PROJETO COMPUTACIONAL #3 – MS993/MT404 – 2S2016 – IMECC/UNICAMP
Matemática Aplicada

Atenção: Este projeto #3 poderá ser desenvolvido individualmente ou em grupos de até no máximo 3 (três) estudantes.

Atividade	**Temas**	**Disponibilização**	**Entrega**
Projeto #3 (**P3**) Peter Sonneveld, CGS, A Fast Lanczos-Type Solver for Nonsymmetric Linear systems, SIAM J. Sci. and Stat. Comput., (1988) 10(1), 36-52. (17 pages) H. A. van der Vorst, Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems,SIAM J. Sci. and Stat. Comput., (1991) 13(2), 631-644. (14 pages)	Iterative solvers, Sparse linear systems, symmetric matrices, nonsymmetric/unsymmetric (mainly), BiConjugate Gradient (BiCG), Conjugate Gradient Squared (CGS), Biconjugate Gradient Stabilized (Bi-CGSTAB)**, Krylov methods,** Matlab native codes “bicgstab” and “cgs”; Preconditioning, Lanczos algorithm, floating point arithmetic, accuracy, stability, efficiency, and applications (University of Florida Sparse Matrix Collection and Matrix Market).	30/set	14/Out

The primary learning goal of the project. The goal of this task is explore the application of two Krylov subspace methods, namely, the CGS and the Bi-CGSTAB, by using matlab routines. The original references for such methods are:

- Peter Sonneveld, CGS, A Fast Lanczos-Type Solver for Nonsymmetric Linear systems, SIAM J. Sci. and Stat. Comput., (1988) 10(1), 36-52.
- H. A. van der Vorst, Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems,SIAM J. Sci. and Stat. Comput., (1991) 13(2), 631-644.

We are now in a position to test iterative solvers for sparse linear systems for both nonsymmetric and symmetric matrices. Type at matlab help Syntax “bicgstab” and “cgs” (see also the examples in what follows). The homework should be done under Matlab to take advantage of the available routines (it is highly recommended). The project is organized in five sections as described in what follows, covering numerics, applications and theory.

I) OVERVIEW: Consider the solution of

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

where $A \in R^{NxN}$, $x \in R^{Nx1}$, and the nonzero vector $b \in R^{Nx1}$, with a integer N large so that we have a very large sparse matrix A. In practice, when the coefficient matrix A is sparse, people always use iterative methods rather than direct methods, say, Gaussian Elimination (GE), for the solution of (1). The reason is that, GE needs to perform two basic steps to solve (1), as follows:

1. Decompose A as A = LU where L is lower triangular and U is upper triangular;
2. Solve Ly = b (for “y”) and Ux = y (for “x”) by forward and backward substitutions, respectively.

However, even though A is sparse, the resulting L and U are usually dense. Although Gaussian elimination is straightforward to program, and rounding errors may be controlled (e.g., by the method of “Partial pivoting” or “Rook pivoting”), it requires then a “N x (N+1)” augmented matrix to be stored, which may be inefficient if N is very large, but most of the coefficients of A are zero. Such systems are called sparse, and some routines (e.g., CG, PCG, Bi-CG and Bi-CGSTAB) may be more efficient for large, sparse systems. Morerover, if one uses GE to solve (1), one needs to store the L and U for the forward $O(N^2)$ and backward $O(N^2)$ substitutions in step #2. Therefore, the storage requirement of GE is $O(N^2)$ of floating point numbers. This is still a huge memory requirement since it is not unusual in industry and many other solutions to real-life problems worldwide (with tremendous global issues: social, political, economic, environmental etc.…) and thus $N=10^9$, $N=10^{12}$ (or more), and hence, $N^2 = 10^{18}$ (or $N=10^{24}$) which is something out of the storage capacity of most of the non-specialized and specialized computers nowadays. As specialized computers, we mean computers to execute high performance parallel computing (MPI, OpenMP, Multicore, etc...) associated to applications.

Iterative methods, on the other hand, are cheap in storage. The typical operation in an iterative method is a matrix (G) times a vector (v), i.e., “Gv”. The matrix G is usually constructed from A and is sparse. Hence G occupies only a little memory in a computer. Therefore, the memory required to implement an iterative method is a very limited and that is why this kind of methods is so popular in practice when solving large, sparse linear systems. Iterative methods nowadays can be divided into three groups (see, e.g., ref [16,17]):

i) Basic iterative methods;

ii) Krylov subspace methods;

iii) Multigrid methods.

In this project, we will only discuss and compare some of the methods from the second group through numerical experiments supported by theory (see, e.g., [1]-[17]). Some well-known methods from the first group was already addressed in Project #2. The methods from the third group is not covered in this course, but for those interested might take a look at first glance at the reference **[14]** in our bibliography, namely, “William L. Briggs, Van Emden Henson and Steve F. McCormick. A multigrid tutorial, 2nd ed, PA, SIAM (2000)”.

In the following lines we will overview some issues concerning **CGS, BiCG and Bi-CGSTAB** in order to highlight some key issues and than facilitates the reading for theoretical purposes.

BiConjugate Gradient (BiCG). The Biconjugate Gradient method generates two Conjugate Gradient like sequences of vectors, one based on a system with the original coefficient matrix A, and one on A^T (transpose of A). Instead of orthogonalizing each sequence, they are made mutually orthogonal, or “bi-orthogonal”. This method, like CG, uses limited storage (as in CG, this is remarkable). It is useful when the matrix is nonsymmetric and nonsingular; however, convergence may be irregular (as in CG), and there is a possibility that the method will break down. BiCG requires a multiplication with the coefficient matrix and with its transpose at each iteration. We point out that the conjugate gradient method is not suitable for nonsymmetric systems because the residual vectors cannot be made orthogonal with short recurrences, as proved in V.V. Voevodin (1983) and V. Faber and T. Manteuffel (1984). A natural improvement of the CG is by the use of preconditioning (PCG). The generalized minimal residual method (GMRES) retains orthogonality of the residuals by using long recurrences, at the cost of a larger storage demand (the GMRES method will be addressed in a forthcoming course project). The biconjugate gradient method (BiCG) takes another approach, replacing the orthogonal sequence of residuals by two mutually orthogonal sequences, at the price of no longer providing a minimization.

Conjugate Gradient Squared (CGS). The Conjugate Gradient Squared method is a variant of BiCG that applies the updating operations for the A-sequence and the A^T-sequences both to the same vectors. Ideally, this would double the convergence rate, but in practice convergence may be much more irregular than for BiCG. A practical advantage is that the method does not need the multiplications with the transpose of the coefficient matrix. Often one observes a speed of convergence for CGS that is about twice as fast as for the biconjugate gradient method (BiCG), which is in agreement with the observation that the same "contraction" operator is applied twice. However, there is no reason that the contraction operator, even if it really reduces the initial residual, should also reduce the once reduced vector. This is evidenced by the often highly irregular convergence behavior of CGS. One should be aware of the fact that local corrections to the current solution may be so large that cancellation effects occur. This may lead to a less accurate solution than suggested by the updated residual (van der Vorst 1992). The method tends to diverge if the starting guess is close to the solution. CGS requires about the same number of operations per iteration as the BiCG, but does not involve computations with A^T. Hence, in circumstances where computation with A^T is impractical, CGS may be attractive.

Biconjugate Gradient Stabilized (Bi-CGSTAB). The Biconjugate Gradient Stabilized method is a variant of BiCG, like CGS, but using different updates for the A^T-sequence in order to obtain smoother convergence than CGS. The conjugate gradient method is not suitable for nonsymmetric systems because the residual vectors cannot be made orthogonal with short recurrences, as proved in V.V. Voevodin (1983) and Faber and Manteuffel (1984). The generalized minimal residual method (GMRES) retains orthogonality of the residuals by using long recurrences, at the cost of a larger storage demand. The BiCG takes another approach, replacing the orthogonal sequence of residuals by two mutually orthogonal sequences, at the price of no longer providing a minimization. The Bi-CGSTAB method was developed to solve nonsymmetric linear systems while avoiding the often irregular convergence patterns of the conjugate gradient squared method (van der Vorst 1992). Bi-CGSTAB often converges about as fast as the conjugate gradient squared method (CGS), sometimes faster and sometimes not. CGS can be viewed as a method in which the biconjugate gradient method (BiCG) "contraction" operator is applied twice. Bi-CGSTAB can be interpreted as the product of BCG and repeated application of the generalized minimal residual method. BCGSTAB requires two matrix-vector products and four inner products, i.e., two inner products more than the biconjugate gradient method or the conjugate gradient squared method (van der Vorst 2003).

II) THE MATLAB PROGRAMS TO HANDLE SPARSE MATRICES "bicgstab" and "cgs". Type at matlab help Syntax "bicgstab" and "cgs" in order to learn about the "bicgstab" and "cgs" Matlab programs to handle sparse matrices we will consider "SHERMAN5" problem from set SHERMAN, from the Harwell-Boeing Collection. For a description of the "sherman5" matrix (actually, A and b linked to a unsymmetric sparse linear system) see the following links. We will consider other similar sample matrices these sites.

Matrix Market: http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/sherman/sherman5.html

UF Sparse Matrix Collection: http://www.cise.ufl.edu/research/sparse/matrices/HB/sherman5.html

The University of Florida Sparse Matrix Collection is a large and actively growing set of sparse matrices that arise in real applications. The Collection is widely used by the numerical linear algebra community for the development and performance evaluation of sparse matrix algorithms. It allows for robust and repeatable experiments. Its matrices cover a wide spectrum of domains, include those arising from problems with underlying 2D or 3D geometry (as structural engineering, computational fluid dynamics, model reduction, electromagnetics, semiconductor devices, thermodynamics, materials, acoustics, computer graphics/vision, robotics/kinematics, and other discretizations) and those that typically do not have such geometry (optimization, circuit simulation, economic and financial modeling, theoretical and quantum chemistry, chemical process simulation, mathematics and statistics, power networks, and other networks and graphs.

The Matrix Market ia a visual repository of test data for use in comparative studies of algorithms for numerical linear algebra, featuring nearly 500 sparse matrices from a variety of applications, as well as matrix generation tools and services.

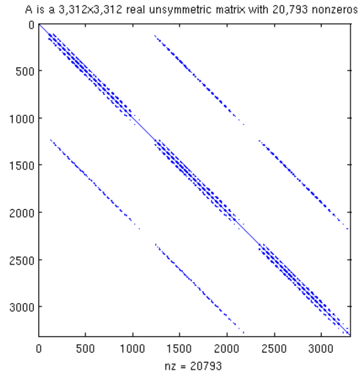
Although the last change in Matrix Market is 10 May 2007 (http://math.nist.gov/MatrixMarket/index.html), one can find a lot of very nice example there. Indeed, nowadays, most of the content of the Matrix Market is now at the UF Sparse Matrix Collection http://www.cise.ufl.edu/research/sparse/matrices/ that is maintained by Tim Davis, last updated 10-Jun-2015.

SHERMAN model problem. About the "SHERMAN5" problem from set SHERMAN, from the Harwell-Boeing Collection. In the summer of 1984, Andy Sherman of Nolan and Associates, Houston, TX, USA, issued a challenge to the petroleum industry and the numerical analysis community for the fastest solution to a set of 5 systems of linear equations extracted from oil reservoir modeling programs. These are those five matrices. Each matrix arises from a three dimensional simulation model on an NX x NY x NZ grid using a seven-point finite-difference approximation with NC equations and unknowns per grid block. The corresponding right-hand side vector is also supplied. See the above links for more information.

Example run using Matlab bicg native program. First, you will download and save the zip file “aux-files-taskP3.zip” at http://www.ime.unicamp.br/~ms993/exerc%C3%ADcios in your computer. Second, unzip aux-files-taskP3.zip to appear the directory aux-files-taskP3 and then move the prompt command to this folder.

Caution. When using Matlab programs you MUST take care to NOT save your own programs cg, bicgstab or others with the SAME name into the aux-files-taskP3 folder. In other words, in the folder with your own Matlab codes take care to NOT contain “qmr”, “bicg”, “cgs”, “bicgstab”, etc... as well as any other related program with the **same name** as available in maltab enviroment. In what follows, commands after ">>" are proper command lines of matlab prompt and the blue color text after % are just some comments about what we are doing here.

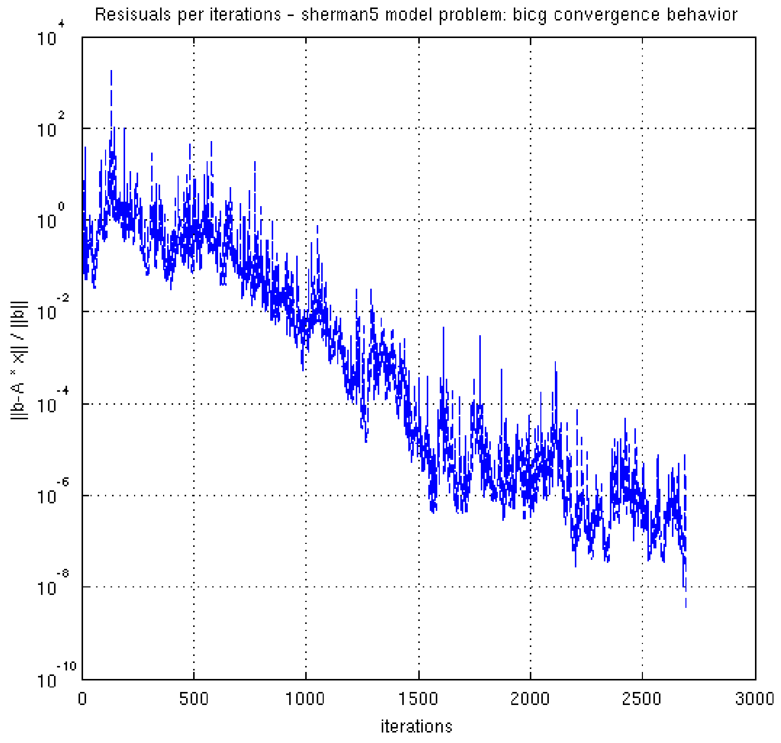
A is a 3,312x3,312 real unsymmetric matrix with 20,793 nonzeros



First, move into folder aux-files-taskP3 and invoke matlab. Next, type the commands:

```
>> load('sherman5.mat');  % loading sherman5.mat file with A and b entries

>> A= Problem.A; % create A in matlab format

>> spy(A); % looking at the A matrix structure (the right figure will appear)

>> [m,n] = size(A); % a way to get dimensions of the matrix A (m-rows and n-colums)

>> xtrue = 0.5 + sin( pi*(0:n-1)'/(n-1)); % choose a known solution vector

>> b = A * xtrue ;  % choose the corresponding right hand side

>> x0 = zeros(n,1);  % a vector of all zeros initial “guess” or “approximation”

>> M = speye(m);  % let M be an m by m sparse identity matrix

>> max_it = 500;  % number of iterations

>> tol = 1.e-8;  % tolerance

>> tic, [x,flag,relres,iter,resvec] = bicg(A,b,tol,max_it);t_bicg = toc; % running the BiConjugate Gradient

>> flag

flag =

     1    % flag=1 means that bicg did not converge

>> max_it=5000;  % So max_it is possibly too small, then you might reset it

>> tic, [x,flag,relres,iter,resvec] = bicg(A,b,tol,max_it);t_bicg = toc; % running the BiConjugate Gradient

>> flag
```

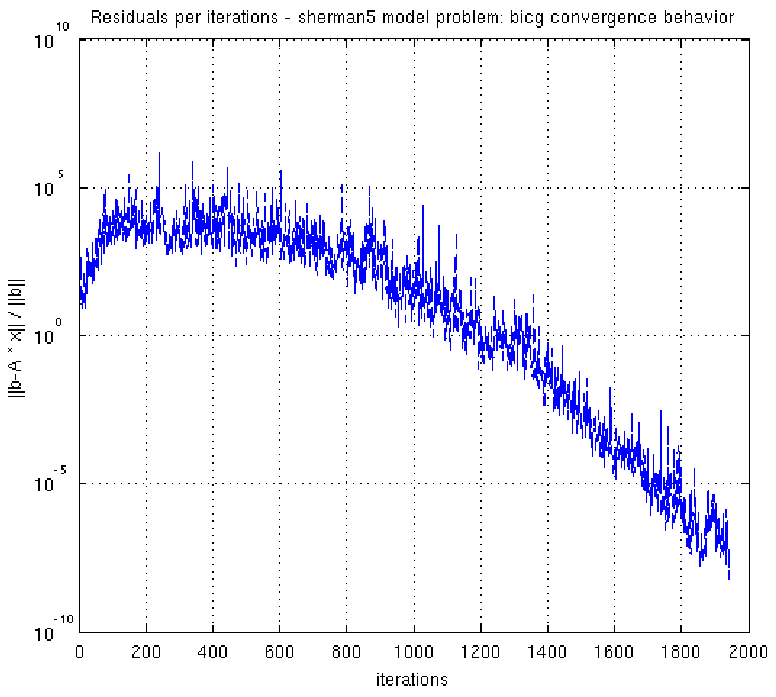
```
flag =
     0     % flag=0 means that bicg did converge
           % To plot the graph of the residuals per iterations, you need to take the following steps
>> semilogy(1:length(resvec),resvec/norm(b),'--'),grid,  %
>> title('Residuals per iterations - sherman5 model problem: cgs convergence behavior')
>> xlabel('iterations')
>> ylabel('||b-A * x|| / ||b||')
           % If everything goes fine, you must see the below graph
```



If the above example we show how to construct a proprer right hand side (rhs) b aiming the use of the matlab native program bicg. But we can also use the rhs associated to the SHERMAN model problem. To this end, just redo all the above command as follows:

```
>> clear
>> clc
>> load('sherman5.mat');
>> A= Problem.A;
>> b= Problem.b;
>> spy(A);
>> spy(b);
>> [m,n] = size(A);
>> x0 = zeros(n,1);
>> M = speye(m);
>> tol = 1.e-8;

>> max_it=5000;

>> tic, [x,flag,relres,iter,resvec] = bicg(A,b,tol,max_it);t_bicg = toc;

>> flag

flag =

    0

>> semilogy(1:length(resvec),resvec/norm(b),'--'),grid,

>> title('Residuals per iterations - sherman5 model problem: bicg convergence behavior')

>> xlabel('iterations')

>> ylabel('||b-A * x|| / ||b||')
```

% As before, if everything goes fine, you must see the below graph



Try yourself the following list of commands after you have rerun the "bicg matlab command" for the both rhs, as discussed in the above.

```
>> condest(A)   % What is does means ?

>> M = speye(m);   % What is does means ?

>> nnz(A)   % What is does means ?

>> [m, n] = size(A)   % What is does means ?

>> iter   % What is does means ?

>> t_bicg   % What is does means ?

>> relres   % What is does means ?

>> flag   % What is does means ?
```

III) TASK NUMERICS

The computational bulk of work in the project #3. We now proceed to present and discuss the tasks into this project #3 with the help of “bicgstab” and “cgs” Matlab programs. Thus, by using “bicgstab” and “cgs”, solve large and sparse nonsymmetric/unsymmetric linear systems associtad to the matrices that appears in the below table.

Caution. For all cases start with tol = 1.e-8 and max_it=5000 and it is your task verify if the rhs is avalible or not; if “yes” than you must used (at least) otherwise you should construct the corresponding rhs b.

Issue Warnings and Errors. You can issue a warning to flag unexpected conditions detected when running a program. The warning function prints a warning message to the command line. Warnings differ from errors in two significant ways: 1) Warnings do not halt the execution of the program and 2) You can suppress any unhelpful MATLAB® warnings. Chekc the value “flag”. Do not ignore warning and error messages in matlab! They help a lot in most of the cases to shed light about what is going wrong.

Solve Ax=b for real unsymmetric linear systems of equations linked fo the files in the below table. The files are in the folder aux-files-taskP3. Matrix id (the id is unique in the UF Sparse Matrix Collection)

Group	Matrix name	Matrix id	Tested ?	Computational time
HB/sherman3	sherman3.mat	244	yes	Less than 1 minute
HB/sherman5	sherman5.mat	246	yes	Less than 1 minute
HB/arc130	arc130.mat	6	yes	Less than 1 minute
Engwirda/airfoil_2d	airfoil_2d.mat	1319	yes	Less than 1 minute
------	------	------	------	------
Bai/dw4096	dw4096.mat	312	yes	(*) Bonus
AtandT/pre2	pre2.mat	285	yes	(*) Bonus
AtandT/onetone1	onetone1.mat	283	yes	(*) Bonus

(*) By using the same configurations and setting as above for “bicgstab” and “cgs” we do not have success. Messages of "warnings" were displayed (type, e.g., flag and take a look at the matlab help.

Moreover, with the same parameters bicg and gmres did not worked properly (gmres with lack of memory!) What to do? It is like a good challenge. I will seriously consider to give a credit of an extra (bonus) point for those that are able to solve ANY one of the above three examples dw4096.mat or pre2.mat or onetone1.mat using ONLY the “bicgstab” and/or “cgs” (it is allowed to use any one of the many settings of matlab native codes “bicgstab” and/or “cgs”; type matlab help).

IV) TASK APPLICATIONS

As part of this project #3, you must search for the science, engineering application, economic, social, etc... behind **ALL** the model problem matrices displayed in above table (even if you do not able to devise a way to solve the pertinent linear system at hand) and than write a summary (maximum half of a page per item) describing what is going on into these very nice looking matrices; see the “geometry” of such matrices (search by id) at http://www.cise.ufl.edu/research/sparse/matrices/list_by_id.html

Applications are an essential complement to our research and development in algorithm design, numerical libraries, collaborative middleware, data analysis and visualization, and exascale hardware and software as well as to do good science and applied mathmatics, or just mathematics in a open minded view!

For concreteness, applied mathematic scientists carry out investigations in such several diciplines as climate science, engineering diagnostics, materials science, nuclear reactor simulation, and metabolic modeling with deep impact in real worl life to improve scientific, economic and social activities.

V) TASK THEORY

The thoery behind the numerical Krylov subspace method CGS and Bi-CGSTAB.

Consider the below key references (of course, you are free to consult any additional reference you wish,).

- Peter Sonneveld, CGS, A Fast Lanczos-Type Solver for Nonsymmetric Linear systems, SIAM J. Sci. and Stat. Comput., (1988) 10(1), 36-52.

- H. A. van der Vorst, Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems,SIAM J. Sci. and Stat. Comput., (1991) 13(2), 631-644.

Now, describe in detailed the algorithms CGS and Bi-CGSTAB (from a theoretical viewpoint). **Hint**: it might be useful to consult the following codes in the “aux-files-taskP3”, namely “template-cgs.m” and “template-bicgstab.m”. In addition, it also might be useful take a look at the Matlab hehp and check for yourself the matlab documentation to learn more about the bicg and bicgstab Syntaxes as follows:

bicg Biconjugate gradients method Syntax

```
x = bicg(A,b)
bicg(A,b,tol)
bicg(A,b,tol,maxit)
bicg(A,b,tol,maxit,M)
bicg(A,b,tol,maxit,M1,M2)
bicg(A,b,tol,maxit,M1,M2,x0)
[x,flag] = bicg(A,b,...)
[x,flag,relres] = bicg(A,b,...)
[x,flag,relres,iter] = bicg(A,b,...)
[x,flag,relres,iter,resvec] = bicg(A,b,...)
```

bicgstab Biconjugate gradients stabilized method Syntax

```
x = bicgstab(A,b)
bicgstab(A,b,tol)
bicgstab(A,b,tol,maxit)
bicgstab(A,b,tol,maxit,M)
bicgstab(A,b,tol,maxit,M1,M2)
bicgstab(A,b,tol,maxit,M1,M2,x0)
[x,flag] = bicgstab(A,b,...)
[x,flag,relres] = bicgstab(A,b,...)
[x,flag,relres,iter] = bicgstab(A,b,...)
[x,flag,relres,iter,resvec] = bicgstab(A,b,...)
```

VI) GENERAL COMMENTS TO THINK AND TO BE ADDRESSED DURING PREPARATION FOR THIS PROJECT

- Does bicgstab converge faster than bicg (or vice-versa)? Is the extra work (if any) per step worth the reduction in steps? Does bicg fail more frequently?

- Does cgs require substantially less work than bicstab? Is the reduction of work per step worth the extra iterations (if any)?

- Is CGS better than bicg in terms of iterations, run time or reliability?

- Compare any two methods (bicg, cgs or bicgstab) in the same way.

- Recall the table from class which suggests some natural comparisons (direct methods versus iterative, Krylov subspace methods)

- Does matlab worked well for all cases ? If not, what would be a possible cure to circunvent this ?

- Finally, try experiments (including theory and numerics) not mentioned above, of your choice!

VII) REFERÊNCIAIS (disponíveis na biblioteca do IMECC e com reserva de curso)

[1] Carl D. Meyer, Matrix Analysis and Applied Linear Algebra, Philadelphia, PA, SIAM (2000).
[2] Charles L. Lawson and Richard J. Hanson. Solving least squares problems, Philadelphia, SIAM (1995).
[3] David S. Watkins, Fundamentals of Matrix Computations, New Jersey: John Wiley & Sons (3 ed., 2010).
[4] Erwin Kreyszig. Introductory functional analysis with applications, NY, John Wiley & Sons (1978).
[5] Gene Golub and Charles F. Van Loan. Matrix computations 3rd ed., Johns Hopkins Univ. Press (1996).
[6] Gérard Meurant. The Lanczos and conjugate gradient algorithms: from theory to finite precision computations, Philadelphia, PA, SIAM (2006).
[7] Gilbert Strang. Linear algebra and its applications, 3rd ed., Brooks/Cole, Thomson Learning,(1988).
[8] Henk A. van der Vorst. [recurso eletrônico, digital, PDF file] Iterative Krylov Methods for Large Linear Systems, Cambridge, UK, Cambridge University Press - Monographs on applied and computational mathematics; n. 13, (2003).
[9] James W. Demmel. Applied numerical linear algebra, Philadelphia, PA, SIAM (1997).
[10] Kenneth Hoffman and Ray Kunze. Linear algebra, 2nd ed, Englewood Cliffs, NJ, Prentice-Hall (1971).
[11] Lloyd N. Trefethen, David Bau III. Numerical linear algebra, Philadelphia, PA, SIAM (1997).
[12] Roger A. Horn and Charles R. Johnson. Matrix analysis, Cambridge University Press (1985).
[13] Timothy A. Davis, Direct methods for sparse linear systems, SIAM (2006).
[14] William L. Briggs, Van E. Henson and Steve F. McCormick. A multigrid tutorial, 2nd ed. SIAM (2000).
[15] Wolfgang Hackbusch. Iterative solution of large sparse systems of equations, NY, Springer (1994).
[16] Yousef Saad. Iterative methods for sparse linear systems, 2nd ed. Philadelphia, PA, SIAM (2003).
[17] Remark: For more practical issues see Barrett, R.; Berry, M.; Chan, T. F.; Demmel, J.; Donato, J.; Dongarra, J.; Eijkhout, V.; Pozo, R.; Romine, C.; and van der Vorst, H. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd ed. Philadelphia, PA: SIAM, 1994. http://www.netlib.org/linalg/html_templates/Templates.html.