesting performance figures from a computational point of view.

Although we have presented a simple case of image reconstruction application as a precision example, the library must be considered a generic tool which can be used in any field where the NNMF decomposition is required.

It is also worth noting that this work will lead to future versions extending current features such as more efficient algorithms and the treatment of sparse or structured matrices.

Acknowledgments This work has been partially supported by “Ministerio de Economía y Competitividad” from Spain, under the projects TEC2012-38142-C04-01 and TEC2012-38142-C04-04 and by ISIC/2012/006 and PROMETEO FASE II 2014/003 projects of Generalitat Valenciana.

References

1. Battenberg E, Freed A, Wessel D (2010) Advances in the parallelization of music and audio applications. In: Proceedings of the International Computer Music Conference, New York
2. Wnag J, Zhong W, Zhang J (2006) NNMF-based factorization techniques for high-accuracy privacy protection on non-negative-valued datasets. In: Proceedings of the Sixth IEEE International Conference on Computing and Processing, Data Mining Workshops ICDM Workshops, pp 513–517
3. Rodriguez-Serrano FJ, Carabias-Orti JJ, Vera-Candeas P, Virtanen T, Ruiz-Reyes N (2012) Multiple instrument mixtures source separation evaluation using instrument-dependent NMF models. In: Pro-

- ceedings of the 10th international conference on latent variable analysis and signal separation, March 12–15, Tel Aviv, Israel. LNCS, vol 7191. Springer, Berlin, pp 380–387
4. Xu W, Liu X, Gong Y (2003) Document clustering based on non-negative matrix factorization. In: Proceedings of the 26th annual international ACM SIGIR conference on research and development in information retrieval, July 28–Aug 1, Toronto, Canada, pp 267–273
 5. Berry MW, Browne M, Langville A, Pauca V, Plemmons R (2007) Algorithms and applications for approximate nonnegative matrix factorization. *Comput Stat Data Anal* 52:155–173
 6. Devajaran K (2008) Nonnegative matrix factorization: an analytical and interpretative tool in computational biology. *PLoS Comput Biol* 4(7):e1000029. doi:[10.1371/journal.pcbi.1000029](https://doi.org/10.1371/journal.pcbi.1000029)
 7. Lee DD, Seung HS (2001) Algorithms for non-negative matrix factorization. In: *Advances in neural information processing systems* MIT Press, Cambridge
 8. Kim J, Park H (2008) Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method. *SIAM J Matrix Anal Appl* 30:713–730
 9. Guan N, Tao D, Luo Z, Yuan B (2012) NeNMF: An optimal gradient method for non-negative matrix factorization. *IEEE Trans Signal Process* 60(6):2882–2898
 10. Cichocki A, Phan AH (2009) Fast local algorithms for large scale nonnegative matrix and tensor factorizations. In: Proceedings of IEICE transactions on fundamentals of electronics communications and computer sciences, E92-A, pp 708–721
 11. Cichocki A, Zdunek R, Amari SI (2007) Hierarchical ALS algorithms for nonnegative matrix and 3D tensor factorization. In: Proceedings of the 7th international conference on independent component analysis and signal separation, September 9–12, London, UK. LNCS, vol 4666. Springer, Berlin, pp 169–176
 12. Alonso P, García VM, Martínez-Zaldívar FJ, Salazar A, Vergara L, Vidal AM (2014) Parallel approach to NNMF on multicore architecture. *J Supercomput*. 70(2):564–576
 13. Díaz-Gracia N, Cocaña-Fernández A, Alonso-González M, Martínez-Zaldívar FJ, Cortina R, García-Mollá VM, Alonso P, Ranilla J, Vidal AM (2014) NNMFPACK: a versatile approach to an NNMF parallel library. In: Proceedings of the 2014 international conference on computational and mathematical methods in science and engineering, Cádiz, 2014, pp 456–465
 14. Carabias-Orti JJ, Rodríguez-Serrano FJ, Vera-Candeas P, Cañadas-Quesada FJ, Ruiz-Reyes N (2013) Constrained non-negative sparse coding using learnt instrument templates for real time music transcription. *Eng Appl AI* 26(7):1671–1680
 15. Carabias-Orti JJ, Virtanen T, Vera-Candeas P, Ruiz-Reyes N, Cañadas-Quesada FJ (2011) Musical instrument sound multi-excitation model for non-negative spectrogram factorization. *IEEE J Select Topics Signal Process* 5(6):1144–1158
 16. Minami M, Eguchi S (2002) Robust blind source separation by beta-divergence. *Neural Comput* 14:1859–1886
 17. Févotte C, Bertin N, Durrieu J-L (2009) Nonnegative matrix factorization with the Itakura-Saito divergence: with application to music analysis. *Neural Comput* 21:793–830
 18. Golub GH, Van Loan CF (1996) *Matrix Comput*. Johns Hopkins University Press, Baltimore
 19. <http://pirserver.edv.uniovi.es>