Uniciencia
ISSN: 1011-0275
ISSN: 2215-3470
Universidad Nacional, Costa Rica

Guillén-Oviedo, Helen; Ramírez-Jiménez, Jeremías;
Segura-Ugalde, Esteban; Sequeira-Chavarría, Filánder

Description and implementation of an algebraic multigrid preconditioner for H1-conforming finite element schemes

Uniciencia, vol. 34, no. 2, 2020, July-December, pp. 55-81
Universidad Nacional, Costa Rica

DOI: https://doi.org/10.15359/ru.34-2.4

Available in: https://www.redalyc.org/articulo.oa?id=475966651004

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

UNICIENCIA Vol. 34, N° 2, pp. 55-81. Julio-Diciembre, 2020
URL: www.revistas.una.ac.cr/uniciencia
Correo electrónico: revistauniciencia@una.cr

Description and implementation of an algebraic multigrid preconditioner for H^1-conforming finite element schemes

Descripción e implementación de un precondicionador multinivel algebraico para esquemas de elementos finitos H^1-conformes

Descrição e implementação de um pré-condicionador multinível algébrico para esquemas de elementos finitos H^1-conformes

Helen Guillén-Oviedo
hellen.guillen.oviedo@una.ac.cr
Escuela de Matemática
Universidad Nacional,
Heredia, Costa Rica.
Orcid: http://orcid.org/0000-0002-5996-7772

Jeremías Ramírez-Jiménez
jeremias.ramirez.jimenez@una.ac.cr
Escuela de Matemática
Universidad Nacional,
Heredia, Costa Rica.
Orcid: http://orcid.org/0000-0001-6608-791X

Esteban Segura-Ugalde
esteban.seguraugalde@ucr.ac.cr
Escuela de Matemática
Centro de Investigación en Matemática Pura y Aplicada
Universidad de Costa Rica,
San José, Costa Rica.
Orcid: http://orcid.org/0000-0003-4743-954X

Filánder Sequeira-Chavarría
filander.sequeira@una.ac.cr
Escuela de Matemática
Universidad Nacional,
Heredia, Costa Rica.
Orcid: http://orcid.org/0000-0002-0593-3446

Received: ***20/nov/2019*** • Accepted: ***25/feb/2020*** • Published: ***31/jul/2020.***

Abstract

This paper presents detailed aspects regarding the implementation of the Finite Element Method (FEM) to solve a Poisson's equation with homogeneous boundary conditions. The aim of this paper is to clarify details of this implementation, such as the construction of algorithms, implementation of numerical experiments, and their results. For such purpose, the continuous problem is described, and a classical FEM approach is used to solve it. In addition, a multilevel technique is implemented for an efficient resolution of the corresponding linear system, describing and including some diagrams to explain the process and presenting the implementation codes in MATLAB®. Finally, codes are validated using several numerical experiments. Results show an adequate behavior of the preconditioner since the number of iterations of the PCG method does not increase, even when the mesh size is reduced.

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

Keywords: Finite element methods; H^1-conforming schemes; low-order approximations; multilevel techniques; computational implementation; MATLAB®

Resumen

En este artículo se presenta, en forma detallada, aspectos sobre la implementación del Método de Elementos Finitos (FEM, por sus siglas en inglés), para resolver una ecuación de Poisson con condiciones de frontera homogéneas. El objetivo de este trabajo es clarificar los detalles de esta implementación, tales como la construcción de los algoritmos, creación de experimentos numéricos y los resultados acerca de estos. Por ello, se describe el problema continuo y se muestra un enfoque FEM clásico para resolverlo. Después, se establece una técnica multiniveles para la resolución eficiente del sistema lineal correspondiente, que describe e incluye algunos diagramas para explicar el proceso y presenta los códigos de la implementación en MATLAB®. Finalmente, se realiza una validación de los códigos con varios experimentos numéricos. Los resultados muestran un comportamiento adecuado del precondicionador debido a que el número de iteraciones del método PCG no se incrementa, incluso cuando el tamaño de la malla se reduce.

Palabras clave: Métodos de elementos finitos; esquemas H^1 conformes; aproximaciones de bajo orden; técnicas multiniveles; implementación computacional; MATLAB®

Resumo

Este artigo apresenta, em detalhes, aspectos sobre a implementação do Método dos Elementos Finitos (MEF) para resolver uma equação de Poisson com condições de contorno homogêneas. O objetivo deste trabalho é esclarecer os detalhes dessa implementação, tais como a construção dos algoritmos, a criação de experimentos numéricos e os resultados sobre eles. Descreve-se, portanto, o problema contínuo e mostra-se uma abordagem clássica do MEF para resolvê-lo. Em seguida, estabelece-se uma técnica multinível para a resolução eficiente do sistema linear correspondente, que descreve e inclui alguns diagramas para explicar o processo e apresenta os códigos de implementação no MATLAB®. Finalmente, realiza-se uma validação dos códigos com várias experiências numéricas. Os resultados mostram um comportamento adequado do pré-condicionador devido ao número de iterações do método PCG não aumentar, mesmo quando o tamanho da malha é reduzido.

Palavras-chaves: Métodos de elementos finitos; esquemas H^1 compatíveis; aproximações de baixa ordem; técnicas multiníveis; implementação computacional; MATLAB®

Introduction

The main goal of this paper is to spread some computational aspects about the implementation of numerical methods for engineering, computer science and mathematics. The original contribution is not related to the mathematical analysis of the numerical methods for partial differential equations (PDE), but rather to explain the aspects about an efficient and clear computational implementation of a numerical method to approximate the solution of a PDE. We hope that the present work will be used for researchers and people interested in learning this kind of topics. It is important to note that there are papers and books which present in detail the methods that we discuss on this work. Nevertheless, there is not literature concerning to the

UNICIENCIA Vol. 34, N°. 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

explanation about the implementation of preconditioning techniques for linear systems associated to PDE.

In the first part of this article we discuss some general aspects related to the H^1-conforming Finite Element Method (FEM). The FEM approach is considered as one of the well-established and useful technique for the numerical solution of problems in different fields, described with the use of partial differential equations with different types of boundary conditions. One of the most important aspects related to the success of FEM corresponds to the fact that it is based largely on the basic finite element procedures used: the formulation of the problem in variational form, the discretization of this formulation and the effective solution of the resulting finite element equations. These features do not change from different types of problems (see, e.g., Ciarlet (2002) and Johnson (2009), for details).

In general, the discrete problem for FEM requires to solve a linear system of the form $\mathbf{Ax} = \mathbf{b}$. This kind of systems are usually solved by iterative methods, which approximate the exact solution of the system with a certain number of accurate digits. The corresponding selection of the iterative solver depends on the properties of matrix $\mathbf{A}$. For example, for symmetric and positive definite matrices the most popular method is the Conjugate Gradient (CG) method. However, in general, iterative solver depends on the condition number of the matrix of the system, which is related to the performance of the method. According to this, preconditioning techniques can be considered to accelerate convergence of the iterative methods. There are several types of preconditioning techniques, for example, one simple preconditioning technique of the system is to consider the equivalent system $\mathbf{M}^{-1}\mathbf{Ax} = \mathbf{M}^{-1}\mathbf{b}$ for some invertible matrix $\mathbf{M}$, which improve the conditioning of the system (see, e.g., Chen (2005)).

An important advance about preconditioning of large linear systems is given by multigrid techniques. The main idea in multigrid approaches is that any iterative solver works on a series of nested meshes, usually defined by a coarsening strategy, which depends on each mesh. Geometric grids are preferred for structured meshes, but algebraic multigrid (AMG) methods are better for unstructured meshes (see, e.g, Stüben (2001)). The AMG requires a simple relaxation process, usually Jacobi or Gauss-Seidel relaxation, and a suitable construction of the coarsening and interpolation operators. The goal of this kind of methods is to define the coarse level variables as a subset of the fine level variables. In other words, to split the fine level variables in two disjoint sets, one of them contains the coarse level variables, and the second one the remaining variables. In practice, the algorithms are usually based on heuristics. For example, for Beck's algorithm, the coarsening strategy are constructed with the information of the graph associated to the matrix of the system (see Beck (1999)).

Finally, this paper is organized as follows. First, we present a classical FEM scheme for a Poisson equation with homogeneous boundary conditions and describe some aspects about its computational implementation. Next, we present an explanation about preconditioning techniques for primal-FEM formulations. In particular, we focus in the algebraic multigrid preconditioner from Beck (1999), since it is an efficient and simple preconditioner for our variational formulation. However, we include more details related to the construction of the meshes and the coarsening operators. In this way,

UNICIENCIA Vol. 34, N°. 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

we describe more aspects than Beck (1999) in order to allow more people to understand the main aspects of the computational implementation of multigrid approaches. In next section, we perform some numerical experiments and discuss the results, to describe the advantages and disadvantages of the preconditioner. Here, we present a complete MATLAB® code. Finally, we provide some conclusions and further work related to the preconditioning techniques for FEM schemes.

H¹-conforming finite element method

In this section we present an example of a classical boundary value problem in order to justify the construction of the algebraic multigrid preconditioner. More precisely, given a bounded polygonal domain Ω in $\mathbb{R}^2$ with boundary Γ, we seek a scalar field u such that

$$-\Delta u = f \text{ in } \Omega \text{ and } u = 0 \text{ on } \Gamma \qquad (1)$$

where $f \in L^2(\Omega)$ is a volume force, and as usual $\Delta u := \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$. Now, given $v \in H_0^1(\Omega)$ from the Poisson equation note that

$$-\int_\Omega \Delta u v = \int_\Omega f v$$

where, employing the Green's formula in the left-hand side, we arrive at

$$\int_\Omega \nabla u \cdot \nabla v = \int_\Omega f v$$

for all $v \in H_0^1(\Omega)$. In other words, we obtain a variational formulation of problem (1), which reads: Find $u \in H_0^1(\Omega)$ such that

$$\int_\Omega \nabla u \cdot \nabla v = \int_\Omega f v \quad \forall\, v \in H_0^1(\Omega). \qquad (2)$$

It is well known that this problem has a unique solution, which continuously depends of the source f.

It is important to note from the new problem (2) that it only requires one order derivative and it establishes a function Hilbert space for the solution. On the other hand, the problem (1) does not allow us the explicit calculation of the solution u, because $H_0^1(\Omega)$ is an infinite dimensional space. A classical way of circumventing this drawback is the introduction of a finite dimensional subspace H_h of the space $H_0^1(\Omega)$ defined above consisting of piecewise polynomial continuous functions. More precisely, given $\mathcal{T}_h$ be a shape-regular triangulation of Ω without the presence of hanging nodes, and an integer $k \geq 1$, it follows that

$$H_h := \{v \in C^0(\Omega) / v|_T \in P_k(T)\ \forall\, T \in \mathcal{T}_h\}$$

where, we let $P_k(T)$ be the space of polynomials defined in T of total degree at most k. Furthermore, we define a discrete variational formulation (i.e. Galerkin scheme) which reads: Find $u_h \in H_h$ such that

$$\int_\Omega \nabla u_h \cdot \nabla v_h = \int_\Omega f v_h \quad \forall\, v_h \in H_h. \qquad (3)$$

It is easy to check that the problem (3) has a unique solution $u_h \in H_h$ (see, for example, Ciarlet (2002)), which in fact is an approximation of the continuous solution $u \in H_0^1(\Omega)$ of problem (2). Furthermore, there is a basis $\{\varphi_i^h\}_{i=1,\ldots,\dim H_h}$ of H_h, and then

$$u_h = \sum_{i=1}^{\dim H_h} \alpha_i \varphi_i^h$$

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

where, at least for $k = 1$ (low order), $\{\varphi_i^h\}_{i=1,\ldots,\dim H_h}$ is called a Lagrange-type basis, because it holds that $\alpha_k = u_h(x_k)$, for each vertex x_k of $\mathcal{T}_h$. In other words, the unknowns of the problem (3) correspond to the values of the solution at the vertices of the triangulation of Ω.

In general, the finite element method (FEM) for solving partial differential equations, is one of the high-order discretization schemes that has become a very active research area during the last decade. The main advantage of FEM approaches includes numerical solving of big partial differential systems through the analysis of small versions of the problem (on each element of $\mathcal{T}_h$). There are some variations of the FEM techniques, for example, the Galerkin scheme (3) is known as the classical finite element technique. On the other hand, there is a locally discontinuous version originally introduced in Cockburn (1998), and a virtual version presented in Beirão da Veiga (2013).

Computational implementation

A fact of the development of FEM methods lies in the difficulty of its computational implementation. Actually, there is few literature detailing the computational implementation of FEM methods, for example, one of the most famous contribution was presented in Carstensen (2002). In the present paper, we do not present details of the implementation of FEM schemes, but in order to explain a preconditioning technique below, we consider some general aspects. Indeed, let T be an element of $\mathcal{T}_h$ and define $\{\varphi_i^T\}_{i=1,\ldots,\frac{(k+1)(k+2)}{2}}$ as the Lagrange basis on T, which holds

$$dof_j^T(\varphi_i^T) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

where $\{dof_j^T(v)\}_{j=1,\ldots,\frac{(k+1)(k+2)}{2}}$ are the local degrees of freedom of v on T given by

1. value of v at the ith vertex of T, for all vertex i of T,
2. values of v at $k-1$ uniformly spaced points on e, for each edge e of T, for $k \geq 2$,
3. values of v at $\frac{k(k-1)}{2}$ uniformly spaced points in the interior of T, for $k \geq 3$.

It is not difficult to see that the local basis $\{\varphi_i^T\}_{i=1,\ldots,\frac{(k+1)(k+2)}{2}}$ is a restriction of $\{\varphi_i^h\}_{i=1,\ldots,\dim H_h}$ on T. Furthermore, the local discrete operators of T that contribute to the global linear system of problem (3) are given by

$$\mathbf{A}^T := \left[\int_T \nabla\varphi_i^T \cdot \nabla\varphi_j^T\right] \text{ and } \mathbf{b}^T = \left[\int_T f\varphi_i^T\right].$$

Using these discrete operators, we can assembly the global system of problem (3) following the algorithm:

1. Define $\mathbf{A} := \mathbf{0}$ and $\mathbf{b} := \mathbf{0}$
2. Define $m := \frac{(k+1)(k+2)}{2}$ and $m_0 := \frac{k(k-1)}{2}$
3. For each $T \in \mathcal{T}_h$ do
4. Set up $\mathbf{A}^T$ and $\mathbf{b}^T$
5. Define $p \in \mathrm{IR}^m$
6. For each vertex i of T
7. $p(i)$ = global index of i on $\mathcal{T}_h$
8. End For
9. For each edge e of T
10. For each j from 1 to $k-1$
11. $p = (3 + (k-1)(e-1) + j)$ (number of nodes of $\mathcal{T}_h$)
12. $+ (k-1)((\text{global index of } e \text{ on } \mathcal{T}_h) - 1) + j$
13. End For
14. End For

UNICIENCIA Vol. 34, N°. 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

15. For each from 1 to m_0
16. $p\,(3k+j) = (\text{number of nodes of } \mathcal{T}_h) + (k-1)\,(\text{number of edges of } \mathcal{T}_h) + j$
17. End For
18. $\mathbf{A}(p,p) = \mathbf{A}(p,p) + \mathbf{A}^T$
19. $\mathbf{b}(p) = \mathbf{b}(p) + \mathbf{b}^T$
20. End For

Finally, for the incorporation of boundary conditions, we modify the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ on the degrees of freedom corresponding to the boundary edges of $\mathcal{T}_h$. The main idea here, is that each row of the linear system associated to the boundary degrees of freedom becomes exactly the boundary condition of problem (1). The incorporation can be done as we described before. However, in order to define a graph-strategy for the construction of an algebraic multigrid for the global linear system, we actually eliminate the degrees of freedom corresponding to the boundary edges of $\mathcal{T}_h$, and then the main unknowns of the system are the values of the solution in the interior nodes of $\mathcal{T}_h$.

Preconditioning techniques

In practice, the linear system associated to problem (3) has a large number of unknowns, then, solving it by Gaussian elimination is quite slow, because this method has a complexity of $O(n^3)$ (see, for details, Saad (2003)). It is well-known (see, for example, Ciarlet (2002)) that $\mathbf{A}$ is a symmetric positive definite sparse matrix. Then, the Conjugate Gradient method (CG) is the most appropriate to approximate the system solution. However, for the mth iteration of CG holds that

$$\|\mathbf{x}-\mathbf{x}^{(m)}\|_{\mathbf{A}} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^m \|\mathbf{x}-\mathbf{x}^{(0)}\|_{\mathbf{A}} \quad (4)$$

where κ is the condition number of $\mathbf{A}$. It follows for $k >> 1$ that the term $\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$ behaves like $1-\frac{2}{\sqrt{\kappa}+1}$, which converges towards 1. Thus, inequality (4) predicts a slow convergence of the CG. On the other hand, the condition number of the matrix $\mathbf{A}$ of linear system (3) has an asymptotic behavior of $O(h^{-2})$, which establishes that $k >> 1$ when we use very fine triangulations.

According to the previous discussion, it is necessary to use techniques that reduce the condition number of $\mathbf{A}$. In general, these techniques construct a nonsingular matrix $\mathbf{M}$, which is called a *preconditioner*, such that the matrix $\mathbf{M}^{-1}\mathbf{A}$ of the equivalent system $\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}$ has a better condition number than the matrix $\mathbf{A}$ of the original system. Note that if $\mathbf{M} = \mathbf{A}$ the system is solved immediately. In the case of CG, which requires symmetry in the matrix, the preconditioned system possess the matrix $\mathbf{M}^{-1} \cdot \mathbf{A} \cdot \text{transpose}\,(\mathbf{M}^{-1})$, which is symmetric. The method of solving a linear system by CG using some preconditioner, is known as a Preconditioned Conjugate Gradient (PCG). Some of the most common preconditioners for symmetric systems are Jacobi, Gauss-Seidel, Cholesky and multigrid techniques (see, for example, Saad (2003)).

Multigrid techniques are the most attractive, because they reduce the size of the matrix of the linear system, by a sequence of nested meshes. Consequently, the number of iterations does not depend on the size of the matrix. More precisely, the matrix of the system is reduced by merging or deleting matrix entries. After the new matrix is generated, the corresponding new system is also

UNICIENCIA Vol. 34, N° 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

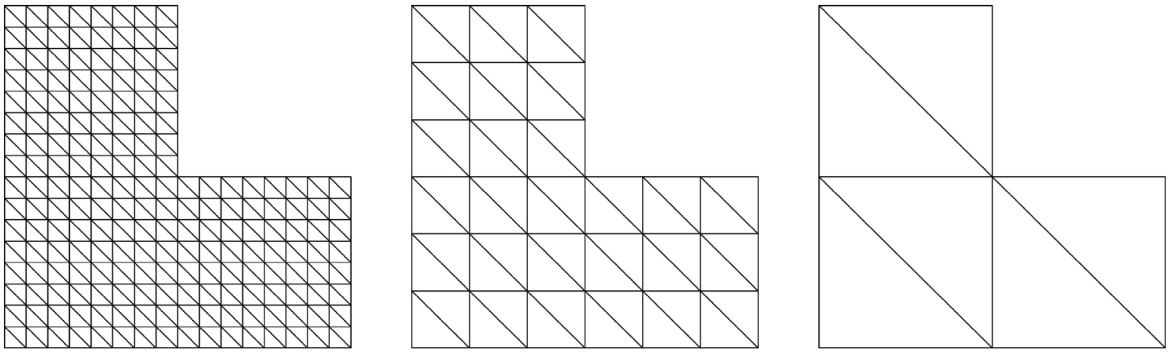

Figure 1. Sequence of nested meshes. *Note:* derived from research.

recursively reduced with the help of the next mesh. Thus, this process is continued and stops when all meshes are used. This sequence of matrices is known as preconditioner levels. For example, in Figure 1 we show a sequence of meshes that satisfy the nested condition. This multigrid preconditioner is known as a *geometric multigrid preconditioner* (see Trottenberg (2000)).

Once all the preconditioner levels have been generated, some steps of a relaxation method, such as Jacobi or Gauss-Seidel, are performed to solve the linear system. The approximation obtained is restricted to be used as the initial value of the auxiliary linear system given by the next lower level. In the same way this process is followed to the last level. Then, when the $\ell + 1$ system is solved, the solution is prolonged to level ℓ, where again some steps of a relaxation method are performed before moving on to the next level. This whole process is known as the *V-cycle*, which is illustrated in Figure 2.

Unfortunately, in practice this sequence of nested meshes is not available, only the finest mesh. For this reason, it is necessary to generate this sequence through the fine mesh by agglomerating cells. In Cartesian meshes this process is simpler than in unstructured meshes, where in most cases the agglomeration of cells is not possible. For example, in the case of a triangle mesh, the agglomeration of triangles does not always result in another triangle. An alternative to solve this problem is to create the next level, using the graph that represents the matrix of the system and not by the graph that represents the mesh. This process is known as *algebraic multilevel* and was introduced by Ruge and Stüben (1987).

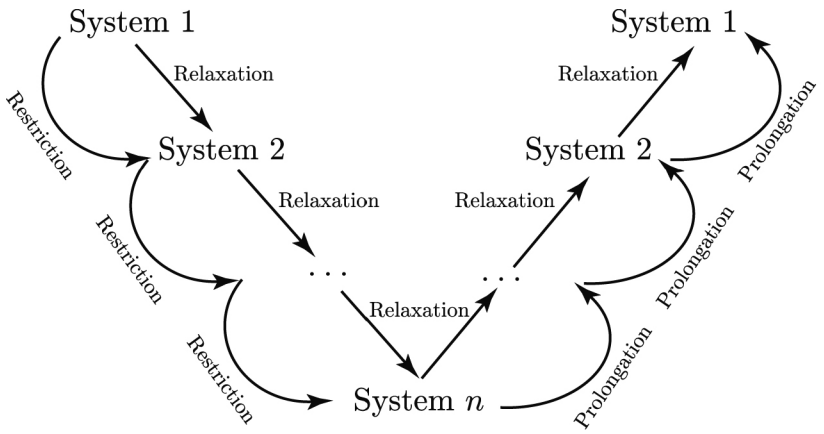

Figure 2. Scheme for a V-cycle. *Note:* derived from research.

UNICIENCIA Vol. 34, N° 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

Methodology employed

The research proposed in this work considers a scientific approach to a particular problem in the area of Applied Mathematics. It is done under an exploratory quantitative approach. Also, the methods presented are commonly used in problems related to finding approximations for solutions of boundary problems in partial differential equations.

Protocol

1. A particular algebraic multigrid preconditioner is described in detail for the model problem, introduced in the previous section. We consider the method introduced by Beck (1999), which is an interesting work, but it does not show a clear description, making it not accessible for beginning readers. Thus, we intend to explain in more detail some relevant elements of the preconditioner.
2. A proposal for computational implementation for the Beck's preconditioner (see Beck (1999)) is presented in MATLAB®. The code follows an object-oriented paradigm and, according to its organization, can even be used for the implementation of other multilevel preconditioners different than Beck's.
3. Once a code has been established in MATLAB® for the Beck's preconditioner, it is shown its behavior, in order to demonstrate its usefulness. Thus, two model examples of the Poisson problem are considered, whose solution is approximated by FEM scheme. To study the behavior of the preconditioner, only the results related to the resolution of the associated linear systems will be reported. Essentially, the number of iterations performed by the Conjugated Gradient method is shown, when it is used without a preconditioner and with the Beck's preconditioner. Additionally, the Incomplete Cholesky method will be considered to compare the behavior between preconditioners. For this purpose, we use meshes that do not exceed 800 000 degrees of freedom. Also, a maximum of 500 iterations and a tolerance relative to the first residue not exceeding 10^{-6}, since these parameters are sufficient to illustrate the desired behavior. On the other hand, it is important to consider one of the examples with a high number of condition to observe that the AMG method retains adequate behavior, which will allow to demonstrate the robustness of this preconditioner.
4. The results are tabulated and plotted to show the good behavior of Beck's preconditioner over the other methods considered. In particular, the execution time will not be reported because its measurement depends on various factors that are not necessarily controlled by the authors. This should not be construed as a disadvantage of the AMG method, but as a consequence of implementation in software such as MATLAB® and low performance computers.

Beck's algebraic multigrid preconditioner

In Ruge and Stüben (1987), Saad (1996), Beck (1999), Castillo and Sequeira (2013), the authors present several algebraic multigrid preconditioners. In particular, in this section we describe an algebraic multigrid technique proposed by Beck (1999), which take advantage of the characteristics of the finite element scheme (3). More

UNICIENCIA Vol. 34, N°. 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

precisely, the author identifies the geometric relations of the mesh in the sparse structure of the matrix.

The first algebraic multigrid preconditioner is presented in Ruge and Stüben (1987), which is based on partitioning the variables in each level of the multigrid in two sets by determining strong connection relations imposed by the structure of the matrix associated to the problem. The idea proposed in Beck's method is a simple coarsening strategy related to the Ruge-Stüben's method, which define a graph for the matrix using only the sparse structure and ignoring the strength of the connections, i.e. all connections are treated equally (the value of the entries of the matrix are not considered in each partition).

According to the previous discussion, we explain the algebraic multigrid approach using a general linear system:

$$\mathbf{Ax} = \mathbf{b}$$

where $\mathbf{A} \in \mathrm{IR}^{m\times m}$ and $\mathbf{b} \in \mathrm{IR}^{m}$. Now we define $\mathbf{A}_1 \coloneqq \mathbf{A}$ and $m_1 \coloneqq m$ as the data of the first level, where the new right-hand side will be explained later. Thus, the main idea of this method is to reduce $\mathbf{A}_1$ to a new matrix $\mathbf{A}_2 \in \mathrm{IR}^{m_2\times m_2}$ with m_2 considerably smaller than m_1. Then, this process is repeated in order to generate linear systems with associated matrices $\mathbf{A}_{\ell+1} \in \mathrm{IR}^{m_{\ell+1}\times m_{\ell+1}}$ such that $m_\ell > m_{\ell+1}$, for $\ell = 1, 2, \ldots, N-1$, where N is the maximun number of levels selected such that the last linear system can be solved efficiently.

To perform this procedure for each level $\ell = 1, 2, \ldots, N-1$ we construct matrices $\mathbf{R}_\ell \in \mathrm{IR}^{m_{\ell+1}\times m_\ell}$ and $\mathbf{P}_\ell \in \mathrm{IR}^{m_\ell\times m_{\ell+1}}$ in order to compute $\mathbf{A}_{\ell+1} \in \mathrm{IR}^{m_{\ell+1}\times m_{\ell+1}}$. The matrix $\mathbf{R}_\ell$ is known as the restriction operator, whereas the matrix $\mathbf{P}_\ell$ is the prolongation operator. Using these operators, we obtain the matrix $\mathbf{A}_\ell$ from the following algorithm, which define the matrices of each level of the algebraic multigrid preconditioner, summarized in Table 1.

1. For $\ell = 1$ until $N-1$ do
2. Construct $\mathbf{R}_\ell$ and $\mathbf{P}_\ell$, usually from $\mathbf{A}_\ell$
3. Define $\mathbf{A}_{\ell+1} = \mathbf{R}_\ell\mathbf{A}_\ell\mathbf{P}_\ell$
4. End For

Table 1

Summary of matrices in each level of Beck's algebraic multigrid preconditioner.

Level	Matrix	Restriction	Prolongation
$\ell = 1$	$\mathbf{A}_1$	$\mathbf{R}_1$	$\mathbf{P}_1$
$\ell = 2$	$\mathbf{A}_2$	$\mathbf{R}_2$	$\mathbf{P}_2$
⋮	⋮	⋮	⋮
$\ell = N-1$	$\mathbf{A}_{N-1}$	$\mathbf{R}_{N-1}$	$\mathbf{P}_{N-1}$
$\ell = N$	$\mathbf{A}_N$	-	-

Note: derived from research.

Finally, in order to describe the construction of each level of the preconditioner (i.e. $\mathbf{R}_\ell$ and $\mathbf{P}_\ell$), in the following section we analize a coarsening strategy for sparse matrices, which allow us to construct the restriction and prolongation operators. After that, we introduce the iterative solver based on the V-cycle approach.

Coarsening

We now explain the classical design of the coarsening technique briefly described in Section 3 of Beck (1999). In addition, we incorporate more explanations and examples, including a suggestion for a MATLAB® implementation.

The main idea of the coarsening algorithm is to select a favorably distributed subset of the matrix nodes (i.e. indexes of rows and columns), which are to become

UNICIENCIA Vol. 34, N°. 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

the nodes of the new matrix (coarse grid matrix). More precisely, the nodes for the new matrix are known as *master nodes*, whereas the remaining (which will be dropped) are known as *slave nodes*. In particular, this coarsening algorithm satisfies that the number of unknowns is reduced substantially in every coarsening step, and the coarse grid matrices preserve the sparsity pattern of the fine grid matrix, that is, coarse grid basis functions retain a local support and a restricted overlap with neighboring ones.

According to Section 3 in Beck (1999), there are three rules for the selection of master nodes. That is:

1. No master node may be directly connected to another master node.
2. There should be as many master nodes as possible.
3. The values of all master nodes are transferred with weight 1. The value for a slave node s is interpolated from the n_s master nodes it is connected to, where each master node contributes with weight $\frac{1}{n_s}$. This is used in the construction of the restriction operator below.

Here is important to remember that every Dirichlet node have been eliminated in order that the sparsity pattern of the matrix allows the separation of its nodes into two disjoint sets.

On the other hand, let $\mathbf{A} \in \mathrm{IR}^m$ be a symmetric sparse matrix. Now consider the graph that represents this matrix. More precisely, letting m nodes (numbering from 1 to m), we say that node i is connect to node j, with $i \neq j$, if entry $\mathbf{A}_{ij} \neq 0$. For example, consider matrix

$$\mathbf{A} := \begin{pmatrix} 9 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 9 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 9 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 9 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 9 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 9 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 9 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 9 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 9 \end{pmatrix} \quad (5)$$

which graph is given in Figure 3.

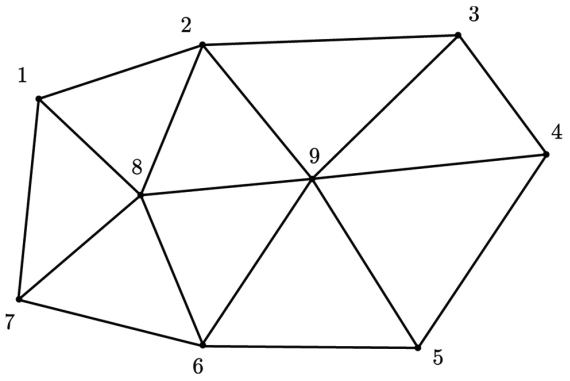

***Figure 3. Graph that represents the matrix* A**. *Note:* derived from research.

Note that the graph follows from the sparsity pattern of **A**. Now, we select one node to be our first master node. For example, we take node 1 and then, every neighboring node of 1 will be mark as slave node (i.e. node 2, 7 and 8). From the remaining nodes (3, 4, 5, 6 and 9) take the next master node. We take node 3, and then its neighbors (2, 4 and 9) are now slave nodes. Following this procedure, we take node 5 as master node and mark its neighbors as slaves. In Figure 4, we summarize the previous classification strategy.

UNICIENCIA Vol. 34, N°. 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

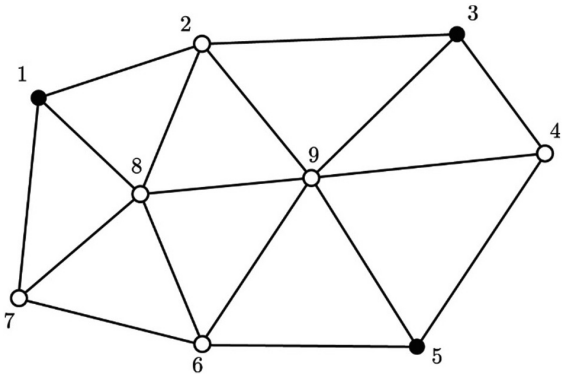

***Figure 4. Master (black) and slave (white) nodes for matrix* A**. *Note:* derived from research.

Observe that the coarsening process operates in a geometric way by sequentially choosing a master node and eliminating the corresponding neighboring nodes of the graph. Moreover, a primary criterion for selecting the next master node is to take the node with the minimum number of connections (taking into account the eliminations). And a secondary criterion is to follow the original numbering. We use the first choice by listing the nodes in ascending order, according to the number of non-zero elements of each row of **A**. We now present a MATLAB® implementation for previous coarsening strategy.

```
% ---------------------------------------------------------
% This function marks the master and slave nodes for a
% given matrix A of order m.
% ---------------------------------------------------------
%   marked: is a m-size vector which contains 0 on its ith
%           component if node i is slave. Otherwise, node
%         i is master, and the ith component of the vector
%           contains a new numbering for node i.
%   sizeM: is the number of master nodes.
% ---------------------------------------------------------
function [marked, sizeM] = coarsening(A)
    m = length(A); % Order of the matrix
    % -----------------------------------------------------
    nnzRow = zeros(1, m); % Number of non-zero elements of
                          % A by row
    for i = 1 : m
        nnzRow(i) = nnz( A(i,:) );
    end
    [~, idxRow] = sort(nnzRow); % Sort indices according to
                        % the number of non-zero elements
    % -----------------------------------------------------
    % First, every node is marked with a negative number,
    % in order to indicate that the node is unmarked.
    marked = -1 * ones(1, m);
    sizeM  = 0;
    % Using the symmetry of A, we follow the columns of A:
    for j = idxRow
        c = A(:,j);  % Define the jth column of A
```


UNICIENCIA Vol. 34, N°. 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

```
            I = find(c); % Find indices of non-zero elements
            if marked(j) == -1 % Unmarked node
                % Node j has not been marked yet. That's
                % why it will be marked as master and its
                % neighboring nodes will be slaves.
                % ----------------------------------------
                % Mark its neighbors as slaves:
                marked(I) = 0;
                % Mark as master:
                sizeM = sizeM + 1; % A master node was found
                marked(j) = sizeM; % New numbering for node j
            end
        end
    end
```

Using function `coarsening()` with the matrix defined in (5), we obtain the following results

```
marked =   1     0     2     0     3     0     0     0     0
sizeM =    3
```

which establish that the current node 1, 3 and 5 will be, respectively, the node 1, 2 and 3 of the matrix in the next level of the preconditioner.

Restriction and prolongation operators

After choosing master nodes, we can assemble the restriction matrix $\mathbf{R}_\ell$. Indeed, given $\mathbf{A}_\ell \in \mathrm{IR}^{m_\ell \times m_\ell}$ the matrix of level ℓ, let $m_{\ell+1}$ be the number of master nodes obtained by the previous coarsening algorithm, such that $m_{\ell+1} < m_\ell$. Then, letting n_s be the number of neighboring master nodes of slave node s, we define $\mathbf{R}_\ell \in \mathrm{IR}^{m_{\ell+1} \times m_\ell}$ as

$$(\mathbf{R}_\ell)_{ij} := \begin{cases} 1 & \text{if } j \text{ is } i\text{th master node} \\ \dfrac{1}{n_j} & \text{if } i \text{ is connected to } j \\ 0 & \text{otherwise} \end{cases}$$

Observe that the values of all master nodes are transferred with weight 1, whereas the values of all slave nodes are interpolated from its neighboring master nodes. This selection is strongly related to the definition of the degrees of freedom of scheme (3), at least for $k = 1$. For example, the restriction matrix for graph in Figure 4 is given by

$$\mathbf{R}_\ell = \begin{pmatrix} 1 & \frac{1}{2} & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & 1 & 1 & 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

On the other hand, Beck (1999) suggests a similar construction for prolongation operator $\mathbf{P}_\ell \in \mathrm{IR}^{m_\ell \times m_{\ell+1}}$. However, in order to preserve symmetry in the matrix of the next level, we define

$$\mathbf{P}_\ell := \mathrm{transpose}(\mathbf{R}_\ell)$$

UNICIENCIA Vol. 34, N° 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

Finally, using the restriction and prolongation operators we define $\mathbf{A}_{\ell+1} \coloneqq \mathbf{R}_\ell \mathbf{A}_\ell \mathbf{P}_\ell$ as the matrix of next level. Then, we continue the process employing $\mathbf{A}_{\ell+1}$ to construct $\mathbf{R}_{\ell+1}$ and $\mathbf{P}_{\ell+1}$. The stopping criterion corresponds to a maximum order for the last matrix. In addition, the last matrix presents a considerably reduction in the order of the original matrix, then direct methods (for example LU factorization) can be used for solver effects.

According to the above discussion, we now propose the following computational implementation for the construction of all the matrices required by the algebraic multigrid preconditioner.

```
% ---------------------------------------------------------
% This function constructs the levels of the algebraic
% multigrid preconditioner.
% ---------------------------------------------------------
%   A: is a mxm symmetric positive definite sparse matrix.
%   maxSize: is the maximum size of the matrix in the last
%            level.
%   MA: is a list of matrices for coarse matrices
%   MR: is a list of matrices for restriction operators
%   MP: is a list of matrices for prolongation operators
%   L: is the lower triangular matrix of the LU factorization
%      of the last matrix A
%   U: is the upper triangular matrix of the LU factorization
%      of the last matrix A
%   steps: is a list of the Smoothing steps for Gauss-Seidel
% ---------------------------------------------------------
function [MA, MR, MP, L, U, steps] = createLevels(A, maxSize)
    MA = {}; % List of matrices
    MR = {}; % List of restriction operators
    MP = {}; % List of prolongation operators
    % ----------------- Create the levels -----------------
    numLevels = 0; % Number of levels
    m = length(A); % Order of the matrix
    while m >= maxSize
        % Mark nodes as Master and Slave
        [marked, sizeM] = coarsening(A);
        % Construct Restriction matrix
        R = sparse(sizeM, m);
        for j = 1 : m
            if marked(j) == 0
                % -------------- SLAVE NODE --------------
                c = A(:,j); % Define the jth column of A
                I = find(c); % Find indices of non-zero
```

UNICIENCIA Vol. 34, N° 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

```
                                % elements of the jth column of A
                    [~, ~, I] = find( marked(I) ); % Find indices
                                                   % of master nodes
                                                    % arround node j
                  nnz = length(I);   % Number of neighboring master
                                     % nodes of node j
                    R(I, j) = 1 / nnz;
                else
                    % -------------- MASTER NODE --------------
                    R(marked(j), j) = 1;
                end
            end
            % Store matrices on lists
            numLevels = numLevels + 1;
            MR{numLevels} = R;
            MA{numLevels} = A;
            MP{numLevels} = R';
            % Preparations for next level
            A = R * A * R';
            m = sizeM;
        end
        % ---- Compute LU Factorization for the last matrix ---
        [L,U] = lu(A);
        % ---------- Create list of smoothing steps ----------
        steps = zeros(numLevels, 1);
        for i = 1 : numLevels
            steps(i) = i + 1;
        end
    end
```

The vector steps, defined by function `createLevels()`, contains the number of relaxation steps for the V-cycles of the preconditioner. For simplicity, we perform $\ell + 1$ relaxation steps for level ℓ.

V-cycles iteration

According to the previous sections, we now aim to present an iterative solver for linear system $\mathbf{Ax} = \mathbf{b}$ based on the *iterative refinement method* (see, e.g, Wilkinson (1994) or Burden, Faires and Burden (2015)). Indeed, given, $\mathbf{A} \in \mathrm{IR}^{n\times n}$, and $\mathbf{b} \in \mathrm{IR}^n$ and $\mathbf{x}_0 \in \mathrm{IR}^n$ an initial guess, we consider the following algorithm

1. Define $\mathbf{r}_1 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
2. For $k = 1$ until number of V-cycles do
3. For $\ell = 1$ until $N - 1$ do
4. Define $\mathbf{x}_\ell = \mathbf{0} \in \mathrm{IR}^{m_\ell}$
5. Perform some steps of a relaxation method to the system $\mathbf{A}_\ell \mathbf{x}_\ell = \mathbf{r}_\ell$
6. Define $\mathbf{r}_{\ell+1} = \mathbf{R}_\ell(\mathbf{r}_\ell - \mathbf{A}_\ell \mathbf{x}_\ell)$
7. End For
8. Solve $\mathbf{A}_N \mathbf{x}_N = \mathbf{r}_N$ by a direct method

UNICIENCIA Vol. 34, N°. 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

9. For $\ell = N - 1$ until 1 do
10. Define $\mathbf{x}_\ell = \mathbf{x}_\ell + \mathbf{P}_\ell \mathbf{x}_{\ell+1}$
11. Perform some steps of a relaxation method to the system $\mathbf{A}_\ell \mathbf{x}_\ell = \mathbf{r}_\ell$
12. End For
13. End For

Here N is the total number of levels defined by `createLevels()` in the previous section. Notice that, after the calculation of the first residual $\mathbf{r}_1$ and according to the iterative refinement method, we need to solve alternative linear system $\mathbf{Ay} = \mathbf{r}_1$ in order to make the correction $\mathbf{x}_0 = \mathbf{x}_0 + \mathbf{y}$. Thus, for solving $\mathbf{Ay} = \mathbf{r}_1$ we perform some steps of a relaxation method, such as the Jacobi iteration or Gauss-Siedel iteration. Next, we restrict the new residual to the following level. We continue this process until level $N - 1$. In last level, we compute the solution of the corresponding system employing a direct solver. In particular, for simplicity we use LU Factorization, but Cholesky Factorization is also a good choice. Finally, through each level, the solution is prolonged and updated until a new approximation to the original system is obtained.

It is important to observe that the original residual $\mathbf{r}_1$ is not changed. In addition, in order to preverse symmetry (see, e.g., Chen (2005)), we utilize Forward Gauss-Seidel iteration when each residual is restricted. On the other hand, we use Backward Gauss-Seidel iteration when each residual is prolonged. We end this section by illustrating the definition of every V-cycle in Figure 5. The MATLAB® code for this solver will be presented in function `apply()` in a section below.

Preconditioned Conjugate Gradient Method

The V-cycle iteration is a solver for the linear system $\mathbf{Ax} = \mathbf{b}$. However, this is used as a preconditioner for the usual preconditioned conjugate gradient (PCG) method. More precisely, consider the PCG method (see, e.g., Saad (2003)) given by

1. Define $\mathbf{r} = \mathbf{b} - \mathbf{Ax}_0$
2. Apply preconditioner to solve $\mathbf{Az} = \mathbf{r}$
3. Define $\mathbf{p} = \mathbf{z}$
4. Define $\mathbf{x} = \mathbf{x}_0$
5. For $k = 0$ until convergence do
6. Define $\mathbf{v} = \mathbf{Ap}$
7. Define $w = \mathbf{r}^t\mathbf{z}$
8. Define $\alpha = \frac{w}{\mathbf{v}^t\mathbf{p}}$
9. Update $\mathbf{x} = \mathbf{x} + \alpha\mathbf{p}$
10. Update $\mathbf{r} = \mathbf{r} - \alpha\mathbf{v}$
11. Apply preconditioner to solve $\mathbf{Az} = \mathbf{r}$
12. Define $\beta = \frac{\mathbf{r}^t\mathbf{z}}{w}$
13. Update $\mathbf{p} = \mathbf{z} + \beta\mathbf{p}$
14. End For

Next, note from steps 2 and 11 that it is necessary to solve linear systems for every preconditioner. In this case, we will use the iterative solver defined in previous section.

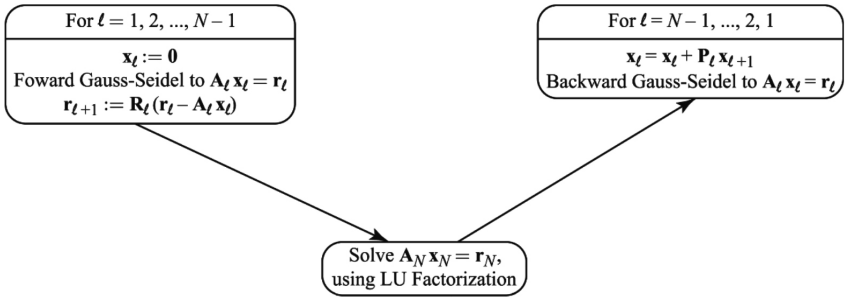

Figure 5. Definition of each V-cycle of the algebraic multigrid preconditioner. Note: derived from research.

UNICIENCIA Vol. 34, N° 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

Complete MATLAB® code

In this section, we propose a simplified computational implementation for the algebraic multigrid preconditioner introduced by Beck (1999). All procedures described in the foregoing sections are unified in a MATLAB® class called `amgBR`, which is presents at end of this paper (see Annex 1).

On the other hand, consider the following implementation for PCG method:

```
% -------------------------------------------------------------
% Preconditioned conjugate gradient method.
% -------------------------------------------------------------
%   P: is a preconditioner
%   A: is the matrix of the linear system
%   b: is the right-hand side of the linear system
%   x: is an initial guess
%   tol: tolerance
%   iterMax: is the maximum number of iterations
%   iterUsed: is the number of iterations used
% -------------------------------------------------------------
function [x, iterUsed] = pcg(P, A, b, x, tol, iterMax)
    r = b - A*x;
    z = P.apply(r);
    p = z;
    nrm0Tol = tol * norm(r);
    iterUsed = 0;
    while iterUsed < iterMax && norm(r) >= nrm0Tol
        v = A * p;
        w = r'*z;
        alpha = w / (v'*p);
        x = x + alpha * p;
        r = r - alpha * v;
        z = P.apply(r);
        beta = (r'*z) / w;
        p = z + beta * p;
        iterUsed = iterUsed + 1;
    end
end
```

Then, the class `amgRB`, defined at the beginning of this section, can be used as in the following example:

```
P = amgRB(A, maxSize, numVCycles); % Define preconditioner
x0 = zeros(length(A),1); % Define initial guess
[x, k] = pcg(P, A, b, x0, 1.0e-6, 1000)
where A, b, maxSize, and numVCycles must be previously defined.
```

UNICIENCIA Vol. 34, N°. 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

Numerical experiments

In this section, we carry out two numerical experiments to validate our code and illustrate the behavior of the algebraic multigrid preconditioner proposed in Beck (1999) at least for $k = 1$. For all the examples, triangular meshes were generated using the software TRIANGLE developed by Shewchuk (1996). In addition, we modify the continuous problem (1) to

$$-\nabla(\mathbf{K}\nabla u) = f \text{ in } \Omega \text{ and } u = 0 \text{ on } \Gamma$$

where $\mathbf{K} \in [L^{\infty}(\Omega)]^{2\times 2}$ is a uniformly positive definite tensor describing the material properties of Ω. The idea of introducing the tensor $\mathbf{K}$ is to consider a problem with high condition number (see Example 2 below).

The numerical results were obtained using the MATLAB® code introduced in previous section. The corresponding linear systems are solved by the Conjugate-Gradient method with a tolerance of 10^{-6}, 500 as the maximum number of iterations, and taking as initial guess the vector which each entry is equal to one. Here, we use *ndofs* for the number of unknown of each linear system, that is, the value of m_1. In addition, we introduce the integer $\mu > 0$ in order to define the number of smoothing steps used by Gauss-Seidel methods. More precisely, for a level $\ell = 1,2,\ldots,N-1$, the number of relaxation steps are given by $\mu + \ell - 1$. Moreover, we only use one V-Cycle for the AMG preconditioner. Finally, we also consider the classical Incomplete Cholesky preconditioner (see, e.g, Saad (2003)) as an alternative solver.

In Example 1, we consider $\Omega := (0,1)^2$, $\mathbf{K}$ the identity matrix, and choose the data ℓ so that the exact solution is given by

$$u(x,y) := \sin(\pi x)\sin(\pi y)$$

for all $(x,y) \in \Omega$. In Figure 6, we show a triangular decomposition for our domain obtained with TRIANGLE, where it is important to note that only unstructured meshes are used.

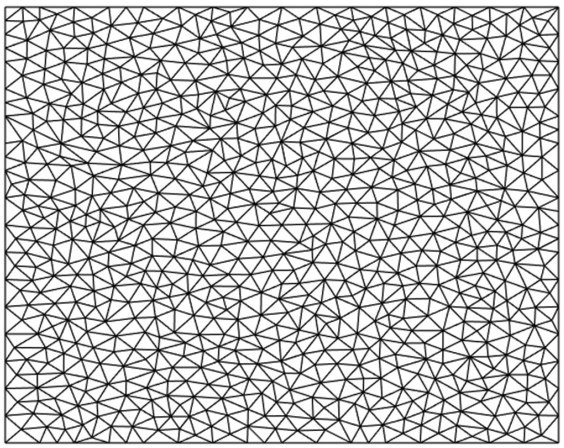

Figure 6. Mesh for Example 1 with 1531 triangles. *Note:* derived from research.

In Table 2, the results of the AMG preconditioner studied in this work are presented and compared to the classical CG method. We also obtain results for the Incomplete Cholesky method. The AMG method shows an improvement with respect to the classical methods, in terms of the number of iterations and the obtained residuals. For instance, note that the CG method almost never converges after 500 iterations for any of the experiments, while the Incomplete Cholesky method, although converging, the number of iterations is significantly increasing compare to AMG. In fact, the most important aspect to observe is that the AMG method always converges, and its number of iterations does not exceed 22 (considering all experiments), which allow us to see the robustness of the preconditioner. In addition, these iterations decrease as the parameter μ increases, which establishes that several relaxation steps carry

UNICIENCIA Vol. 34, N°. 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

Table 2

Example 1: Number of iterations and last residual.

ndofs	CG		Incomplete Cholesky		AMG		
	Iter.	Residual	Iter.	Residual	μ	Iter.	Residual
30802	418	5.3970e-08	52	3.5578e-05	2	15	3.6363e-08
					4	14	2.6465e-08
					8	13	1.4924e-08
258153	500	1.0513e-04	146	6.3646e-05	2	21	9.4378e-09
					4	19	1.5173e-08
					8	18	8.6993e-09
516513	500	1.2038e-04	199	8.1081e-05	2	22	6.4887e-09
					4	20	1.1890e-08
					8	19	8.0017e-09
782939	500	9.8561e-02	228	9.8735e-05	2	18	6.0660e-05
					4	17	4.5690e-05
					8	15	6.9232e-05

Note: derived from research.

out fewer iterations, but will require more execution time, which is not shown in this paper because the measurement of time depends on multiple reasons related to particular situations.

On the other hand, for the last mesh of 782939 degrees of freedom, we show in Table 3 a description of each level created by the AMG preconditioner. There, it is possible to see a significant reduction in the unknowns of the system. Finally, in Figure 7, we show that the residuals of the AMG method decreases faster.

Table 3

Example 1: Description of the levels of the AMG preconditioner for the mesh of 782939 degrees of freedom.

Level	Order of the matrix	Number of non-zero entries
1	782939	5472891
2	207841	2001147
3	42911	478627
4	7873	94105
5	1364	16372
6	237	6010

Note: derived from research.

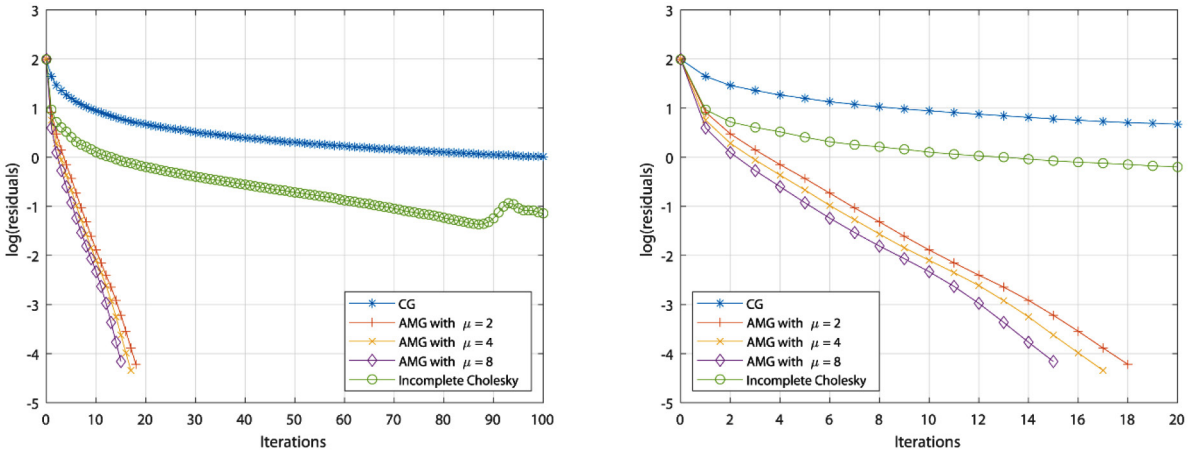

Figure 7. Example 1: Residual history for 100 iterations (left) and 20 iterations (right).
Note: derived from research.

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

In Example 2, we consider the model problem on a domain with two different materials and an isotropic diffusion. More precisely, Ω consists of two squares, the first one is the unit square $(0,1)^2$, which has diffusion $\mathbf{K}$ equal to the identity matrix $\mathbf{I}_2$, and within this square, a second square $\left(\frac{1}{3},\frac{2}{3}\right)^2$ with diffusion $\mathbf{K} = 10^{-10}\mathbf{I}_2$. An example of the meshes for this domain is shown in Figure 8. Here the source function ℓ is equal to zero.

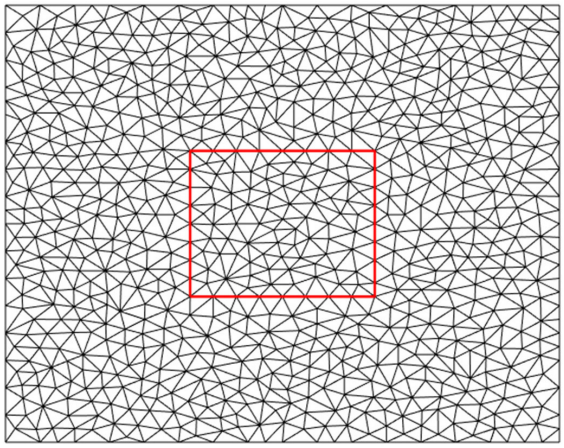

Figure 8. Mesh for Example 2 with 1543 triangles. *Note:* derived from research.

In Table 3, we note that the AMG method performs a behavior similar to the obtained in Example 1, although the condition number of the system matrix in Example 2 is 10^9 greater than the condition number of Example 1. Thus, the AMG preconditioner does not show a significant change by increasing the conditioning of the matrix. Finally, in Table 4, a level description is presented, whereas in Figure 9, we show the residuals history, all this for the last mesh used in Table 4.

Table 5

Example 2: Description of the levels of the AMG preconditioner for the mesh of 781506 degrees of freedom.

Level	Order of the matrix	Number of non-zero entries
1	781506	5458246
2	208021	2003307
3	42978	477048
4	7909	93343
5	1404	16444
6	250	4927

Note: derived from research.

Table 4

Example 2: Number of iterations and last residual.

ndofs	CG		Incomplete Cholesky		AMG		
	Iter.	Residual	Iter.	Residual	μ	Iter.	Residual
30649	255	4.3738e-05	42	3.4858e-05	2	11	2.2661e-05
					4	10	1.8617e-05
					8	9	1.4392e-05
257621	500	4.2431e-04	105	7.4950e-05	2	15	3.6332e-05
					4	14	2.8257e-05
					8	12	5.5559e-05
515705	500	1.1529e-02	145	8.6156e-05	2	15	8.4997e-05
					4	14	9.1141e-05
					8	13	7.9850e-05
781506	500	7.3480e-02	174	1.0637e-04	2	17	8.6464e-05
					4	16	5.9244e-05
					8	14	8.7667e-05

Note: derived from research.

UNICIENCIA Vol. 34, N° 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

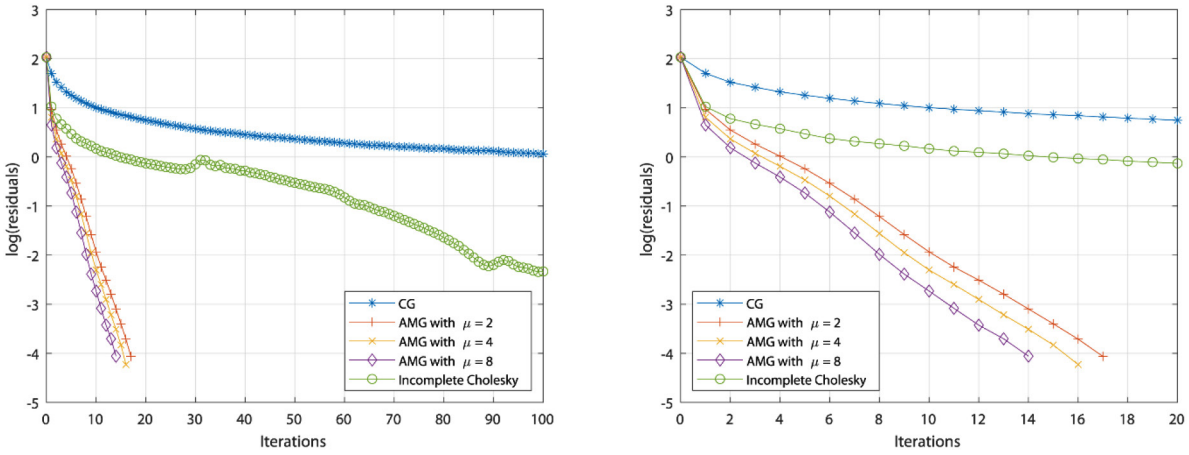

Figure 9. Example 2: Residual history for 100 iterations (left) and 20 iterations (right).
Note: derived from research.

Conclusions and future directions

In this article, we have described an object oriented implementation of the algebraic multigrid preconditioner introduced in Beck (1999), applied to elliptic problems using unstructured triangular meshes. The numerical experiments presented illustrate the appropriate behavior of the preconditioner. In particular, there is not a considerable increase in the number of iterations of PCG method even when the mesh size is decreased.

In the future we would like to consider a new coarsening strategy, which allow us a better selection for master nodes. In addition, in this work we used only H^1-conforming elements in order to explain the preconditioner in a cleaner way. However, the results presented here can be extended to the case of H(div)-conforming elements. Finally, we are interesting to develop an extension of the Beck's preconditioner to polynomial bases of degree higher than one (that is, $k > 1$), which implies that the degrees of freedom are not only in the vertices, and then the coarsening technique would not be accurate.

Acknowledgements

The work of Jeremías Ramírez was partially supported by Escuela de Matemática of Universidad Nacional, Costa Rica, through the project 0161-17. The work of Esteban Segura was partially supported by Escuela de Matemática and CIMPA, Universidad de Costa Rica, Costa Rica, through the project 821-B7-254. The work of Filánder Sequeira was partially supported by Escuela de Matemática of Universidad Nacional, Costa Rica, through the project 0103-18.

References

Beck, R. (1999). *Graph-based algebraic multigrid for Lagrange-type finite elements on simplicial meshes*. Berlin, Germany: Konrad Zuse Zentrum für Informationstechnik. Recuperado de https://pdfs.semanticscholar.org/c719/beb475ff33cab3f235c361f7e8d76e9f820c.pdf

Beirão da Veiga, L.; Brezzi, F.; Cangiani, A.; Manzini, G.; Marini, L. D. & Russo, A. (2013). Basic principles of virtual element methods. *Mathematical Models and Methods in Applied Sciences*, *23*(01), 199-214. doi: https://doi.org/10.1142/S0218202512500492

UNICIENCIA Vol. 34, N° 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

Burden, R. L.; Faires, J. D. & Burden, A. M. (2015). *Numerical Analysis* (10th ed.). United States: Cengage Learning.

Carstensen, C. & Klose, R. (2002). Elastoviscoplastic finite element analysis in 100 lines of Matlab. *Journal of Numerical Mathematics*, *10*(3), 157-192. doi: https://doi.org/10.1515/JNMA.2002.157

Castillo, P. E. & Sequeira, F. A. (2013). Computational aspects of the Local Discontinuous Galerkin method on unstructured grids in three dimensions. *Mathematical and Computer Modelling*, *57*(9-10), 2279-2288. doi: https://doi.org/10.1016/j.mcm.2011.07.032

Chen, K. (2005). *Matrix preconditioning techniques and applications*. United Kingdom: Cambridge University Press. doi: https://doi.org/10.1017/CBO9780511543258

Ciarlet, P. G. (2002). *The finite element method for elliptic problems.* United States: SIAM. doi: https://doi.org/10.1137/1.9780898719208

Cockburn, B. & Shu, C. W. (1998). The local discontinuous Galerkin method for time-dependent convection-diffusion systems. *SIAM Journal on Numerical Analysis*, *35*(6), 2440-2463. doi: https://doi.org/10.1137/S0036142997316712

Johnson, C. (2009). *Numerical solution of partial differential equations by the finite element method*. United States: Courier Corporation.

Ruge, J. W. & Stüben, K. (1987). Algebraic multigrid. In: S.F., McCormick. (Ed.), *Multigrid Methods* (pp. 73-130). United States: SIAM. doi: https://doi.org/10.1137/1.9781611971057.ch4

Saad, Y. (1996). ILUM: a multi-elimination ILU preconditioner for general sparse matrices. *SIAM Journal on Scientific Computing*, *17*(4), 830-847. doi: https://doi.org/10.1137/0917054

Saad, Y. (2003). *Iterative methods for sparse linear systems* (2th ed.). United States: SIAM. Recuperado de https://www-users.cs.umn.edu/~saad/IterMethBook_2ndEd.pdf

Shewchuk, J. R. (1996). Triangle: engineering a 2D quality mesh generator and Delaunay triangulator. *In Workshop on Applied Computational Geometry* (pp. 203-222). Heidelberg, Germany: Springer. doi: https://doi.org/10.1007/BFb0014497

Stüben, K. (2001). A review of algebraic multigrid. In: C., Brezinski, L., Wuytack. (Eds.), *Numerical Analysis: historical Developments in the 20th Century* (pp. 331-359). Elsevier. doi: https://doi.org/10.1016/B978-0-444-50617-7.50015-X

Trottenberg, U.; Oosterlee, C. W. and Schuller, A. (2000). *Multigrid*. United States: Academic Press.

Wilkinson, J. H. (1994). *Rounding errors in algebraic processes*. United States: Courier Corporation.

UNICIENCIA Vol. 34, N° 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

ANNEX 1: MATLAB Class for algebraic multigrid preconditioner

```
% ------------------------------------------------------------
% Class for the Algebraic Multigrid Preconditioner
% introduced by Rudolf Beck
% ------------------------------------------------------------
classdef amgRB
    properties (GetAccess=private)
        maxSize = 100;  % Default values
        numVCycles = 5; % Default values
        A   = [];     % Matrix for first level
        MA = {};     % List of matrices
        MR = {};     % List of restriction operators
        MP = {};     % List of prolongation operators
        L   = [];     % For LU factorization of the last matrix
        U   = [];     % For LU factorization of the last matrix
        tol = 1e-6; % Tolerance for the iterative solver
        steps = []; % Smoothing steps for Gauss-Seidel (GS)
        ML = {};     % List of lower triangular matrices for GS
        MU = {};     % List of upper triangular matrices for GS
        MFGS = {};  % List of inverse matrices for Forward-GS
        MBGS = {};  % List of inverse matrices for Backward-GS
    end
    % ------------------------------------------------------------
    %                       PUBLIC METHODS
    % ------------------------------------------------------------
    methods
    % ------------------------------------------------------------
    % Construct method
    % This function constructs the levels of the algebraic
    % multigrid preconditioner.
    % ------------------------------------------------------------
    % Receive:
    %   A: is a mxm symmetric positive definite sparse matrix.
    %   maxSize: is the maximum size of the matrix in the last
    %            level.
    %   numVCycles: is the number of V-cycles.
    % Update:
    %   MA: is a list of matrices for coarse matrices
    %   MR: is a list of matrices for restriction operators
    %   MP: is a list of matrices for prolongation operators
    %   L: is the lower triangular matrix of the LU factorization
    %      of the last matrix A
    %   U: is the upper triangular matrix of the LU factorization
    %      of the last matrix A
    %   steps: is a list of the smoothing steps for Gauss-Seidel
    %   ML: is a list of lower triangular matrices for GS
```

UNICIENCIA Vol. 34, N°. 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

```
%   MU: is a list of upper triangular matrices for GS
%   MFGS: is a list of inverse matrices for Forward-GS
%   MBGS: is a list of inverse matrices for Backward-GS
% -------------------------------------------------------------
function obj = amgRB(A, maxSize, numVCycles)
    obj.A = A;
    obj.maxSize = maxSize;
    obj.numVCycles = numVCycles;
    % ----------------- Create the levels -----------------
    numLevels = 0; % Number of levels without the last one
    m = length(A); % Order of the matrix
    while m >= maxSize
        % Mark nodes as Master and Slave
        [marked, sizeM] = coarsening(obj, A);
        % Construct Restriction matrix
        R = sparse(sizeM, m);
        for j = 1 : m
            if marked(j) == 0
                % -------------- SLAVE NODE --------------
                c = A(:,j);  % Define the jth column of A
                I = find(c); % Find indices of non-zero
                             % elements of the jth column of A
                [~,~,I] = find( marked(I) ); % Find indices
                                             % of master nodes
                                             % around node j
                nnz = length(I); % Number of neighboring
                                 % master nodes of node j
                R(I, j) = 1 / nnz;
            else
                % -------------- MASTER NODE --------------
                R(marked(j), j) = 1;
            end
        end
        % Store matrices on lists
        numLevels = numLevels + 1;
        obj.MR{numLevels} = R;
        obj.MA{numLevels} = A;
        obj.MP{numLevels} = R' ;
        % Preparations for next level
        A = R * A * R' ;
        m = sizeM;
    end
    % ---- Compute LU Factorization for the last matrix ---
    [L,U] = lu(A);
    obj.L = L;
    obj.U = U;
    % ---------- Create list of smoothing steps -----------
```

UNICIENCIA Vol. 34, N°. 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

```
    obj.steps = zeros(numLevels, 1);
    for i = 1 : numLevels
        obj.steps(i) = i + 1;
    end
    % ---------------- Setup Gauss-Seidel -----------------
    obj.ML{numLevels}   = []; % Store memory
    obj.MU{numLevels}   = []; % Store memory
    obj.MFGS{numLevels} = []; % Store memory
    obj.MBGS{numLevels} = []; % Store memory
    % Create Gauss-Seidel matrices
    for l = 1 : numLevels
        A = obj.MA{l};      % Get matrix A_l
        I = speye(size(A)); % Create identity matrix
        obj.ML{l} = tril(A,-1); % Lower triangular part of A
        obj.MU{l} = triu(A, 1); % Upper triangular part of A
        obj.MFGS{l} = tril(A) \ I; % Forward-GS matrix
        obj.MBGS{l} = triu(A) \ I; % Backward-GS matrix
    end
end
% ---------------------------------------------------------
% Get the number of levels of the preconditioner
% ---------------------------------------------------------
%   k: is the number of levels.
% ---------------------------------------------------------
function [k] = getNumLevels(obj)
    k = length(obj.MR) + 1;
end
% ---------------------------------------------------------
% Apply preconditioner to a vector, used by PCG method
% ---------------------------------------------------------
%   x, y: are m-size vectors
% ---------------------------------------------------------
function [y] = apply(obj, x)
    y = zeros( size(x) );
    [y, ~, ~] = solve(obj, x, y);
end
% ---------------------------------------------------------
% Iterative solver for Ax = b
% ---------------------------------------------------------
%   b: is the right-hand side of the system
%   x0: is an initial guess for system Ax = b
%   x: is the new approximation for system Ax = b
%   cyclesUsed: is the number of V-cycles used
%   normResidual: is a list for the norm-2 of the
%                 residual for each V-cycle
% ---------------------------------------------------------
function [x, cyclesUsed, normResidual] = solve(obj, b, x0)
```

UNICIENCIA Vol. 34, N°. 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

```
numLevels = length(obj.MR); % Number of levels without
                            % the last one, that is, N-1
% ---------- When the number of levels is 1 -----------
if numLevels == 0
    % Solve system using LU Factorization
    x = obj.U \ ( obj.L \ b );
    cyclesUsed = 0;
    normResidual(1) = norm(b - obj.A*x);
    return % Leave the function
end
% ------------ Compute the first residual ------------
A = obj.A;      % Matrix of the first level
r = b - A*x0;   % First residual
nrmR = norm(r); % Norm-2 of the residual
normResidual(1) = nrmR;
nrm0Tol = obj.tol * nrmR; % Tolerance relative to the
                          % first residual
% ------------------- Star V-cycles -------------------
cyclesUsed = 0;
while cyclesUsed < obj.numVCycles && nrmR >= nrm0Tol
    % ------------ Go downward in V-cycle -------------
    for l = 1 : numLevels
        A = obj.MA{l};        % Get matrix A_l
        x = zeros( size(r) ); % Create solution x_l
        % -----------------------------------------
        x = forwardGaussSeidel(obj, r, x, l);
        % -----------------------------------------
        % Stores new vectors
        approxim{l} = x;
        residual{l} = r;
        % Restrict residual
        r = obj.MR{l} * ( r - A*x );
    end
    % --- Perform LU Factorizacion in the last level --
    approxim{numLevels+1} = obj.U \ ( obj.L \ r );
    % ------------- Go upward in V-cycle --------------
    for l = numLevels : -1 : 1
        A = obj.MA{l};   % Get matrix A_l
        x = approxim{l}; % Get vector x_l
        r = residual{l}; % Get vector r_l
        % Prolonging residual
        x = x + obj.MP{l} * approxim{l + 1};
        % -----------------------------------------
        x = backwardGaussSeidel(obj, r, x, l);
        % -----------------------------------------
        approxim{l} = x; % Stores new approximation
    end
```

UNICIENCIA Vol. 34, N°. 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

```
        % ----------------- End of V-cycle ----------------
        r0 = b - A*x; % Create new residual for the first
                      % level, but this does not modify the
                      % original
        nrmR = norm(r0);
        normResidual(cyclesUsed + 2) = nrmR;
        cyclesUsed = cyclesUsed + 1;
    end
end
end % End Public Methods
% ---------------------------------------------------------
%                      PRIVATE METHODS
% ---------------------------------------------------------
methods (Access = private)
% ---------------------------------------------------------
% This function marks the master and slave nodes for a
% given matrix A of order m.
% ---------------------------------------------------------
%   marked: is a m-size vector which contains 0 on its ith
%           component if node i is slave. Otherwise, node
%           i is master, and the ith component of the vector
%           contains a new numbering for node i.
%   sizeM: is the number of master nodes.
% ---------------------------------------------------------
function [marked, sizeM] = coarsening(obj, A)
    m = length(A); % Order of the matrix
    % ---------------------------------------------------
    nnzRow = zeros(1, m); % Number of non-zero elements of
                          % A by row
    for i = 1 : m
        nnzRow(i) = nnz( A(i,:) );
    end
    [~, idxRow] = sort(nnzRow); % Sort indices according to
                          % the number of non-zero elements
    % ---------------------------------------------------
    % First, every node is marked with a negative number,
    % in order to indicate that the node is unmarked.
    marked = -1 * ones(1, m);
    sizeM  = 0;
    % Using the symmetry of A, we follow the columns of A:
    for j = idxRow
        c = A(:,j);  % Define the jth column of A
        I = find(c); % Find indices of non-zero elements
        if marked(j) == -1 % Unmarked node
            % Node j has not been marked yet. That's
            % why it will be marked as master and its
            % neighboring nodes will be slaves.
```

UNICIENCIA Vol. 34, N°. 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr

ISSN Electrónico: 2215-3470
DOI: http://dx.doi.org/10.15359/ru.34-2.4

```
                % ---------------------------------------------
                % Mark its neighbors as slaves:
                marked(I) = 0;
                % Mark as master:
                sizeM = sizeM + 1; % A master node was found
                marked(j) = sizeM; % New numbering for node j
            end
        end
    end
    % -------------------------------------------------------
    % Forward Gauss-Seidel iteration.
    % -------------------------------------------------------
    %   b: is the right-hand side vector.
    %   x: is the initial vector.
    %   l: is the corresponding level.
    % -------------------------------------------------------
    function x = forwardGaussSeidel(obj, b, x, l)
        numSteps = obj.steps(l);
        for k = 1 : numSteps
            x = obj.MFGS{l} * ( b - obj.MU{l}*x );
        end
    end
    % -------------------------------------------------------
    % Backward Gauss-Seidel iteration.
    % -------------------------------------------------------
    %   b: is the right-hand side vector.
    %   x: is the initial vector.
    %   l: is the corresponding level.
    % -------------------------------------------------------
    function x = backwardGaussSeidel(obj, b, x, l)
        numSteps = obj.steps(l);
        for k = 1 : numSteps
            x = obj.MBGS{l} * ( b - obj.ML{l}*x );
        end
    end
    end % End Private Methods
end
% ---------------------------------------------------------
%                         END OF THE FILE
% ---------------------------------------------------------
```

UNICIENCIA Vol. 34, N°. 2, pp. 55-81. Julio-Diciembre, 2020 • URL: www.revistas.una.ac.cr/uniciencia • Correo electrónico: revistauniciencia@una.cr