Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXVIII, págs. 217-226 (artículo completo)
Cristian García Bauza, Pablo Lotito, Lisandro Parente, Marcelo Vénere (Eds.)
Tandil, Argentina, 3-6 Noviembre 2009

SPARSE BEM IMPLEMENTATIONS USING AN APPLICATION FRAMEWORK FOR DISCRETE METHODS

Marco Dondero[a], Diego Santiago[b], Adrián P. Cisilino[a] and Santiago Urquiza[b]

[a]*División Soldadura y Fractomecánica, INTEMA-CONICET, Universidad Nacional de Mar del Plata, J. B. Justo 4302, 7600, Argentina, mdondero@fi.mdp.edu.ar, cisilino@fi.mdp.edu.ar*

[b]*Grupo de Ingeniería Asistida por Computadora, Facultad de Ingeniería, Universidad Nacional de Mar del Plata, CONICET, J. B. Justo 4302, 7600, Argentina, dsantiago@fi.mdp.edu.ar*

Keywords: Boundary Element Method, Application Framework, discrete methods, low-memory requirement.

Abstract. A practical strategy to increase the software reliability and to reduce programming effort is to implement the code using Application Frameworks (AF). A strategy for the implementation of a Boundary Element Method (BEM) solver within an AF for discrete methods is presented in this work. The rationale behind this approach is to reuse existing code for the implementation of the BEM solver. Thus, the effort put into the implementation of the BEM code is much less than that required for an ad-hoc development starting from scratch. Two strategies are introduced in order to reduce the memory requirements of the direct BEM . Both approaches consist in iterative procedures in which a number of coefficients of the fully-populated BEM system matrix are moved to the to the right hand side multiplied by the corresponding values of the unknowns in the previous iteration. In the R-approach the coefficients are selected using a criterion based on the distance between the collocation and field points, while in the N-approach consists in a condensation procedure for clusters of elements. A benchmark problem is used to verify and assess the developed code. Convergence and accuracy is studied for both memory reduction strategies while the performance of the implementations is assessed in terms of execution times and memory requirements. The ability of the proposed methodologies for reducing the memory requirements is demonstrated. The analysis of an example showed that the N-approach has a better convergence behavior than the R-approach.

INTRODUCTION

The solution of problems formulated in terms of partial differential equations by means of numerical methods is a topic of great interest in engineering and science. A practical strategy to increase the software reliability and to reduce programming effort is to implement the code using Application Frameworks (AF). An Application Framework is an abstraction in which common code providing generic functionality can be selectively overridden or specialized by a user in order to provide specific functionality. Frameworks are similar to software libraries in the sense that they consists of reusable abstractions of code wrapped in an Application Programming Interface (API); but unlike libraries, the overall control is not governed by a program, but by the framework itself.

It is presented in this work the implementation of a Boundary Element Method (BEM) solver within the AF "SolverGP". This AF has been developed in the Department of Mechanical Engineering of the University of Mar del Plata, Argentina. SolverGP is coded using FORTRAN and Object Oriented Programming tools. It has been successfully employed to implement FEM applications to solve a variety of problems in the fields of hemodynamics (Urquiza, 2006), friction stir welding (Santiago, 2004) and mold-filling using the resin transfer molding process (Santiago, 2007). The rationale behind using SolverGP to implement a BEM solver is to reuse the existing code. Thus, the effort put into the implementation of the BEM code is much less than that required for an ad-hoc development starting from scratch.

There are introduced two strategies to reduce the memory requirements of BEM. Both strategies are based on distinguishing the entries in the BEM matrix contributed by elements located far away and in the vicinity of the collocation point. As a result, a sparse BEM matrix is obtained and the system of equations is solved using an iterative solver. The developed code is verified and assessed solving a benchmark example for the Laplace equation which has analytical solution (the so-called Motz problem, Motz 1946). The performance of the algorithm is compared to that of a standard direct BEM solver and assessed in terms of computing time and memory requirements.

1 BEM IMPLEMENTATION USING THE APPLICATION FRAMEWORK

1.1 Direct BEM formulation

The BEM discrete formulation for a two-dimensional stationary potential problem discretized using N constant elements is

$$0 = g_{ij} q_j - f_{ij} u_j, \quad g_{ij} = \int_{\Delta S_j} G(\mathbf{x}_i, \mathbf{y}) \mathrm{d}S(\mathbf{y}), \quad f_{ij} = \int_{\Delta S_j} F(\mathbf{x}_i, \mathbf{y}) \mathrm{d}S(\mathbf{y}) + \tfrac{1}{2}\delta_{ij} \tag{1}$$

where $\boldsymbol{u}_j$ and $\boldsymbol{q}_j$ (j = 1,2,…,N) are the values of the potential and the flux in the element ΔS_j (see Figure 1); $G(\mathbf{x},\mathbf{y})$ and $F(\mathbf{x},\mathbf{y})$ are the well-known potential and the flux Kelvin fundamental solutions; and $\mathbf{x}$ and $\mathbf{y}$ stand for the collocation and the field points respectively. When the problem boundary conditions are replaced into the the equations in Eq. (1), it results a linear system of equations of the form $\mathbf{Ax}=\mathbf{b}$ where the matrix $\mathbf{A}$ is of dimension $N \times N$ and the vectors $\mathbf{x}$ and $\mathbf{b}$ are of length N. The vector $\mathbf{x}$ contains all the problem unknowns which are the nodal values of the fluxes, $\boldsymbol{q}_i$, on the boundary S_1 (the portion of the model boundary with prescribed potential, $\overline{u}_j$, see Figure 1) and the nodal values of the potentials, $\boldsymbol{u}_i$, on the boundary S_2 (the portion of the model boundary with prescribed flux, $\overline{q}_j$, see Figure 1). For

further details, the reader is referred to the book by Aliabadi and Wrobel (2002).

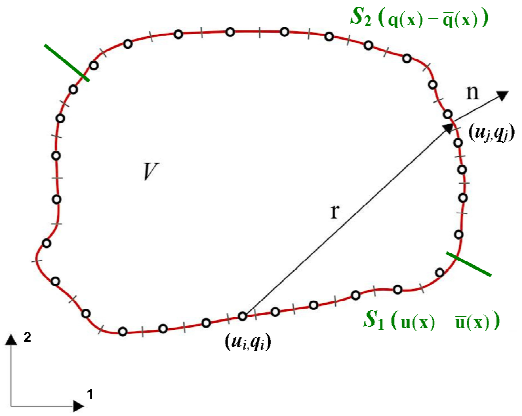

Figure 1: Boundary discretization using constants elements.

1.2 BEM implementation using the application framework

The assembly strategy in the SolverGP consists in using "*i-j* archetypal" assemble-elements (Urquiza, 2002). Each *i-j* archetypal assemble-element accounts for the contribution of a boundary element pair *i-j* (being *i* the element containing the collocation point and *j* the field element) to the BEM system of equations. Thus, for a problem discretized using *N* boundary elements there will be approximately $N^2/2$ archetypal assemble-elements. The elemental matrix for given *i-j* archetypal element is

$$\underbrace{\begin{pmatrix} 0 & 0 & f_{ij} & g_{ij} \\ 0 & 0 & f_{ij} & g_{ij} \\ f_{ji} & g_{ji} & 0 & 0 \\ f_{ji} & g_{ji} & 0 & 0 \end{pmatrix} \begin{pmatrix} u_i \\ q_i \\ u_j \\ q_j \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}}_{i \neq j} \quad \text{and} \quad \underbrace{\begin{pmatrix} f_{ii} & g_{ii} \\ f_{ii} & g_{ii} \end{pmatrix} \begin{pmatrix} u_i \\ q_i \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}}_{i=j} \tag{2}$$

The archetypal element embeds the information of the *i* and *j* boundary elements (element conectivities and node coordinates) so that the f_{ij} and g_{ij} entries in Eq. (2) can be evaluated. The elemental matrices contribute to the BEM system by assembling their contributions into the global matrix which is computed by performing loop over all the archetypal elements. The number of unknowns for the resulting system of equations is twice the number of equations. Then, the AF applies a reduction technique which eliminates the equation related to the known value of $\overline{u}_j$ or $\overline{q}_j$ at each node and leaves the equation for the q_j or u_j unknown accordingly. The process results in the system of equations **Ax=b** introduced in the previous section.

1.3 Sparse BEM strategies

This work proposes two alternative BEM assembly strategies which aim to reduce the memory requirements of the standard direct approach. Both approaches consist in iterative procedures in which a number of coefficients of the matrix **A** of the BEM system of equations **Ax=b** are moved to the to the right hand side (RHS), **b**, multiplied by the corresponding values of the unknown (or the value of the boundary conditions) evaluated in the previous

iteration (*it*). In this way the number of non-zero entries in the **A** matrix is reduced and consequently the associated memory requirements.

In the first approach (which will be referred in what follows as the R-approach) the **A**-matrix coefficients moved to the RHS are selected based on the distance $R_{ij} = |\mathbf{x}_i - \mathbf{x}_j|$ between the collocation and field points *i* and *j*, respectively. When the distance R_{ij} for the *i-j* archetypal element is minor than a given threshold value *R* ($R_{ij}<R$), the elemental matrix (see Eq. (2)) is assembled in the $\mathbf{A}^C$ matrix. In contrast, when the distance R_{ij} is greater than threshold value *R* ($R_{ij}>R$), the elemental matrix is assembled in the $\mathbf{A}^L$ matrix and moved to the RHS as follows:

$$\begin{aligned} &\mathbf{A}^C \{\mathbf{x}\}^{(it+1)} = -\mathbf{A}^L \{\mathbf{x}\}^{(it)} + \mathbf{b} \\ &\text{where} \quad A_{ij}^L = 0\,, \quad A_{ij}^C = A_{ij} \quad \text{if} \quad R_{ij} < R \\ &\qquad\qquad A_{ij}^L = A_{ij}\,,\; A_{ji}^C = 0 \quad \text{if} \quad R_{ij} > R. \end{aligned} \tag{3}$$

The second approach (from now on referred as the N-approach) incorporates an auxiliary matrix, $\mathbf{A}^G$, into the BEM system of equations. This new matrix accounts for the contribution of a number of clusters of BEM neighbor elements which influences are condensed in a sparse mode (see Figure 2). The BEM elements clusters are $\mathbf{G}^C = \{\mathrm{E}_1, \ldots, \mathrm{E}_r, \ldots, \mathrm{E}_n\}$, where E_r is the element which condenses the group influence, and '*n*' is the number of elements in the group. In other words, every element into the cluster $\mathbf{G}^C$ is considered to have the same value of *u* and *q*, which is assigned to the element E_r . Using this approach the global system of equations is:

$$\begin{aligned} &\left[\mathbf{A}^C + \mathbf{A}^G\right]\{\mathbf{x}\}^{(it+1)} = \left[-\mathbf{A}^L + \mathbf{A}^G\right]\{\mathbf{x}\}^{(it)} + \mathbf{b} \\ &\text{where} \quad A_{ij}^L = 0 \;,\; A_{ij}^C = A_{ij} \quad \text{if} \quad \mathrm{E}_i \wedge \mathrm{E}_j \in \mathbf{G}^C \\ &\qquad\qquad A_{ij}^L = A_{ij} \;,\; A_{ij}^C = 0 \quad \text{if} \quad \mathrm{E}_i \wedge \mathrm{E}_j \notin \mathbf{G}^C \\ &\qquad\qquad A_{ij}^G = \sum_{j=E_1}^{E_n} A_{ij} \quad \text{if} \quad \mathrm{E}_j = \mathrm{E}_r \quad \text{and} \quad A_{ij}^G = 0 \quad \text{if} \quad \mathrm{E}_j \neq \mathrm{E}_r \end{aligned} \tag{4}$$

The above strategies are implemented into the framework during the system matrix assembly. In this way the matrix **A** can be stored in a sparse format, resulting in a memory requirement reduction when compared to the dense matrix of the standard direct BEM. It is worth mentioning, that the above introduced strategies in contrast to other acceleration techniques (fast multipole, panel clustering, wavelets, etc.) keep the precision of the direct BEM (no extra approximations are done) but they solve the linear system of equations iteratively.

Figure 2 shows a schematic representation of the sparse **A** matrix on the left and the corresponding geometry discretized with 20 constant BEM elements on the right. Here the **A** matrix obtained with the N-approach is shown, in which $\mathbf{A}^C$ matrix, $\mathbf{A}^G$ matrix and $\mathbf{A}^L$ matrix are represented with different colors. The $\mathbf{A}^L$ matrix is given by $\mathbf{A}^L = \mathbf{A} - \mathbf{A}^C - \mathbf{A}^G$, i.e. the white blocks in Figure 2.

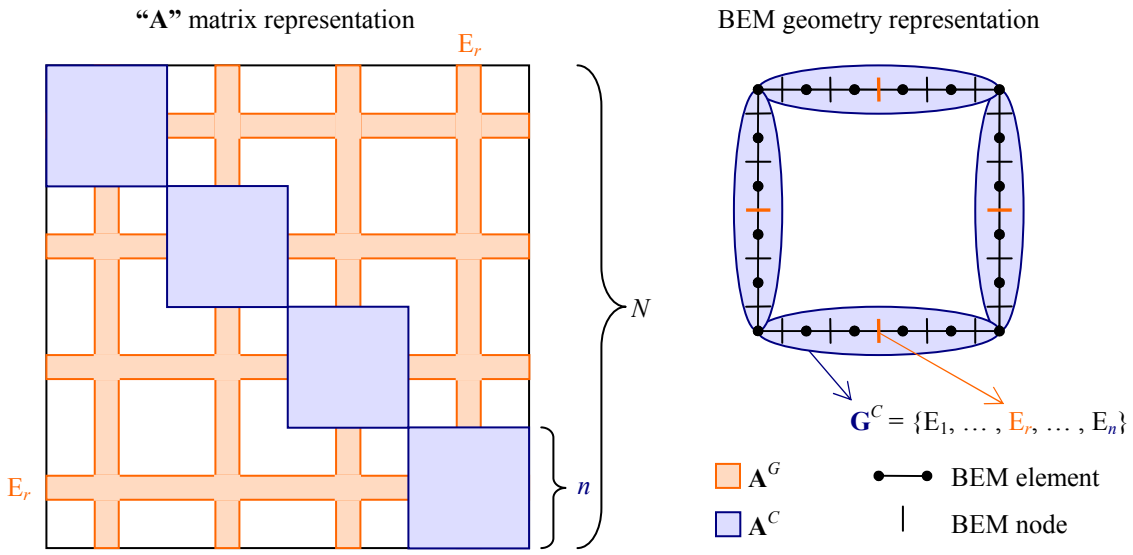

Figure 2: Matrix and geometry representations for the N-approach for N=20 and n=5.

2 NUMERICAL EXAMPLE AND VALIDATION

The Motz problem (Motz, 1946) is used to validate the proposed implementations. The problem geometry and boundary conditions are depicted in Figure 3. The problem presents the challenge of possessing a singularity at x=y=0, where the boundary conditions suddenly change from u=0 to $\partial u/\partial y$=0. An analytical solution to the problem is given in terms of the series expansion

$$u(r,\theta) = \sum_{j=1}^{\infty} \alpha_j r^{(2j-1)/2} \cos\left[\theta(2j-1)/2\right] \tag{5}$$

where the origin of the polar system is in the location of the singularity (see Figure 3). The coefficients α_j are tabulated in the literature (Georgiou, 1996).

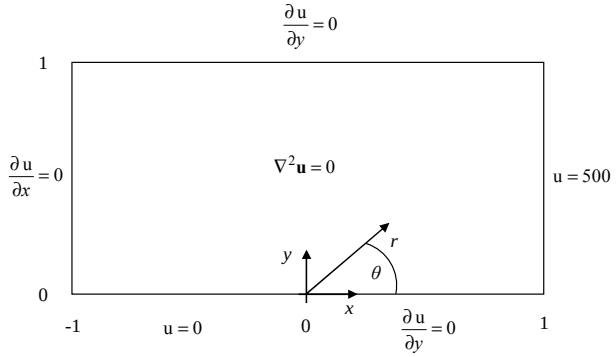

Figure 3: The Motz problem: geometry, governing equations and boundary conditions.

The performance of the standard direct BEM to solve the Motz problem was assessed for a model discretized using a 240-element discretization (case 1). Figure 4 depicts the computed solution which exhibits an error less than less that 0.5% relative to the analytical solution.

The same problem was solved using both strategies described in Section 1.3. For the R-

approach the threshold value R was chosen to range from 0.4 to 1, while for the N-approach the number of elements in the group was set to n=5, 10, 20 and 40. The BEM system of equations was solved using Gauss factorization for the case 1, including the direct BEM case (dense **A** matrix) and both sparse BEM strategies. Computer runs were performed in an Intel Core 2 Duo at 1.66 GHz PC with 2GB of RAM.

Results are reported in Table 1 and Figure 5. Table 1 reports the memory requirements and the solution status for the different analysis. Figure 5 shows a log-log plot of the evolution of the relative error with respect to the analytical solution as a function of the number of iterations.

It can be seen that the solution for the R-approach is convergent for $R \geq 0.5$ and the results are coincident with those of the direct implementation. In contrast, for values of R<0.5, the solution diverges (see Figure 5). For the N-approach, the convergence is achieved for all the values of n.

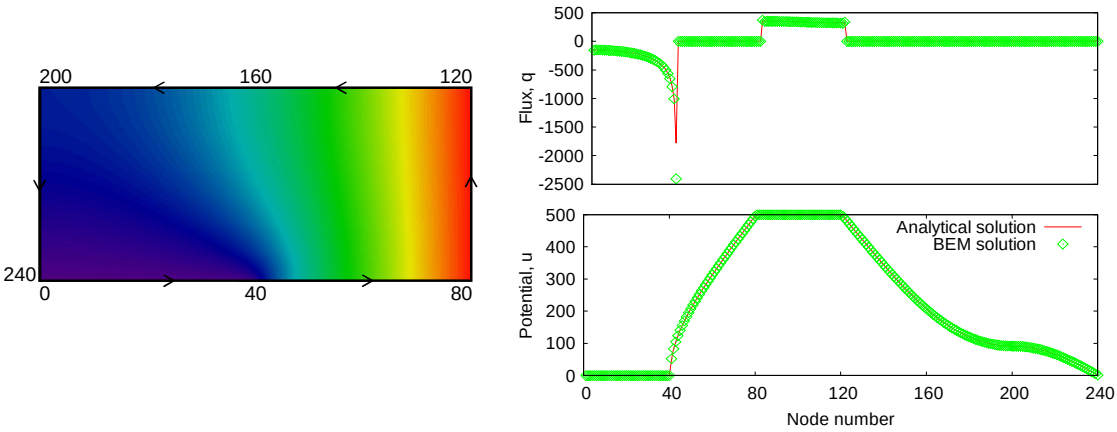

Figure 4: Left: Contour plot of potential field for case 1. Right: plots of the analytical and BEM solutions for the potential and flux fields versus node number along the boundary (see arrows and node numbers in the left figure).

R-approach			N-approach		
R	Memory (KB)	Convergence	n	Memory (KB)	Convergence
0.40	424	Not achieved	5	1151	Achieved
0.45	471	Not achieved	10	922	Achieved
0.50	519	Achieved	20	836	Achieved
0.70	711	Achieved	40	865	Achieved
1.00	1008	Achieved			
Direct BEM	2194	- -			

Table 1: BEM memory requirements and convergence for the case 1.

To further investigate the performance of the N-approach, the problem was discretized using larger number of elements: 2.400 elements (case 2) and 12.000 elements (case 3). In every case the system of equations was solved using the Jacobian pre-conditioned GMRES iterative solver provided by the SPARSEKIT, Saad (1994). Computer runs were performed in an AMD PhenomX4 at 2GHz PC, with 2GB of RAM.

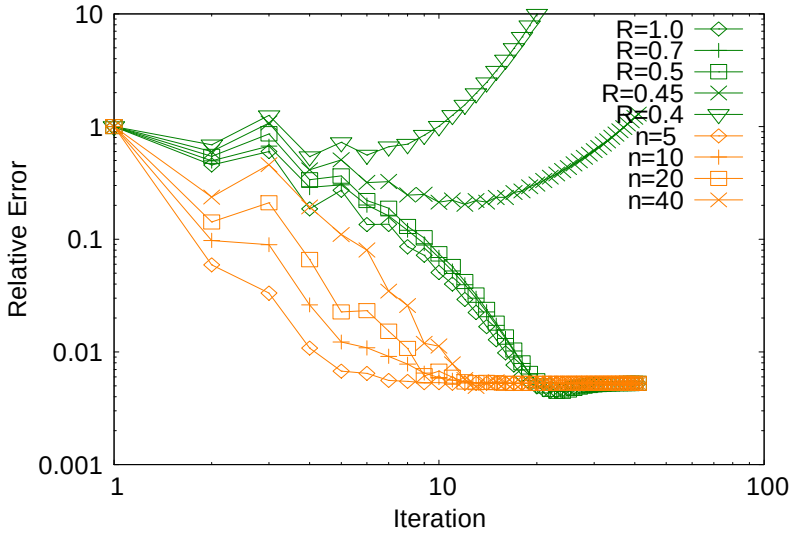

Figure 5: Log-log plot of the relative error vs. iteration number for both approaches (case 1).

The resulting memory requirements, execution times and relative error (up to the 10th iteration) with respect to the analytical solution for cases 2 and 3 are reported in Table 2. For the case 2, good agreement with the direct BEM solution for all n values was found using only 10 iterations. Memory reduction was important for all n values and for both cases. It can be inferred from the analysis of the results in Table 2 that the optimal value for the parameter n in the case 2 is in the range $50 < n < 100$ (with a memory reduction up to 93%), while in the case 3 the optimal value for n is around 125 (with a memory reduction up to 97%). It is worth to note that the memory requirement for the SolverGP is proportional to the parameter *Nzero* in Eq. (6). This difference is due to the auxiliary data structures used by SolverGP.

Case 2				Case 3			
N	Memory (MB)	Time (s) ([a])	Relative error (10^{-2}) ([b])	n	Memory (MB)	Time (s) ([a])	Relative error (10^{-2}) ([b])
5	77	11.5	0.098	50	233	124	0.093
10	42	6.9	0.11	80	173	100	0.16
20	24	4.9	0.14	100	158	93	0.21
50	15	3.3	0.33	125	149	87	0.27
80	15	3.1	0.64	250	167	79	0.58
100	15	3.1	0.89	400	221	78	1.11
200	23	3.3	1.04				
400	40	4.3	1.15				
Direct BEM	222	50	0.093	Direct BEM	5.500([c])		

([a]) Time per iteration, ([b]) Relative error after 10 iterations, ([c]) Insufficient RAM memory available.

Table 2: Memory requirements, execution time and relative error for different strategies.

The optimum value for n can be computed analytically. The number of non-zero coefficients (*Nzero*) in the matrix **A** is (see Figure 2)

$$Nzero = \underbrace{n^2 Nz}_{\text{\# of non-zeros in } \mathbf{A}C} + \underbrace{(Nz-1)\cdot Nz\cdot(2n-1)}_{\text{\# of non-zeros in } \mathbf{A}G} \tag{6}$$

where $Nz = N/n$, and N is the total number of BEM elements. Then, for a fixed value of N,

there exists an optimal value of n that minimizes the memory requirements, given by

$$\frac{dNzero}{dn} = 0 = \frac{N(n-1)(n^2+n-2N)}{n^3}. \tag{7}$$

The positive root of the quadratic equation in the right hand side of Eq. (7) is the number that will minimize the memory requirement. This is

$$n_{opt} = \frac{-1+\sqrt{1+8N}}{2}. \tag{8}$$

If N is maintained big enough so that $N >> 1$, then the optimum value of n can be approximated by

$$n_{opt} \approx \sqrt{2N}. \tag{9}$$

Thus, for case 2 the optimum value $n_{opt} \approx 69$, and for the case 3, $n_{opt} \approx 155$. These values for n_{opt} are in agreement with those observed in the memory requirements in Table 2. This is so because the memory required by SolverGP is proportional to the number non-zero coefficients in **A**, as mentioned before.

Besides, the memory requirement relative to a direct BEM for a given N can be computed using

$$\text{Relative Storage (\%)} = \frac{Nzero(n_{opt})}{N^2}\cdot 100 = \frac{1}{N\sqrt{2N}}\left(4N - \frac{3\sqrt{2N}}{2} - 1\right)\cdot 100. \tag{10}$$

Then, if $\sqrt{N} \gg 1$, then the Relative Storage can be approximated by

$$\text{Relative Storage (\%)} \approx \sqrt{\frac{8}{N}}\cdot 100. \tag{11}$$

For the case 2, the memory used by the solver, with n=80, is the 6.8% of the direct BEM, and the Relative Storage approximated by the Eq. (11) is 5.8%, showing that this measure is a good estimation of the memory saving. For the case 3, the Relative Storage approximation is 2.58%.

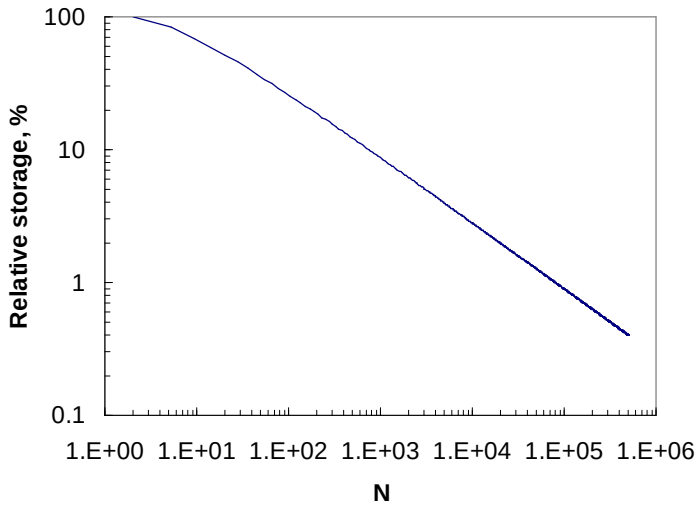

Figure 6: Memory requirements log-log plot for the N-approach.

The evolution of the Relative Storage with the number of elements, N, is shown in Figure 6. It can be seen that for problems with more than 10^5 unknowns a reduction of two or more orders of magnitude in the memory requirement is achieved. This improvement increases with N letting this strategy be a candidate to solve large-scale problems with BEM.

Figures Figure 7 and Figure 8 depict the relative error versus the number of iterations for the cases 2 and 3 respectively. The green lines correspond to the values of n closer to n_{opt}. In both cases and for all values of n studied, a stable convergence to the direct BEM solution is observed. It can also be seen that the lower the value of n, the greater the convergence speed, nevertheless, for values smaller than n_{opt} the memory requirement increases. In spite of the fact that the convergence speed is slower for values of n closer to n_{opt}, each iteration takes a shorter time, what results in the same relative error with respecto to the direct BEM for a similar total time.

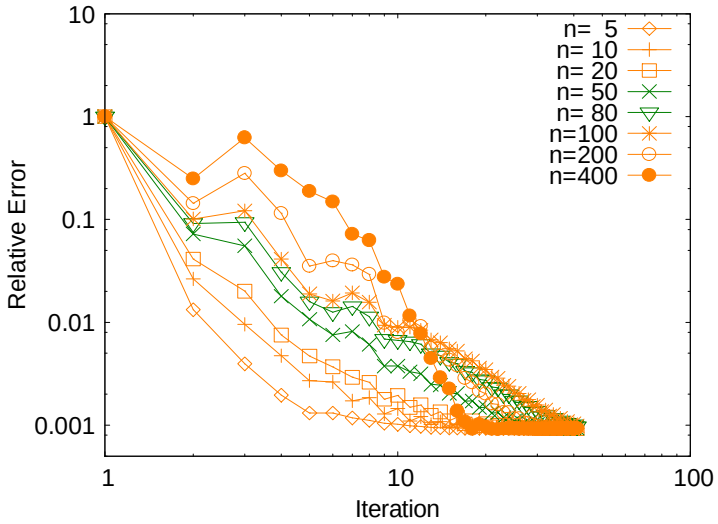

Figure 7: Log-log plot of the relative error vs. iteration number for case 2.

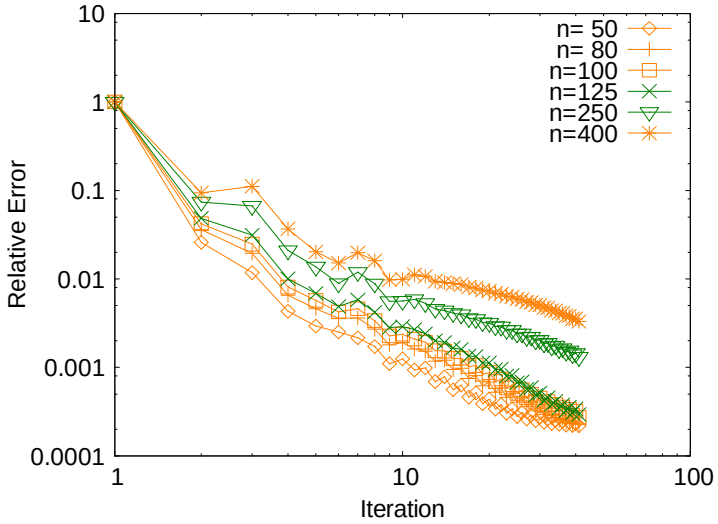

Figure 8: Log-log plot of the relative error vs. iteration number for case 3.

3 CONCLUSIONS

This work introduced an effective implementation of a BEM solver using the SolverGP Application Framework for discrete methods. In this way the effort needed for the implementation of the BEM solver is much less than that required for an ad-hoc development from scratch.

Two strategies are introduced in order to reduce the memory requirements of the direct BEM. Both approaches consist in iterative procedures in which a number of coefficients of the fully-populated BEM system matrix are moved to the to the right hand side multiplied by the corresponding values of the unknowns in the previous iteration. In the R-approach the coefficients are selected using a criterion based on the distance between the collocation and field points, while the N-approach consists in a condensation procedure for clusters of elements. The analysis of an example showed that the N-approach has a better convergence behavior than the R-approach.

For large problems (10^5 unknowns) the N-approach allows for reductions of two orders of magnitude in the memory requirement when compared to the standard direct BEM. This makes of it a good a candidate to solve large-scale problems. On the other hand, and although effective to reduce the memory requirements, the N-approach requires longer execution times than the standard direct BEM in order to achieve the same level of accuracy. However, these execution times are of the same order of direct BEM. Actual work is devoted to investigate different strategies to improve the convergence of the iterative solver and to further reduce the memory requirements. Further developments aim to extend the N-approach sparse BEM strategy to transient heat problems and three dimensions.

Finally it is worth noting that although the actual implementation is based on the classical direct BEM formulation, the Application Framework SolverGP possesses the potential for the implementation of accelerated BEM formulations such as Fast Multipole, Adaptive Cross Approximation and Hierarchical Matrices.

ACKNOWLEDGEMENTS

The authors acknowledge the support of the UNMdP, CONICET and grant PICT 1154 (2007) of the ANPyCT of the República Argentina.

REFERENCES

Aliabadi M.H, Wrobel L.C., 2002. “The Boundary Element Method”, Wiley, Chichister, UK.

Georgiou, G.C.: A singular function Boundary Integral Method for the Laplace equation. *Com. Num. Meth. In Eng*. 12, 127-134 (1996)

Motz, H.: The treatment of singularities in relaxation methods. *Q. Appl. Math*. 4, 371 (1946)

Saad, Y., SPARSEKIT: a basic tool kit for sparse matrix computation (version2). University of Illinois. http://www-users.cs.umn.edu/~saad/software/SPARSKIT/sparskit.html, 1994.

Santiago, D., Lombera, G., Cassanelli, A., Urquiza, S., de Vedia, L.: Numerical Modeling of Welded Joints by the “Friction Stir Welding” *Process. Mat. Res*. 7 (4), 569-574 (2004)

Santiago, D., Lombera, G., Urquiza, S.: Modelado numérico del proceso “Resin Transfer Moulding” (RTM). *Mecánica Computacional XXVI*, 931-937 (2007)

Urquiza, S.A., Venere, M.J.: An application framework architecture for FEM and other related solvers. *Mecánica Computacional XXI*, 3099-3109 (2002.)

Urquiza, S.A., Blanco, P., Venere, M.J., Feijoo, R.A.: Multidimensional Modelling for the Carotid Artery Blood Flow. *Comp. Meth. App. Mech. Eng.*, 195, 4002-4017 (2006)