Universidade do Estado do Rio de Janeiro

Centro de Tecnologia e Ciências

Faculdade de Engenharia

Ítalo Cristiano Nievinski Lima

JinSol, Interface em Java Para Sistemas Lineares

Rio de Janeiro

2013

Ítalo Cristiano Nievinski Lima

JinSol, Interface em Java para Sistemas Lineares

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre em Ciências, ao Programa de Pós-Graduação em Engenharia Mecânica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Fenômenos de Transporte.

Orientador: Prof. Dr. Luiz Mariano Paes de Carvalho Filho

Rio de Janeiro

2013

CATALOGAÇÃO NA FONTE

UERJ / REDE SIRIUS / BIBLIOTECA CTC/B

L732 Lima, Ítalo Cristiano Nievinski.
JinSol, interface em Java para sistemas lineares / Ítalo Cristiano Nievinski Lima. - 2014.
70 f.

Orientador: Luiz Mariano Paes de Carvalho Filho.
Dissertação (Mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.

1. Engenharia Mecânica. 2. Java (Linguagem de programação de computador) – Dissertações. 3. Sistemas lineares – Dissertações. I. Carvalho Filho, Luiz Mariano Paes de. II. Universidade do Estado do Rio de Janeiro. III. Título.

CDU 004.4

Assinatura

Data

Ítalo Cristiano Nievinski Lima

JinSol, Interface em Java Para Sistemas Lineares

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre em Ciências, ao Programa de Pós-Graduação em Engenharia Mecânica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Fenômenos de Transporte.

Aprovado em: 31 de Julho de 2013

Banca Examinadora:

Prof. Dr. Luiz Mariano Paes de Carvalho Filho (Orientador)
Instituto de Matemática e Estatística da UERJ

Norberto Mangiavacchi
Faculdade de Engenharia da UERJ

Nelson Maculan Filho
Universidade Federal do Rio de Janeiro - UFRJ - COPPE

Rio de Janeiro

2013

AGRADECIMENTO

Agradeço aos meus pais Marco Antonio e Lourdes Inês pela árdua tarefa de me guiar com maestria nos tortuosos rumos da vida e pela plena confiança nas minhas capacidades e escolhas. Aos meus irmãos Márcio André e Andressa Aline e os demais familiares que sempre me apoiaram.

Agradeço à rosa que um dia disse sim, e perfumou meu mundo com sua fragrância inconfundível, minha amada esposa Giselle Nievinski.

Ao meu orientador, Luiz Mariano Paes de Carvalho que mostrou os caminhos nos momentos de escuridão e soube dar o incentivo correto nas situações onde era necessário.

Agradeço aos amigos. Pois sem eles seria só eu contra o mundo todo, e o mundo todo é muita gente.

Agradeço também ao Crono, Ryu, Vault Dweller, Blue Mary, Griswold, Jill Valentine, Leneth, Solid Snake e dezenas de outros personagens com quem tanto aprendi.

Agradeço, enfim, à FAPERJ pela bolsa de Mestrado concedida, até porque, infelizmente, boa vontade não enche barriga.

RESUMO

LIMA, Ítalo Cristiano Nievinski. *JinSol*, Interface em Java para Sistemas Lineares. 2013. 70f. Dissertação (Mestrado em Engenharia Mecânica) - Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro (UERJ), Rio de Janeiro, 2013.

Este trabalho apresenta o JinSol, um programa em Java, com interface gráfica, que realiza testes de métodos de solução de sistemas lineares para ser usado em estudos comparativos e desenvolvimento de novas técnicas de solução. O programa apresenta ferramentas de análise gráfica e gera tabelas de texto e planilhas com os resultados dos testes que podem ser usadas para comparações de eficiência entre diferentes métodos ou precondicionadores sobre diferentes problemas. Integrado ao Matlab, o JinSol possui uma biblioteca de *solvers*, precondicionadores e matrizes que podem ser utilizados pelo usuário, de forma comparativa, para auxiliar na busca por métodos mais eficientes para tipos específicos de problemas.

Palavras-chave: Sistemas Lineares, Precondicionadores, Métodos de Krylov, Java.

ABSTRACT

LIMA, Ítalo Cristiano Nievinski. *JinSol*, Java Interface for Linear Systems. 2013. 70f. Dissertação (Mestrado em Engenharia Mecânica) - Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro (UERJ), Rio de Janeiro, 2013.

This master thesis presents JinSol, a Java program with graphic user interface which runs tests on linear systems solution methods, that was designed for use in comparative studies and in the development of new solution techniques. This program provides graphical analysis tools and generates text tables and spread sheets with test results that might be used to compare the efficiency among methods or preconditioners on different problems. Integrated with Matlab, the JinSol has a library of solvers, preconditioners and matrices that can be used by the user in a comparative way to help in the search for more efficient methods to specific types of problem.

Keywords: Linear Systems, Preconditioners, Krylov Methods, Java.

LISTA DE FIGURAS

SUMÁRIO

INTRODUÇÃO

Para realizar a simulação de alguns fenômenos físicos é necessário resolver, numericamente, EDPs através da discretização do domínio, gerando um sistema de equações lineares. Estima-se que mais de 80% do tempo de simulação seja fruto da resolução dos sistemas lineares obtidos, tornando este um dos pontos cruciais para a viabilidade de grandes simulações em tempo hábil para análise de dados e tomada de decisões importantes.

Temos atualmente uma grande variedade de métodos de solução de sistemas lineares e precondicionadores disponíveis e uma vasta bibliografia identificando algumas de suas propriedades e resultados com diferentes problemas. Entretanto, ao nos depararmos com um novo problema buscamos também realizar diversos testes afim de obter informações sobre o comportamento de cada método visando encontrar uma escolha viável de solução, tanto em questão de velocidade de processamento quanto em necessidade de armazenamento. Realizar estes testes pode ser uma tarefa exaustiva e consumir tempo devido a dificuldade de se configurar um ambiente de testes, organizar relatórios com os dados dos testes, construir gráficos para análise entre outros aspectos. Da mesma forma ao desenvolver ou modificar um *solver* ou um precondicionador, desejamos igualmente realizar testes comparativos, com outros métodos aplicados sobre diferentes problemas, o que nos leva à um cenário semelhante. Pensando neste cenário foi iniciado o desenvolvimento de uma ferramenta que funcione como plataforma de testes de solucionadores de sistemas lineares com interface gráfica. O objetivo é tornar este processo simples e eficiente, integrando plataformas e linguagens, com portabilidade e capaz de receber novos métodos e ferramentas em forma de *plug-ins* além dos que já estarão configurados para o usuário.

O primeiro capítulo deste trabalho é destinado a apresentação das características dos sistemas lineares, métodos de solução, com foco em métodos iterativos, e algumas técnicas de precondicionamento. O segundo capítulo destina-se a apresentação do programa JinSol como proposta para a ferramenta de testes descrita acima. O terceiro capítulo introduz alguns casos de uso do programa para solução de alguns problemas cujos resultados são apresentados no quarto capítulo.

O Apêndice A apresenta outras informações obtidas dos testes descritos no terceiro capítulo que não foram apresentadas no quarto capítulo. O Apêndice B apresenta

informações sobre o desenvolvimento do programa como os diagramas de classes e especificações técnicas.

1 SISTEMAS LINEARES

O estudo de métodos para a resolução destes sistemas de equações será o principal foco deste capítulo, onde apresentaremos algumas técnicas para solução destes problemas. Primeiro será feita a descrição do problema, em seguida uma apresentação dos métodos de Krylov, uma família de métodos para solução de sistemas lineares, e então serão apresentadas características sobre precondicionadores seguidas por alguns exemplos.

1.1 Solução de Sistemas Lineares

Os sistemas lineares podem ser representados matricialmente da seguinte forma:

$$Ax = b. \tag{1.1}$$

Onde A é a matriz dos coeficientes do sistema de equações obtidos através do processo de dicretização do problema, b é o vetor de constantes do lado direito e x é o vetor de incógnitas a ser determinado e que nos dará as informações que estamos buscando em cada nó, tais como pressão e velocidade.
Para a solução de sistemas lineares existem duas grandes linhas de abordagem;

- Métodos diretos ou exatos;
  Determinam as soluções exatas de um sistema de quações lineares com um número finito de operações em aritmética exata.

- Métodos iterativos ou de aproximação;
  Conduzem à solução aproximada de um sistema de equações lineares a partir de uma estimativa inicial da solução através de um procedimento iterativo. A cada iteração utiliza-se a aproximação anterior da solução para o cálculo de uma nova aproximação, a qual deve convergir para a solução do sistema dentro de uma margem de erro aceitável.

Estamos interessados em resolver sistemas de grande porte e portanto vamos nos dedicar ao estudo de métodos iterativos, em particular, métodos de Krylov, por suas propriedades, como é explicado em [1, prefácio]: "Não pode restar a menor dúvida de que, atualmente, o melhor e mais utilizado método para resolver sistemas lineares é a

eliminação gaussiana com pivoteamento parcial. Mas a ordem das matrizes a serem resolvidas em problemas atuais alcançam cifras enormes. No caso de matrizes esparsas, sem uma estrutura conhecida, o procedimento padrão de eliminação gaussiana não é indicado, pois rapidamente chega-se à exaustão de memória e, em muitos casos, o tempo necessário para a solução é inviável. Nesse momento, os métodos iterativos são chamados à cena, e dentre eles, os mais utilizados, em aplicações acadêmicas e industriais, são os métodos de Krylov, por suas boas propriedades numéricas e computacionais."

1.1.1 Métodos de Krylov

Métodos de projeção em subespaços de Krylov, conhecidos simplesmente como métodos de Krylov, tratam-se de métodos de projeção. Para falar dos métodos de projeção em subespaços iremos introduzir os conceitos de métodos de projeção e de subespaços de Krylov.

1.1.1.1 Método de projeção

Segundo [2, cap. 5], seja A uma matriz real $n \times n$ e $\mathcal{K}$ e $\mathcal{L}$ dois subespaços m-dimensionais de $\mathbb{R}^n$. Uma técnica de projeção sobre o subespaço $\mathcal{K}$ e ortogonal à $\mathcal{L}$ é um processo que encontra uma solução aproximada $\tilde{\mathbf{x}}$ de (1.1) através da imposição das condições de que $\tilde{\mathbf{x}}$ pertença à $\mathcal{K}$ e o novo vetor do resíduo seja ortogonal à $\mathcal{L}$, isto é,

$$\text{Encontre } \tilde{\mathbf{x}} \in x_0 + \mathcal{K}, \text{tal que } b - A\tilde{x} \perp \mathcal{L}. \tag{1.2}$$

Onde x_0 é um chute inicial para a solução. Matricialmente podemos observar da seguinte forma; Seja $V = [v_1, ..., v_m]$, uma matriz $n \times m$ cujos vetores coluna formam uma base de $\mathcal{K}$ e, similarmente, $W = [w_1, ..., w_m]$, uma matrix $n \times m$ cujos vetores coluna formam uma base de $\mathcal{L}$. Se a solução aproximada é escrita como

$$\tilde{x} = x_0 + Vy,$$

então a condição de ortogonalidade leva imediatamente ao seguinte sistema de equações para o vetor y:

$$W^T AVy = W^T r_0.$$

Se assumirmos que a matrix W^TAV, $m \times m$, é não-singular, obtemos a seguinte expressão para a solução aproximada $\tilde{x}$,

$$\tilde{x} = x_0 + V(W^TAV)^{-1}W^Tr_0. \tag{1.3}$$

Em geral a matrix W^TAV não precisa ser formada pois ela acaba sendo um produto direto do algoritmo, como será visto à frente. Em [1, cap. 1] são apresentadas condições de suficiência da existência da matriz $(W^TAV)^{-1}$.

1.1.1.2 Subespaços de Krylov

Seja A uma matriz tal que $A \in \mathbb{R}^n \times \mathbb{R}^n$ e o vetor $b \in \mathbb{R}^n$ chamamos de espaço de Krylov o conjunto formado por todas as combinações lineares dos vetores $(b, Ab, A^2b, ..., A^{n-1}b)$ e denotaremos como $\mathcal{K}(A, b)$. Um subespaço de Krylov será o conjunto formado pelas combinações lineares dos k vetores $(b, Ab, Ab^2, ...Ab^{k-1})$ e será denotado por $\mathcal{K}_k(A, b)$.
O seguinte teorema apresentado em [1] é uma das suas principais motivações.

Teorema 1.1 *Sejam $A \in \mathbb{C}^{m\times m}$, matriz regular, e $b \in \mathbb{C}^m$. Seja x^* a solução exata do sistema linear $Ax = b$. Seja x_0 um valor inicial para x^* e $r_0 = b - Ax_0$, o resíduo inicial. Se o polinômio mínimo do vetor r_0 relativo à A tem grau k-1, então $x^* - x_0$ pertence ao subespaço de Krylov $\mathcal{K}_{(k-1)}(A, r_0)$.*

Em [1] temos também a seguinte observação: Se estamos em busca de uma solução para $Ax = b$, o espaço natural de busca é o subespaço de Krylov gerado por A e r_0. Segundo observam Ilpsen & Meyer em [3], o polinômio mínimo de um vetor em relação a uma matriz pode ter um grau bem menor do que o polinômio mínimo da mesma matriz. E por isso, dependendo de r_0, que por sua vez depende de x_0 e do lado direito b, um método de Krylov pode convergir em um número de passos notadamente inferior ao grau do polinômio mínimo da matriz em questão.

Assim, ao utilizarmos subespaços de Krylov $\mathcal{K}_k(A, r_0)$ em (1.2) no lugar de $\mathcal{K}$, ao variarmos o $\mathcal{L}$ e usarmos diferentes formas de projeção daremos origem a distintos métodos de projeção em subespaços de Krylov.

Para termos a devida compreensão do funcionamento de um método de Krylov, iremos apresentar um método, que é amplamente utilizado e que possui diversas im-

plementações diferentes publicadas, conhecido como GMRES(Generalized Minimum Resiual) [4]. Este método é baseado no procedimento de Arnoldi [5] que será introduzido a seguir, e difere de outros métodos de Krylov como o CG(Gradiente Conjulgado) [6], por sua generalidade, não exigindo que a matriz seja simétrica ou positiva definida.

1.1.1.3 O Método de Arnoldi

O procedimento de Arnoldi é um dos pontos centrais de alguns dos métodos de Krylov como o FOM(Full Ortogonalization Method) [7] e o já citado GMRES. Este método é utilizado para ortogonalizar uma base de um subespaço de Krylov. A razão pela qual desejamos realizar esta ortogonalização é o fato da base natural para um subespaço de Krylov associado à matriz A, $\left\langle b\ Ab\ A^2b\ \ldots\ A^{k-1}\right\rangle$ não ser adequada para uma implementação numérica, como pode ser visto em [8, cap. 5] e [1, cap. 1].

Veremos uma motivação interessante apresentada em [9] originalmente formulada por Kees Vuik. Queremos construir uma base ortonormal para $\mathcal{K}_k(A, r_0)$, tal que $\mathcal{K}_k(A, r_0) = \langle v_1, v_2, ..., v_k\rangle$. Seja $V = (v_1\ v_2\ ...\ v_k)$, logo $V^H V = I$. Vale, também observar que a matriz de Krylov associada a $\mathcal{K}_k(A, r_0)$, $K_k = (r_0\ Ar_0\ ...\ A^{k-1}r_0)$, goza da seguinte propriedade:

$$
\begin{aligned}
AK_k &= (Ar_0\ A^2r_0\ \ldots\ A^kr_0) = \\
&= (Ar_0\ A^2r_0\ \ldots\ A^{k-1}r_0\ 0) + (0\ 0\ \ldots\ A^kr_0) = \\
&= K_k \begin{pmatrix} 0 & 0 & \ldots & 0 & 0 \\ 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & 0 & 0 \\ 0 & \ldots & \ldots & 1 & 0 \end{pmatrix} + A^kr_0e_k^H
\end{aligned}
\tag{1.4}
$$

onde e_k é o k-ésimo vetor da base canônica. Como buscamos uma base ortonormal, o método usual é o da fatoração $K_k = QR$, onde Q é uma matriz $m \times k$, cujas colunas são vetores ortonormais, e R é uma matriz regular triangular superior. Chamando de H_1 a matriz de Hessenberg [10] que aparece em (1.1), teremos

$$AQR = QRH_1 + A^kr_0e_k^H$$

Mais algumas contas:

$$AQ = (QRH_1 + A^k r_0 e_k^H)R^{-1} \Rightarrow Q^H AQ = (RH_1 + Q^H A^k r_0 e_k^H)R^{-1} \Rightarrow$$
$$\Rightarrow Q^H AQ = R(H_1 + R^{-1}Q^H A^k r_0 r_k^H)R^{-1},$$

ora, sendo H_2 definida da seguinte forma,

$$H_2 := H_1 + R^{-1}Q^H A^k r_0 e_k^H = \begin{pmatrix} 0 & 0 & \dots & 0 & \vdots \\ 1 & 0 & \dots & 0 & \vdots \\ 0 & 1 & \ddots & \vdots & R^{-1}Q^H A^k r_0 \\ \vdots & \vdots & \ddots & 0 & \vdots \\ 0 & \dots & \dots & 1 & \vdots \end{pmatrix}$$

é uma matriz de Hessenberg e RH_2R^{-1} também o será. E assim, vemos que a decomposição $Q^H AQ$ é uma matriz de Hessenberg superior. E, podemos tomar para V as colunas de Q. Vamos desenvolver a última coluna de RH_2R^{-1}. Ela será igual a $Q^H A^k r_0 R^{-1}(k,k)$, para isso bastando interpretar a multiplicação de matrizes como um produto externo. $R^{-1}(k,k)$ é igual a $1/\left\|\tilde{Q}(:,k)\right\|_2$, onde $\tilde{Q}(:,k)$ é o vetor $Q(:,k) = \tilde{Q}(:,k)/\left\|\tilde{Q}(:,k)\right\|_2$. E assim, comparando as últimas colunas das matrizes $Q^H AQ$ e RH_2R^-1, temos

$$Q^H AQ(:,k) = Q^H A^k r_0 / \left\|\tilde{Q}(:,k)\right\|_2,$$

ou ainda

$$A\tilde{Q}(:,k) = A^k r_0.$$

Ou seja, as projeções ortogonais de $A\tilde{Q}(:,k)$ e de $A^k r_0$ no subespaço de Krylov $\mathcal{K}_k(A, r_0)$ são as mesmas. Através do lema 1.1 concluímos portanto que se temos uma decomposição QR do subespaço $\mathcal{K}_k(A, r_0)$ podemos adicionar um novo vetor à esta base sem de fato calcular o vetor $A^k r_0$, mas utilizando $A\tilde{Q}(:,k)$ ou $AQ(:,k)R(k,k)$. Neste momento precisamos utilizar o seguinte lema.

Lema 1.1 *Como visto em [1]. Se* $v_1 = b/\left\|b\right\|_2$ *e*

$$\langle v_1, v_2, \dots, v_{j-1}, v_j \rangle = \langle v_1, v_2, , \dots, v_{j-1}, Av_{j-1} \rangle,$$

para todo $j > 1$*,então* $\langle v_1, v_2, \ldots, v_{j-1}, v_j \rangle = \langle b, Ab, \ldots, A^{j-1}b \rangle$.

Utilizando então um processo de ortogonalização como Gram-Schmidt, podemos calcular uma base ortonormal para o subespaço de Krylov sem precisar de fato calcular as vetores $Ar_0, A^2r_0, ..., Ak^{-1}$ 1.1, como iremos ver no algoritmo do método de Arnoldi que estará apoiado no lema 1.1.

Algorítmo 1.1 *Método de Arnoldi com Gram-Schmidt clássico.*

---

1. $V(:,1) = r_0/\left\|r_0\right\|$
2. *para* $j = 1 : k$
3. $\quad w = AV(:,j)$
4. $\quad H(1:j,j) = V(:,1:j)^H w$
5. $\quad w = (I - V(:,1:j)V(:,1:j)^H)w$
6. $\quad H(j+1,j) = \left\|w\right\|_2$
7. $\quad V(:,j+1) = w/H(j+1,j)$
8. *fim-para*

O algoritmo acima utiliza Gram-Schmidt como processo de ortogonalização, entretanto, este algoritmo é considerado instável como pode ser visto em [11, Cap. 5], e portanto recomenda-se o uso do Gram-Schmidt modificado [12] como pode ser visto no algoritmo 1.2, porém outros processos de ortogonalização podem ser utilizados além destes, como reflexões de Householder [13] ou rotações de Givens.

Algorítmo 1.2 *Método de Arnoldi com Gram-Schmidt modificado.*

---

1. $V(:,1) = r_0/\left\|r_0\right\|$
2. *para* $j = 1 : k$
3. $\quad w = AV(:,j)$
4. $\quad$ *para* $i = 1 : j$
5. $\quad\quad H(i,j) = (V(:,i), w)$
6. $\quad\quad w = w - H(i,j)V(:,i)$
7. $\quad$ *fim-para*
8. $\quad H(j+1,j) = \left\|w\right\|_2$
9. $\quad V(:,j+1) = w/H(j+1,j)$
10. *fim-para*

Em ambos os algoritmos não há um teste sobre $H(j, j+1)$ ser numericamente zero. Este fato, chamado de ruptura, ocorre quando o novo vetor pertence ao mesmo subespaço dos vetores gerados até aquele momento. Isto significa que o grau do polinômio mínimo de r_0 em relação a A é $j-1$. Assim, pelo teorema 1.1, teremos que $x - x0$ se encontra no subespaço gerado pelos vetores já calculados. A este fato é dado o nome de ruptura benéfica.

1.1.1.4 Generalized Minimum Residual

Em [1, Sec. 2.3] encontramos a seguinte descrição: O método do resíduo minimal generalizado é um método de projeção em subespaços de Krylov com as seguintes características. Para resolvermos $Ax = b$, partimos de um valor inicial x_0 e calculamos o resíduo inicial, $r_0 = b - Ax_0$. $\mathcal{K}_k$ será o subespaço de Krylov $\mathcal{K}_k(A, r_0)$, ou seja, $(x_k - x_0) \in \mathcal{K}_k(A, r_0)$, e o espaço de restrições será $\mathcal{L} = A\mathcal{K}_k(A, r_0)$ e, assim o GMRES assegura que o resíduo, a cada iteração, não aumentará, no pior caso o resíduo das novas iterações será igual ao da(s) anterior(es). Como a cada passo o espaço de busca está aumentando, mesmo depois de alguma estagnação, o método encontrará um ponto melhor. Um desenho esquemático da iteração do GMRES, retirado de [1], pode ser visto na Figura 1.

Já em [14] o autor destaca o fato de que este método foi desenvolvido para matrizes não simétricas e que se não for utilizada uma técnica de restart há convergência em no máximo n passos, onde n é a ordem da matriz A. Essa informação é, porém, pouco relevante, pois com o aumento do número de iterações a quantidade de armazenamento necessário para o método alcançaria números muito elevados tornando-o extremamente ineficiente, por esta razão os autores do método apresentam em [4] uma alternativa para remediar essa dificuldade prática, o GMRES(m), implementando o chamado restart, que reinicializa o método após algumas iterações, reduzindo drásticamente as dificuldades com armazenagem. No algoritmo 1.3 é apresentada a versão com restart.

Algorítmo 1.3 *GMRES(m)*

---

1. *Escolhe* x_0 *e calcula* $r_0 = b - Ax_0$ *e* $v_1 = r_0 / \|r_0\|$.

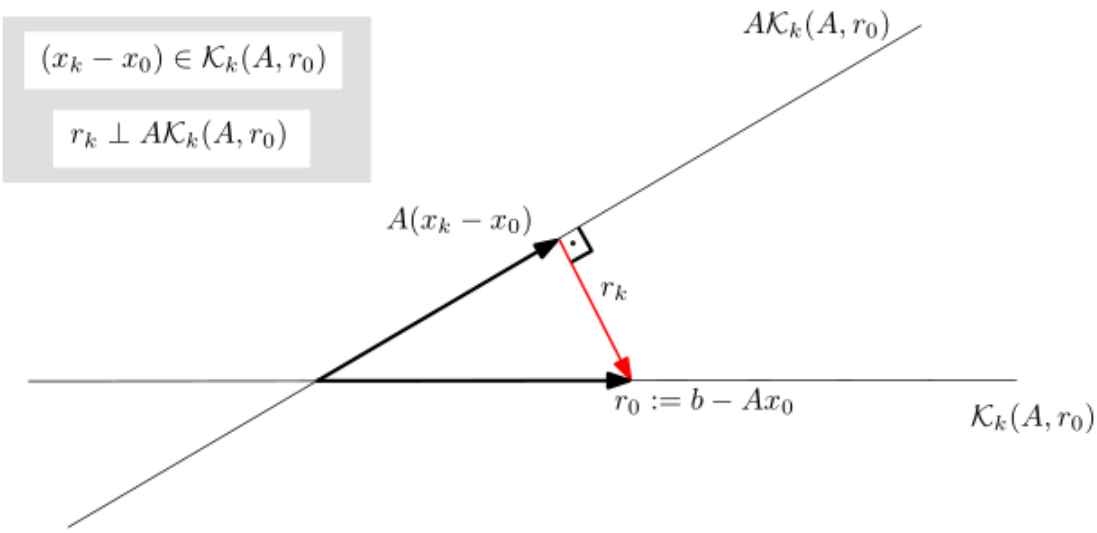


Figura 1 Representação esquemática da condição de ortogonalidade do resíduo do GMRES.

2. até convergência, faça
3. para $j = 1 : m$
4. adiciona um vetor à base V *pelo método de Arnoldi*
5. fim-para
6. calcula x_j *tal que* $x_j - x_0 \in \mathcal{K}_j(A, r_0)$ *e que* $rj \perp A\mathcal{K}_j(A, r0)$.
7. calcula $r_j = b - Ax_j$
8. se r_j *satisfaz a tolerância: finaliza*
9. do contrário, faz $x_0 = x_j$, $r_0 = r_j$ *e* $v_1 = r_j / \|r_j\|$
10. fim-até

É importante lembrar que o GMRES é apenas um entre uma grande variedade de métodos de Krylov, que é uma família de métodos iterativos entre outras. A eficiência de um método é dependente do problema e de alguns outros fatores como o precondicionador, como pode ser visto no seguinte trecho extraído de [1]. Há uma unanimidade entre os pesquisadores da área de métodos iterativos para solução de sistemas lineares: não existe o melhor método para a solução de problemas com matrizes não-simétricas [86]. Outro ponto de vista comum é o de que, para matrizes não-normais, há muito ainda o que se trabalhar na compreensão dos fatores que influenciam na convergência dos métodos. Ainda outro consenso, é o da necessidade de precondicionadores para acelerar os métodos de Krylov.

1.2 Precondicionadores

"Nada será mais central para a ciência da computação no próximo século do que a arte de transformar um problema aparentemente intratável em outro cuja a solução pode ser aproximada rapidamente. Para métodos em subespaços de Krylov, isto significa precondicionamento." Trefethen e Bau em [15]. Em [9] o autor observa que a taxa de convergência de muitos métodos iterativos para solução de sistemas lineares depende do número de condicionamento $\kappa(A)$ (definição 1.1) e/ou da distribuição dos autovalores de A. Portanto, é uma ideia natural transformar o sistema linear de tal forma que o novo sistema tenha a mesma solução (ou a solução do sistema original pode ser facilmente recuperada) e a matriz transformada tenha um número de condicionamento [16] menor e/ou uma melhor distribuição dos autovalores.

Definição 1.1 *Seja $A \in \mathbb{C}^{n\times n}$ não-singular. O número $\kappa(A) = \|A\|_p \|A^{-1}\|_p$ é o número de condicionamento relativo de A com respeito à inversão.*

O número $\kappa(A)$, definido acima, neste texto será chamado apenas de número de condicionamento de A.

Suponha uma matriz não singular M. Podemos transformar o sistema $Ax = b$ em

$$M^{-1}Ax = M^{-1}b. \tag{1.5}$$

Observa-se que a solução x de $Ax = b$ não se altera em (1.5), portanto podemos aplicar um método iterativo ao sistema precondicionado ao invés do original. A expressão (1.5) é chamada de precondicionamento pela esquerda, visto que a matriz A é multiplicada pelo precondicionador à esquerda. Há também o precondicionamento pela direita

$$AM^{-1}y = b, \quad Mx = y. \tag{1.6}$$

Neste caso a matriz A é mutiplicada pela direita. No caso onde o problema é simétrico positivo definido, podemos querer que o sistema precondicionado mantenha essa simetria, neste caso podemos aplicar o precondicionamento por ambos os lados, da seguinte forma

$$M^{-\frac{1}{2}}AM^{-\frac{1}{2}}y = M^{-\frac{1}{2}}b, \quad M^{\frac{1}{2}}x = y. \tag{1.7}$$

Em [1, cap. 3] temos a seguinte afirmação sobre a qualidade do precondicionador. Um bom precondicionador tem que acelerar a convergência do método, no mínimo, para compensar o custo de sua construção, mas o objetivo é sempre mais ambicioso. O difícil problema de se encontrar um precondicionador eficiente é que se deve identificar um operador M, linear ou não, que atenda a pelos menos, necessária mas não exclusivamente, às seguintes propriedades:

1. De alguma forma M^{-1} é uma boa aproximação para A^{-1}. Embora não haja uma teoria geral, pode-se dizer que M deve ser de tal forma que M^{-1} ou AM^{-1}, deve ser próxima da matriz identidade e que seus autovalores sejam aglomerados em uma pequena região.

2. M seja eficiente. De forma que o método precondicionado convirja muito mais rapidamente do que o não precondicionado, compenando largamente o custo de construção do precondicionador.

3. M ou M^{-1} sejam construídas em paralelo. Para explorar as várias arquiteturas que cada vez mais utilizam o processamento paralelo.

Isto nos dá uma ideia do que devemos procurar para obter um precondicionador em termos de eficiência e mostra que essa pode não ser uma tarefa fácil. A literatura na área nos mostra que existem diversos precondicionadores disponíveis e que a área de pesquisa ainda está constantemente disponibilizando novas opções. Em geral o precondicionamento é particular para cada problema, porém existem alguns precondicionadores com propostas mais genéricas (*general purpose preconditioners*), que dependem apenas de informações da matriz do sistema, como as fatorações incompletas e a inversa aproximada esparsa [2] que serão vistas mais à frente.

Uma classe particular de precondicionadores é discutida em [1], chamados de precondicionadores flexíveis, que vem apresentando desenvolvimentos nos últimos anos, onde a ideia central é o uso de diferentes precondicionadores em cada iteração. Mesmo outros métodos de Krylov podem ser usados como precondicionadores. Considere o precondicionamento pela direita como em (1.6). Uma das motivações para métodos com precondicionadores variáveis é de encontrar casos relevantes onde se necessita resolver apenas aproximadamente

$$Mz = v,$$

considerando-se que M já é uma aproximação de A. Com isso poderá haver um M diferente para cada passo k do método de Krylov, pelo menos implicitamente.

1.2.1 Fatorações Incompletas

Ao realizar a fatoração $A = LU$ surge o preenchimento do fatores L e U em posições nulas da matriz A o que significa que estes fatores serão menos esparsos do que matriz de coeficientes. Mesmo sabendo que métodos diretos não são considerados uma opção viável na solução de grandes sistemas esparsos, ao descartar parte do preenchimento no processo de fatoração podemos obter precondicionadores simples porém poderosos na forma $M = \tilde{L}\tilde{U}$ onde $\tilde{L}$ e $\tilde{U}$ são fatores LU incompletos(aproximados).

Diferentes métodos de fatoração incompleta (ILU) surgem com a escolha das regras para descartar os elementos de preenchimento. Os critérios para descarte de elementos de preenchimento podem ser baseados na sua posição, no valor ou em uma combinação de ambos. Em uma fatoração incompleta baseada em valor, podemos escolher um valor τ, chamado de tolerância de descarte, tal que um novo preenchimento é aceito somente se for maior que τ em valor absoluto. Esta é uma técnica com tolerância sobre o valor absoluto, o que pode não funcionar bem em matrizes mal escaladas, neste caso pode-se usar uma tolerância de valor relativo. Por exemplo durante a eliminação da linha i, um novo preenchimento é aceito somente se tiver um valor absoluto maior que $\tau\|a_i\|_2$, onde a_i é a i-ésima linha de A.

1.2.2 Inversa Aproximada Esparsa

Técnicas de inversa aproximada esparsa buscam encontrar, a partir de uma matriz esparsa A, uma matriz esparsa M que seja, em algum sentido, uma boa aproximação de A^{-1}. Entretanto, isso não é simples, visto que é comum que a inversa de uma matriz esparsa seja densa. Existem diversos algoritmos diferentes que se propõe a calcular esta inversa aproximada partindo de princípios diversos.

Uma classe de métodos de inversa aproximada é a de minimização da norma de Frobenius, cuja ideia básica é calcular $M \approx A^{-1}$ como a solução do problema de mini-

mização

$$\min_{M \in \mathcal{S}} \|I - AM\|_F,$$

onde $\mathcal{S}$ é um conjunto de matrizes esparsas e $\|.\|_F$ denota a norma de frobenius da matriz. Assim

$$\|I - AM\|_F^2 = \sum_{j=1}^{n} \|e_j - Am_j\|_2^2,$$

onde e_j corresponde à j-ésima coluna da matriz identidade, o cálculo de M reduz-se à solução de n problemas de mínimos quadrados independentes, sujeito às restrições de esparsidade. Essa independência dos problemas é um dos principais atrativos desta classe de métodos de inversa aproximada pois permite que o algoritmo seja extremamente paralelizável.

1.2.3 Decomposição de domínio

Técnicas de decomposição de domínio [9] consistem basicamente na divisão do problema em diversos subproblemas menores, que podem ser distribuídos e resolvidos paralelamente, e na união das soluções de acordo com a técnica utilizada para encontrar a solução global. Atualmente a técnica de decomposição de domínios é utilizada para introduzir um forte paralelismo na construção de precondicionadores para métodos de Krylov. Existe uma numorosa quantidade de métodos de decomposição de domínio que podem ser divididos em duas principais categorias; com e sem sobreposição.

Esta técnica é em geral utilizada para sistemas lineares obtidos da discretização de EDPs. Seu nome está intimamente ligado ao particionamento do domínio da EDP a ser resolvida. Portanto os métodos de decomposição de domínio nem sempre podem ser vistos de um ponto de vista puramente algébrico.

1.2.4 Reordenamento

Sejam P e Q matrizes de permutação o sistema linear $Ax = b$ terá sido reordenado ao ser substituído pelo equivalente

$$PAQy = Pb, \; x = Qy. \tag{1.8}$$

Reordenamentos [17] são utilizados para reduzir preenchimento (em solvers diretos

esparsos), para introduzir paralelismo na construção e na aplicação de precondicionadores (ILU), e para melhorar a estabilidade de fatorações incompletas. Em muitos casos, reordenamentos afetam a taxa de convegência de métodos de Krylov precondicionados.

2 JINSOL

Neste capítulo iremos fazer uma descrição do programa JinSol, datalhando seu uso e alguns aspectos do seu desenvolvimento. Primeiramente será apresentada uma versão inicial de uma ferramenta que foi sua precursora. Em seguida será feito um detalhamento do programa em suas principais funções para preparação e execução de testes e análise de dados.

2.1 SaddleOO

A atual proposta é uma reformulação e um grande avanço sobre o trabalho apresentando em [18], que foi desenvolvido inteiramente em MATLAB. A ideia que contempla o JinSol surgiu no projeto "Solução de Sistemas Lineares de grande porte com aplicações" com a observação de que uma ferramenta de auxílio na pesquisa de soluções de sistemas lineares tornaria o processo mais eficiente. Neste ambiente uma primeira versão do programa foi concebida com foco em problemas de ponto de sela [19].

O problema de ponto de sela se caracteriza por sua matriz de coeficientes em blocos 2×2 apresentada abaixo

$$\begin{bmatrix} \mathbf{A} & \mathbf{G} \\ \mathbf{D} & -\mathbf{C} \end{bmatrix} \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{g} \end{bmatrix} \quad \text{ou} \quad \mathcal{A}\boldsymbol{x} = \boldsymbol{b}, \tag{2.1}$$

$$\mathbf{A} \in \mathbb{R}^{n\times n}, \quad \mathbf{G}^T, \mathbf{D} \in \mathbb{R}^{m\times n}, \quad \mathbf{C} \in \mathbb{R}^{m\times m} \quad \text{com} \quad m \geq n. \tag{2.2}$$

Um problema de ponto de sela(generalizado) satisfaz uma ou mais das seguintes condições:

1. $\mathbf{A}$ é simétrica: $\mathbf{A} = \mathbf{A}^T$;
2. A parte simétrica de $\mathbf{A}$, $H = \frac{1}{2}(\mathbf{A} + \mathbf{A}^T)$ é positiva semidefinida;
3. $\mathbf{G}^T = \mathbf{D}$;
4. $\mathbf{C}$ é simétrico ($\mathbf{C} = \mathbf{C}^T$) e positiva semidefinida;
5. $\mathbf{C} = \mathbf{0}$ ($\mathbf{C}$ é uma matriz identicamente nula);

Esta primeira versão foi chamada de SaddleOO. Na Figura 2 temos a tela inicial da interface com uma matriz carregada. Na aba Graphic Analisys, na parte direita da interface, o usuário pode visualizar a estrutura de esparsidade da matriz carregada, bloco à bloco. Reordenamentos podem ser aplicados à matriz para análise das mudanças no seu padrão de esparsidade em consequência do reordenamento.

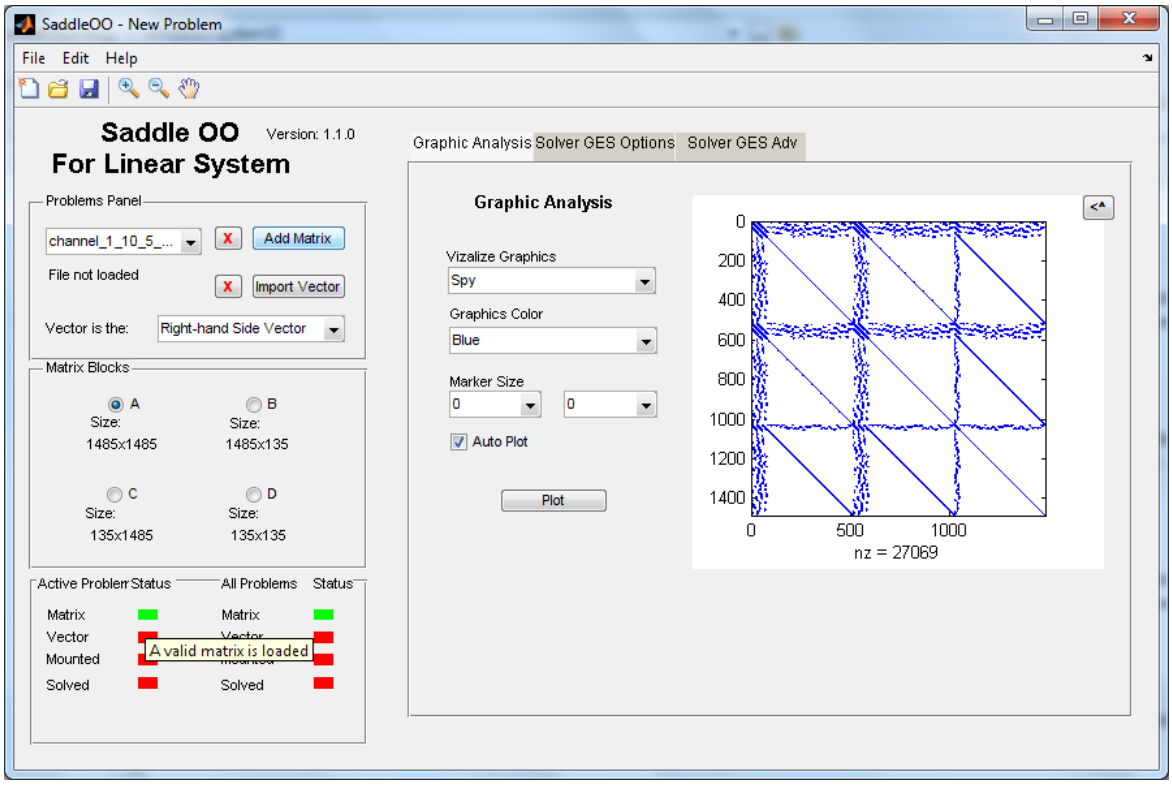


Figura 2 SaddleOO - tela inicial.

Ao lado esquerdo o usuário poderá carrega a matriz A e o lado direito b do sistema $Ax = b$. Na aba Solver Ges Options são oferecidas 4 opções de métodos de Krylov para resolução do sistema e outros quatro grupos de opções que definem um precondicionador flexível específico para problemas de ponto de sela Figura 3. O usuário pode selecionar quantas opções desejar de cada grupo, o programa irá solucionar o sistema com cada combinação possível das opções selecionadas. Esta rotina de testes deriva do trabalho de Wagner Fortes [20]. Após a execução da rotina um relatório com as informações da solução pode ser gerado em formato de planilha .xls, ou em tex, com uma tabela para ser compilada para pdf em LaTeX. A terceira aba, Solver Ges Adv permite a alteração de parâmetros do método de Krylov, tais como número máximo de iterações e tolerância.

Esta primeira versão foi uma fase muito importante para o surgimento do JinSol, onde localizamos pontos de dificuldade na execução do projeto e as ferramentas que

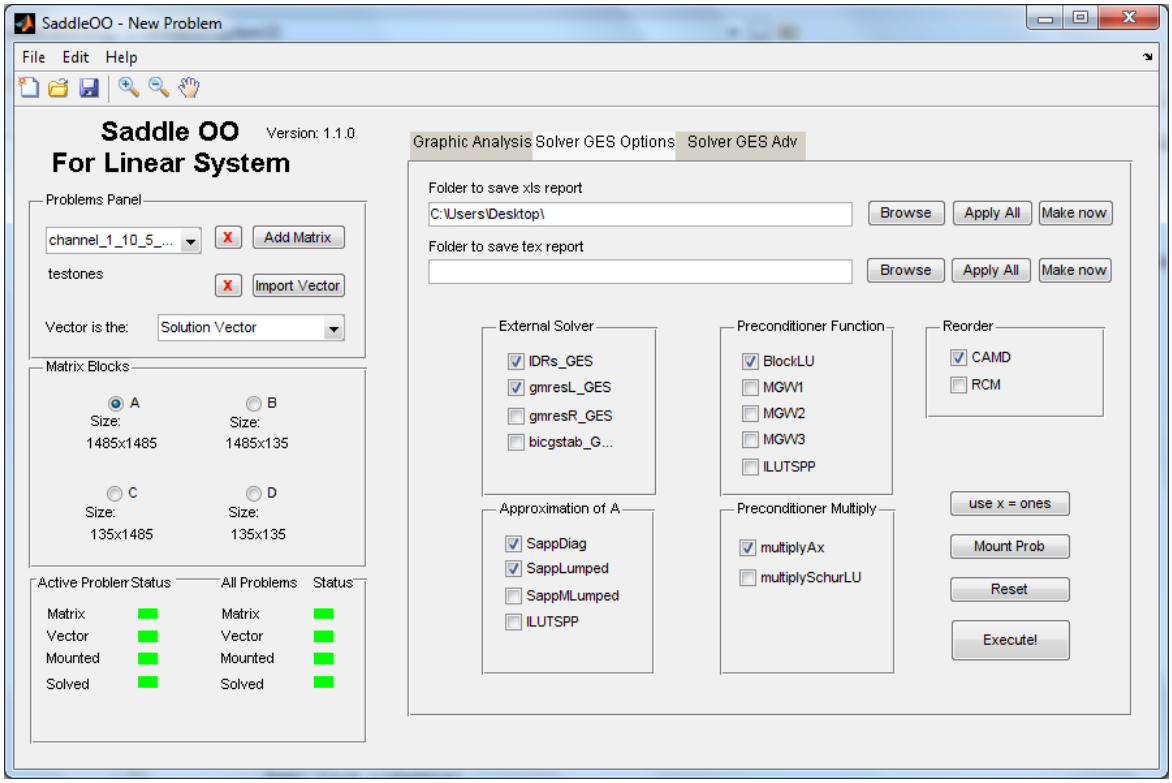


Figura 3 SaddleOO - aba Solver Ges Options.

precisaríamos para nos aproximarmos do objetivo, entre elas:

1. Uma proposta mais genérica para sistemas lineares.

   Os problemas de ponto de sela são apenas uma classe de sistemas lineares não tão representativa. Portanto é importante uma ferramenta mais geral para sistemas lineares que inclua, entre outros, o problema de ponto de sela.

2. Interface gráfica em uma plataforma mais especializada.

   Os recursos para interface gráfica e o uso de orientação a objeto em Matlab se mostraram muito restritivos e ineficientes para a escala do projeto.

3. Integração de diferentes ferramentas disponíveis em várias linguagens.

   O uso do Matlab para resolução dos sistemas também se mostra restritivo pois seu processamento não alcança a eficácia, em termos de velocidade, de rotinas desenvolvidas em outras linguagens, como C ou Fortran, portanto é importante que o programa consiga utilizar ferramentas externas.

4. Possibilidade do usuário adicionar seus métodos ao programa.

   Este aspecto é importante para que o programa possa ser usado no desenvolvimento

de novas técnicas de resolução de sistemas lineares e também para que o usuário possa comparar seus próprios métodos com os métodos oferecidos na solução de um dado problema.

2.2 Apresentação do JinSol

Levando em consideração os pontos que se destacaram na experiência do SaddleOO foi iniciado o desenvolvimento de uma nova ferramenta. Assim surge o JinSol, que se apresenta como um programa com interface gráfica em Java para realização de testes de solução de sistemas lineares em geral, com ferramentas de análise do problema e de resultados, integrada ao Matlab e com proposta de integração à outros programas.

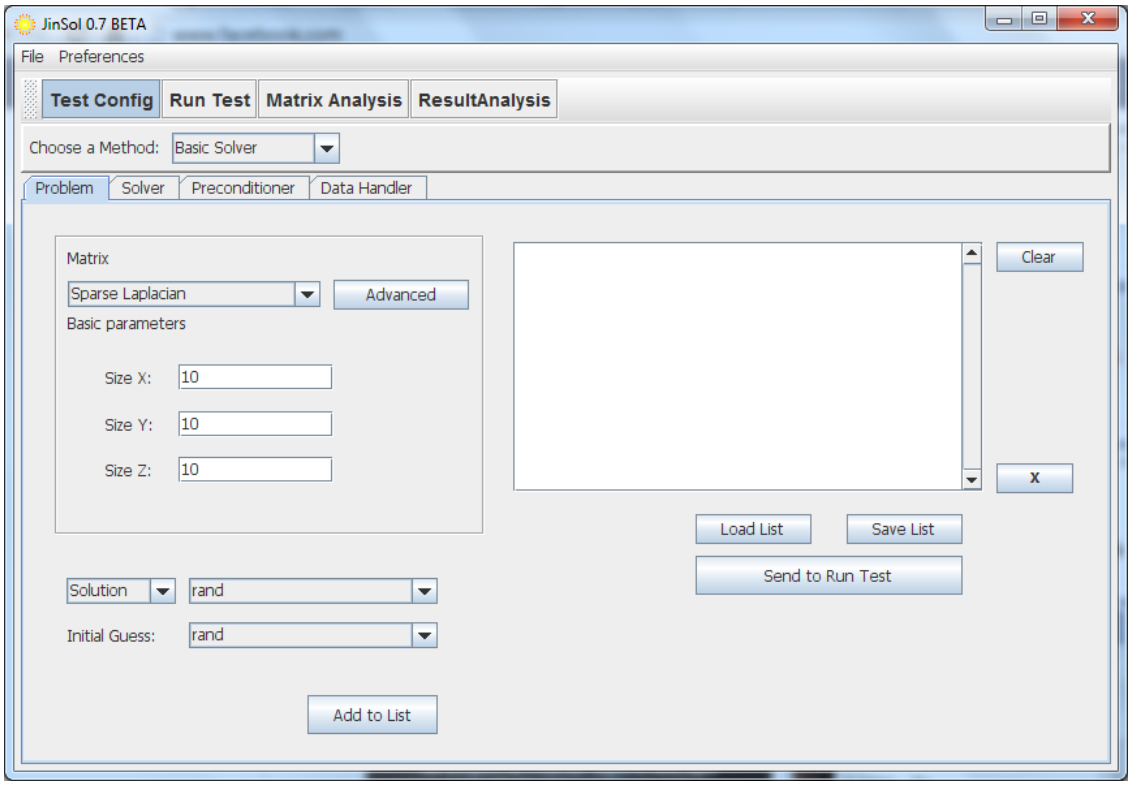


Figura 4 JinSol - tela inicial.

Na Figura 4 podemos ver a interface JinSol como é apresentada ao ser iniciada. Logo abaixo da barra de menus temos a barra de modos. O JinSol possui 4 diferentes modos sendo eles:

1. Configuração de testes(Test Config).

   Neste modo o usuário poderá configurar baterias de testes com múltiplos problemas, métodos e precondicionadores.

2. Execução de Testes(Run Tests).
   Neste modo o usuário poderá fazer os últimos ajustes e executar as baterias de testes utilizando as configurações realizadas no modo de Configuração de Testes.

3. Análise de Matriz(Matrix Analysis)
   Neste modo o usuário poderá recuperar informações variadas sobre matrizes em geral, incluindo informações gráficas.

4. Análise de Resultados(Result Analysis)
   Neste modo o usuário poderá recuperar informações variadas sobre os dados gerados durante os testes, inclusive com gráficos.

Nas próximas páginas serão descritos cada um dos modos do JinSol.

2.2.1 Configuração de testes

Para configuração de baterias de testes estão disponíveis, na versão atual, três métodos de solução.

1. Basic Solver

2. Solver Ges

3. Presol Driver Test

O primeiro item trata-se da resolução do sistemas do tipo $Ax = b$ através de um método iterativo, com a escolha de um reordenamento e um precondicionador de uso geral(general purpuse preconditioner). O usuário irá configurar uma lista de problemas, outra de *solvers* e uma última de precondicionamento. Como pode ser visto na Figura 4, há 4 abas, 3 referentes aos itens citados e uma última de manuseio dos dados obtidos nos testes. Ainda nesta imagem podemos ver uma área para escolha da matriz A do sistema. Neste painel é possível obter uma matriz a partir de uma das rotinas de criação de matrizes disponíveis, geradas a partir da especificação de alguns parâmetros ou carregar matrizes nos formatos disponíveis.

Assim como será observado em outros pontos do programa, alguns parâmetros avançados poderão ser escolhidos apertando o botão *advanced* que abrirá uma caixa de diálogo, que pode ser visto na Figura 5. Uma vez escolhida a matriz o usuário poderá

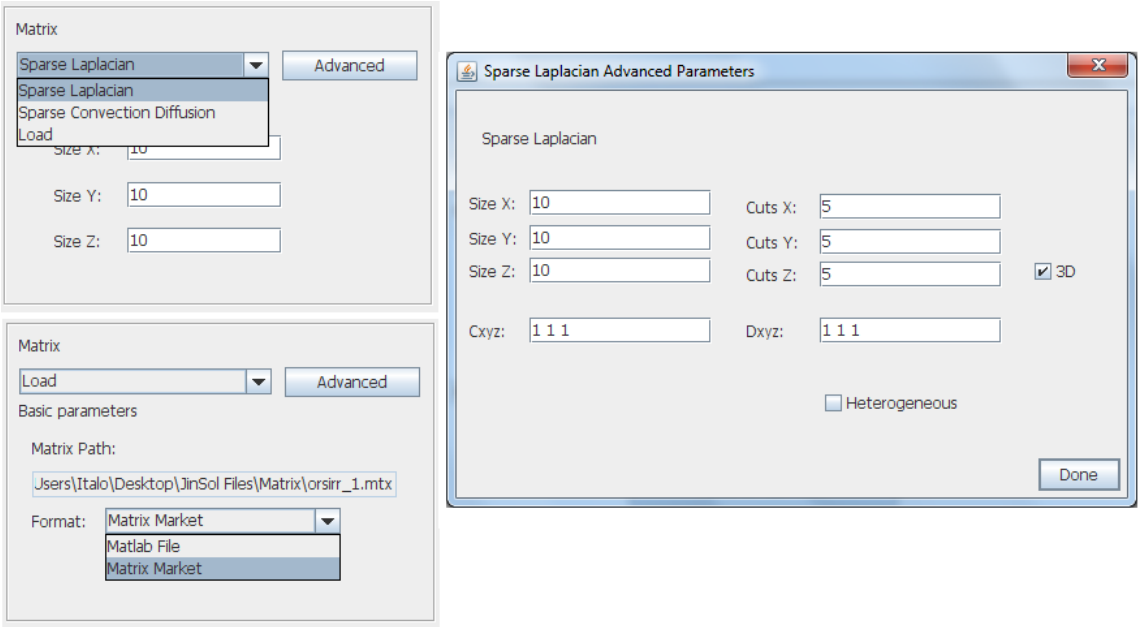


Figura 5 Matrix Creator - Advanced panel

escolher um lado direito b ou uma solução x^*, de onde será gerado $b = Ax^*$, e um chute inicial x_0 para o método iterativo. Ao clicar no botão *Add to List* o problema composto pelos itens selecionados será adicionado à lista à direita na Figura 4. Uma vez configurada a lista de problemas, ela poderá ser salva em arquivo ou enviada para a área de execução de testes, que será vista na próxima subseção. Listas salvas em arquivos podem ser carregadas a qualquer momento pelo usuário.

Na aba *Solver* (Figura 6) é configurada a lista de *solvers*, ou seja, métodos iterativos que são aplicados ao sistema linear. Algumas opções estão disponíveis para serem adicionadas à lista com os parâmetros escolhidos, de forma semelhante ao processo realizado na lista de problemas. No caso do GMRES estes parâmetros tratam-se do restart, número máximo de iterações(Max of iterations) e tolerância(tolerance).

Na aba *Preconditioners*(Figura 7), analogamente às outras apresentadas acima, será organizada uma lista de precondicionamento do problema. Para cada item nesta lista o usuário escolherá um reordenamento definido pelas matrizes de permutação P e Q onde o sistema original será substituído por

$$\bar{A}y = \bar{b}, \; x = Qy \tag{2.3}$$

onde $\bar{A} = PAQ$ e $\bar{b} = Pb$.

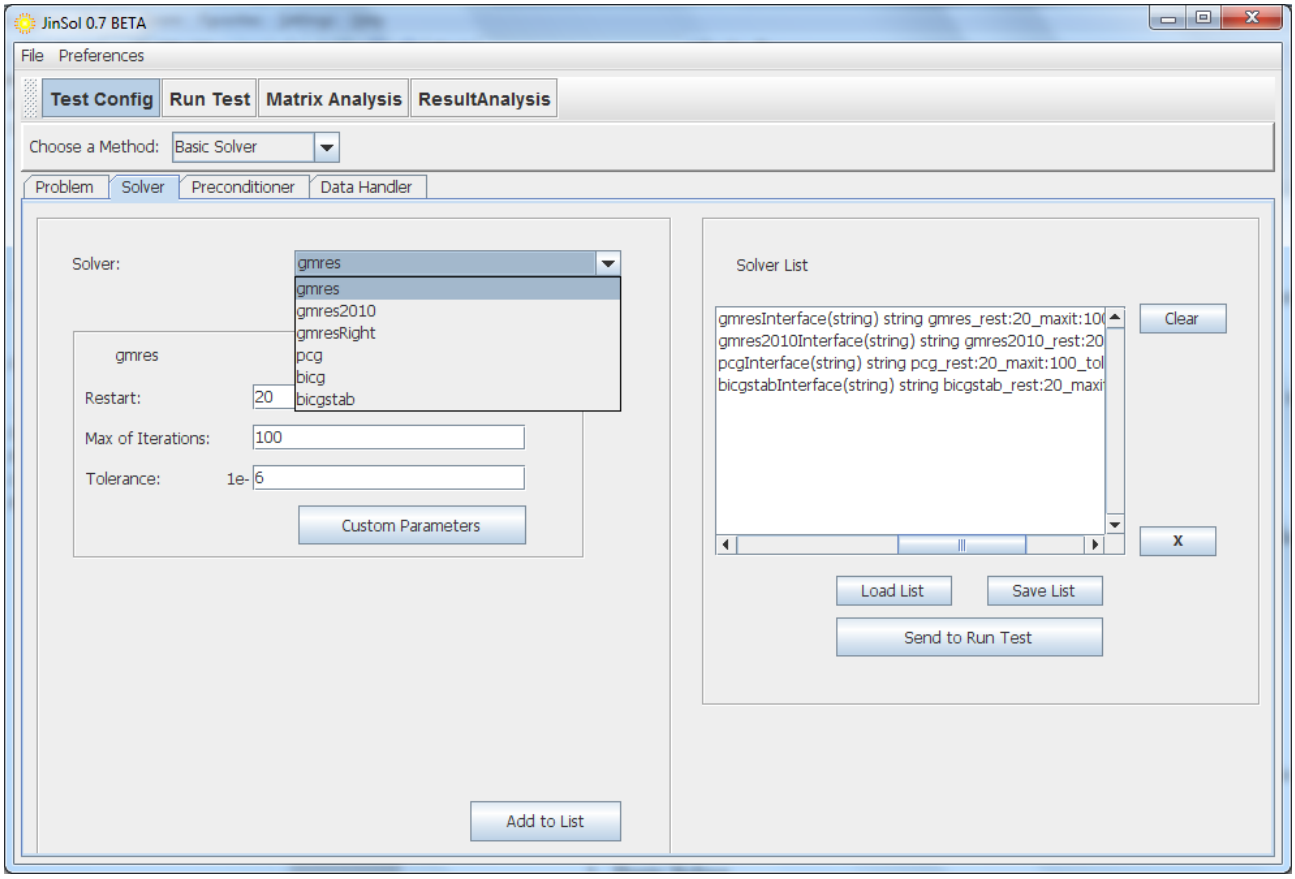


Figura 6 JinSol - aba *Solver*.

Além do reordenamento será também escolhido um precondicionador M que será construído sobre $\bar{A}$ e aplicado sobre o sistema (2.3). Algumas opções podem ser observadas à esquerda na Figura 8. À direita é apresentada a caixa de diálogo para opções avançadas do precondicionador para que o usuário possa ajustar os parâmetros do precondicionador.

Assim sendo, dado um problema $Ax = b$ da lista de problemas, tomamos um item da lista de precondicionamento, o sistema linear será substituído por $\bar{A}y = \bar{b},\ x = Qy$ relativo ao reordenamento deste item, e será então precondicionado. Obteremos então $M^{-1}\bar{A}y = M^{-1}\bar{b},\ x = Qy$ que será resolvido por um item da lista de *solvers*. Desta forma o número de testes que será realizado será o produto do número de itens em cada uma das 3 listas. Os dados gerados por estes testes poderão ser armazenados segundo as opções configuradas na ultima aba, *Data Handler*(Figura 9).

O painel *File Options* será utilizado para as opções de caminho e nome dos arquivos gerados durante os testes realizados. O usuário pode escolher um prefixo para o nome dos arquivos e pode escolher colocar a data e hora do teste executado no nome do arquivo. Também é possível escolher se arquivos de mesmo nome devem ser sobrescritos

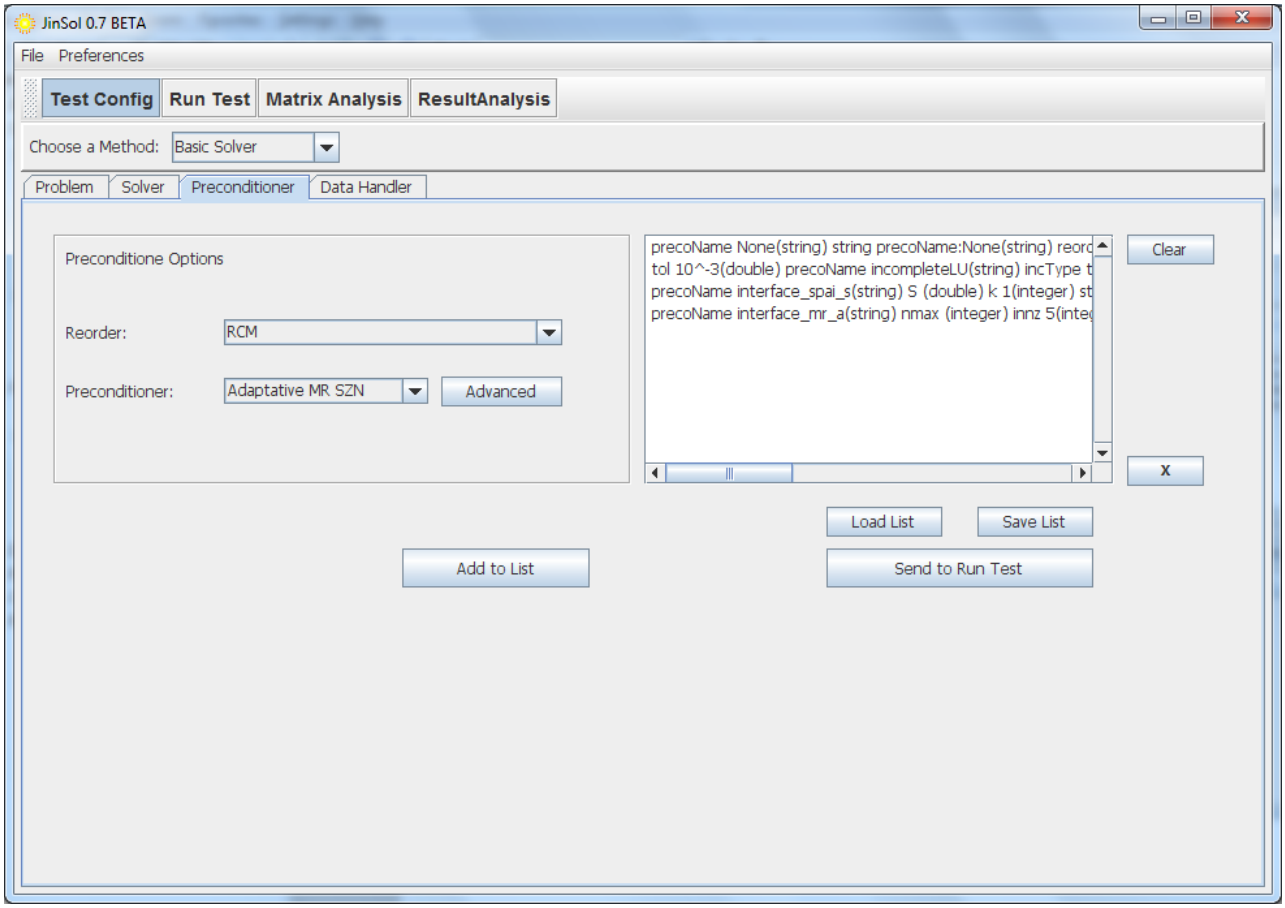


Figura 7 JinSol - aba *Preconditioner*.

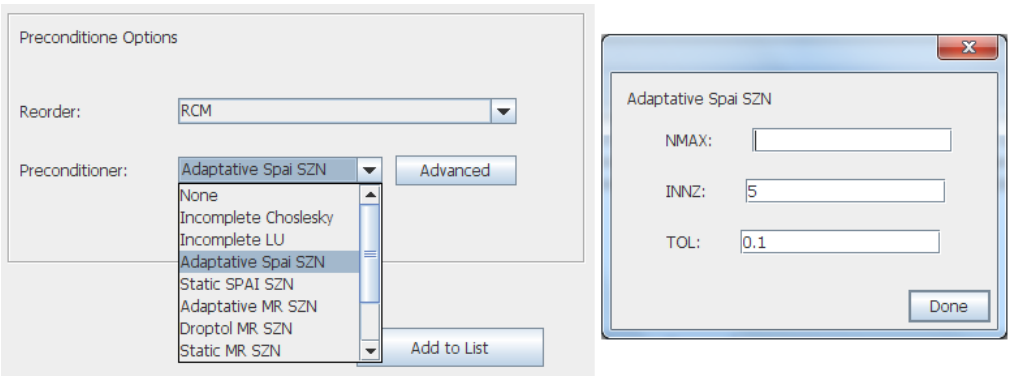


Figura 8 Lista de precondicionadores - Caixa de diálogo avançado.

ou anexados. No painel de opções pode ser escolhido qual o formato do relatório gerado e quais dados gerados durante o teste devem ser armazenados em disco.

O demais métodos de solução seguem um padrão semelhante na configuração dos testes. No Solver Ges o problema será resolvido com uma matriz de ponto de sela, e por isso é necessário informar a divisão dos blocos a partir da ordem do bloco $\mathbf{A}$ na equação (2.1). O método utilizado no Solver Ges é o mesmo apresentado no programa SaddleOO,

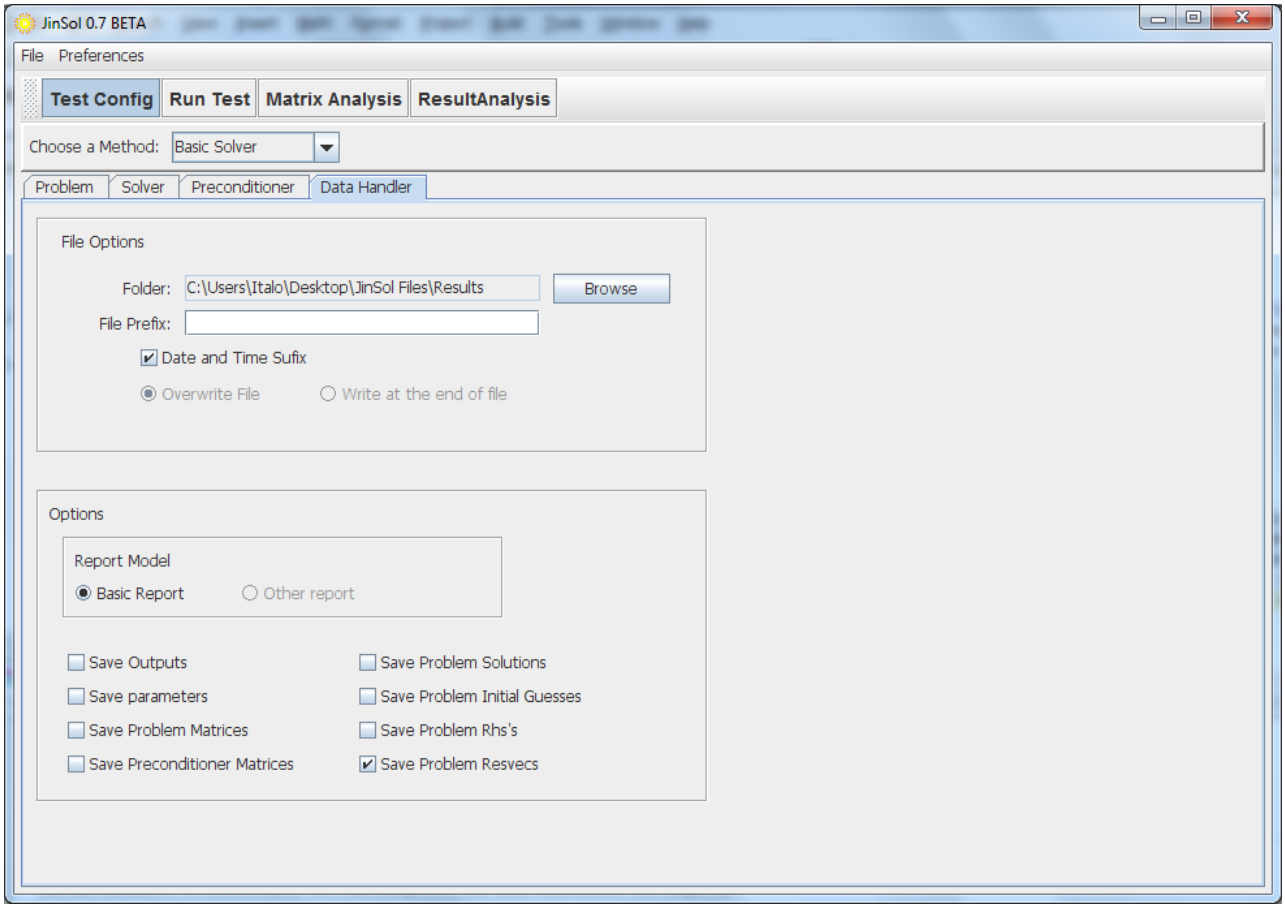


Figura 9 JinSol - aba *Data handler*.

o JinSol porém permite a configuração de diversos parâmetros através da interface, os quais o usuário não tinha acesso anteriormente.

O Presol Driver Test utiliza uma família de precondicionadores que utilizam técnicas de decomposição do domínio, e por isso o problema deve conter informações de domínio que devem ser informadas pelo usuário ou obtidos através de metadados associados às matrizes de coeficientes carregadas.

2.2.2 Execução de Testes

No modo de execução de testes(Run Tests) o usuário irá carregar suas listas de problemas, *solvers* e precondicionadores relativos ao método que deseja executar os testes. Na Figura 10 podem ser observadas as listas carregadas. O botão *Count Tests* irá fornecer o número total de testes que serão realizados a partir das listas. Os conjuntos de listas podem também ser salvos como arquivos de execução de testes. Ao clicar no botão *Run Tests* os testes serão executados.

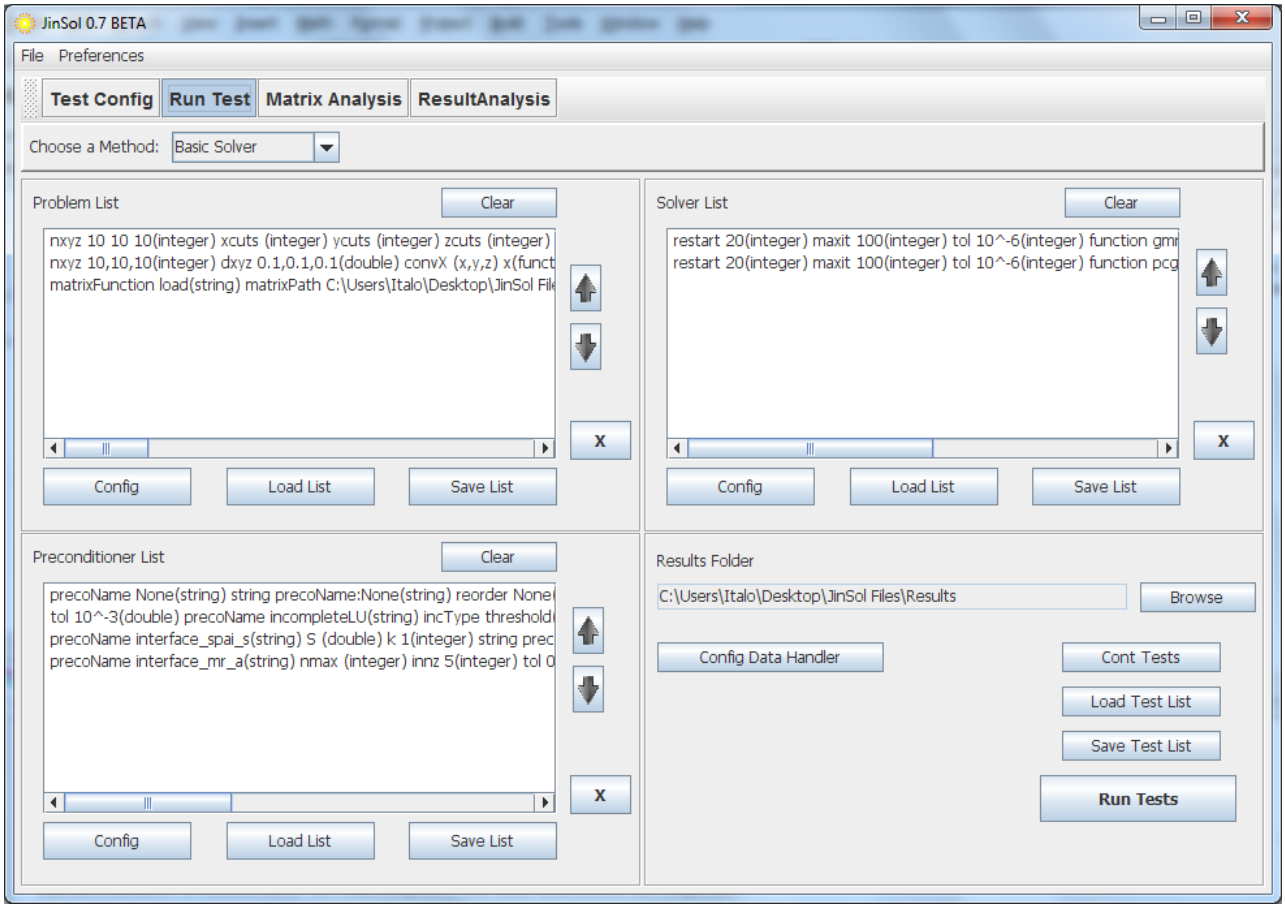


Figura 10 JinSol - modo de execução de testes

Atualmente todas as rotinas de testes são executadas no Matlab, que é controlado remotamente pela interface. Caso a interface não esteja conectada à alguma instância aberta do programa, ela é aberta no momento de executar uma rotina e é mantida aberta até o fechamento do JinSol ou até o usuário encerrá-la através da opção na barra de menu(Figura 11).

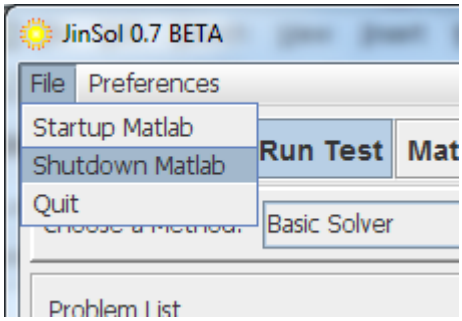


Figura 11 JinSol - Menu File.

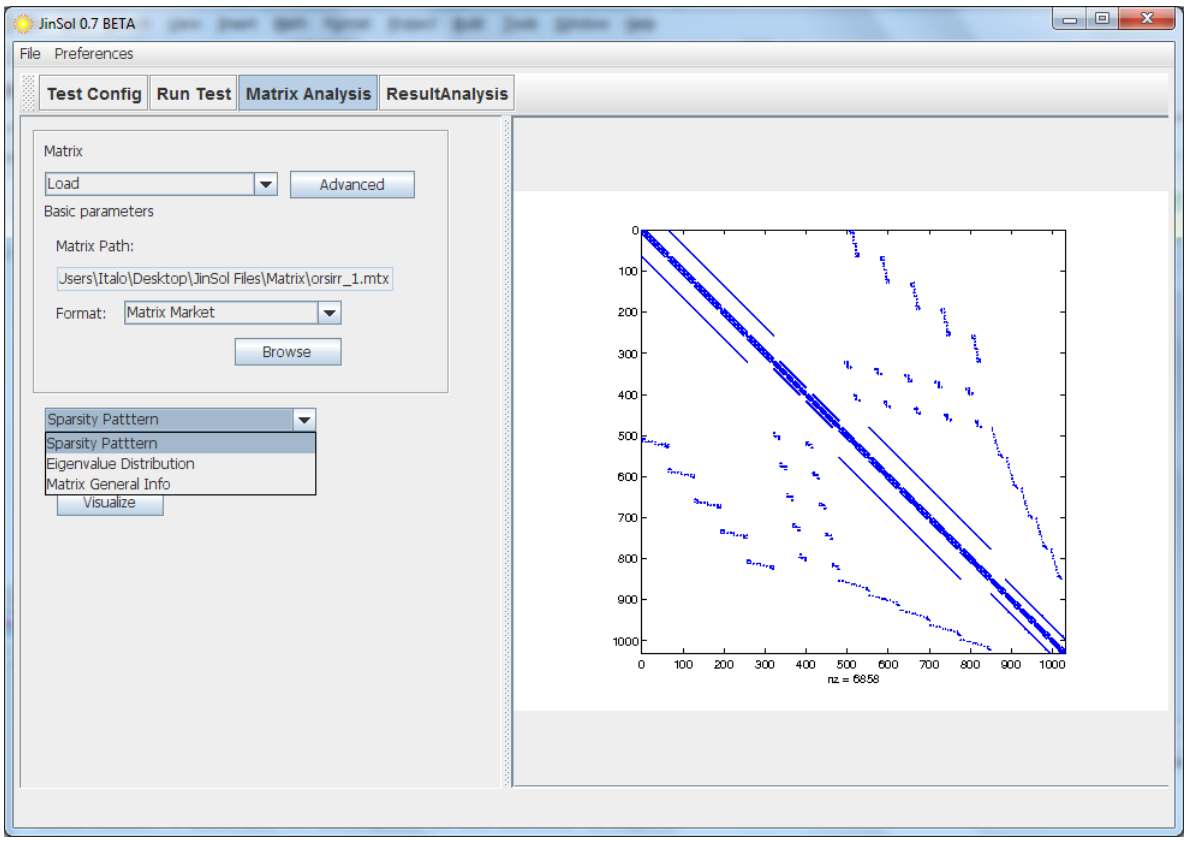


Figura 12 JinSol - Matrix Analysis.

2.2.3 Análise de Matriz

O modo de análise de matriz(Matrix Analysis) é dedicado ao estudo das propriedades da matriz de coeficientes. Informações como o gráfico do padrão de esparsidade, autovalores e número de condicionamento da matriz podem ser obtidos utilizando as ferramentas disponíveis nesta área do JinSol. A análise de informações da matriz de coeficientes pode ajudar a prever o comportamento de determinadas técnicas de solução ou mesmo compreender os resultados obtidos.

Na Figura 12 podemos ver na esquerda novamente o painel de matriz descrito acima, onde será escolhida a matriz a ser analisada. Logo abaixo o usuário escolherá quais dados deseja visualizar. Na mesma imagem podemos ver à direita a visualização do padrão de esparsidade de uma matriz carregada no formato mtx.

O usuário pode também visualizar a distribuição dos autovalores da matriz no plano complexo(Figura 13). A interface disponibiliza as opções de quantos e quais autovalores devem ser calculados(em valor percentual ou absoluto).

O programa também pode gerar uma lista com diversas informações sobre a matriz,

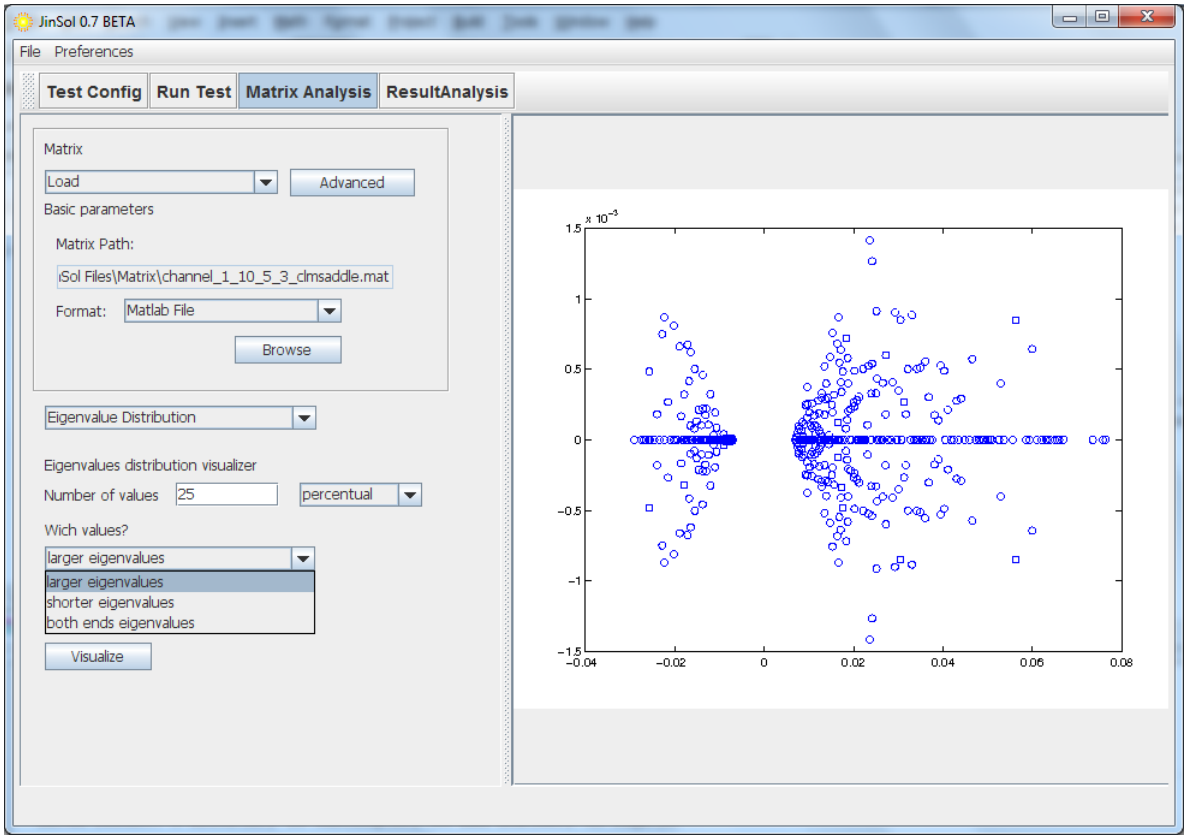


Figura 13 JinSol - Distribuição de autovalores.

incluindo norma e número de precondicionamento (Figura 14).

2.2.4 Análise de Resultados

Ao final da execução de testes o programa oferece a opção de mostrar automaticamente o relatório em formato de texto na área de análise de resultados (Figura 15). O relatório em formato de texto permite uma visualização rápida de dados importantes obtidos na resolução dos problemas. Além do relatório de texto, o programa gera também um relatório em formado csv(comma separated values) que pode ser visualizado em programas de gerenciamento de planilhas, que permitem que o usuário organize os dados a serem analisados da forma que achar mais conveniente para o estudo das informações.

Caso o usuário tenha optado por armazenar os vetores de decaimento do resíduo, o JinSol possui uma interface para visualizá-los graficamente nos testes de sua escolha. A análise do decaimento do resíduo pode ajudar a detectar alguns comportamentos do método de solução como pontos de estagnação, que podem ser analisados e tratados na melhoria do método ou do precondicionamento. Um exemplo pode ser visto na Figura

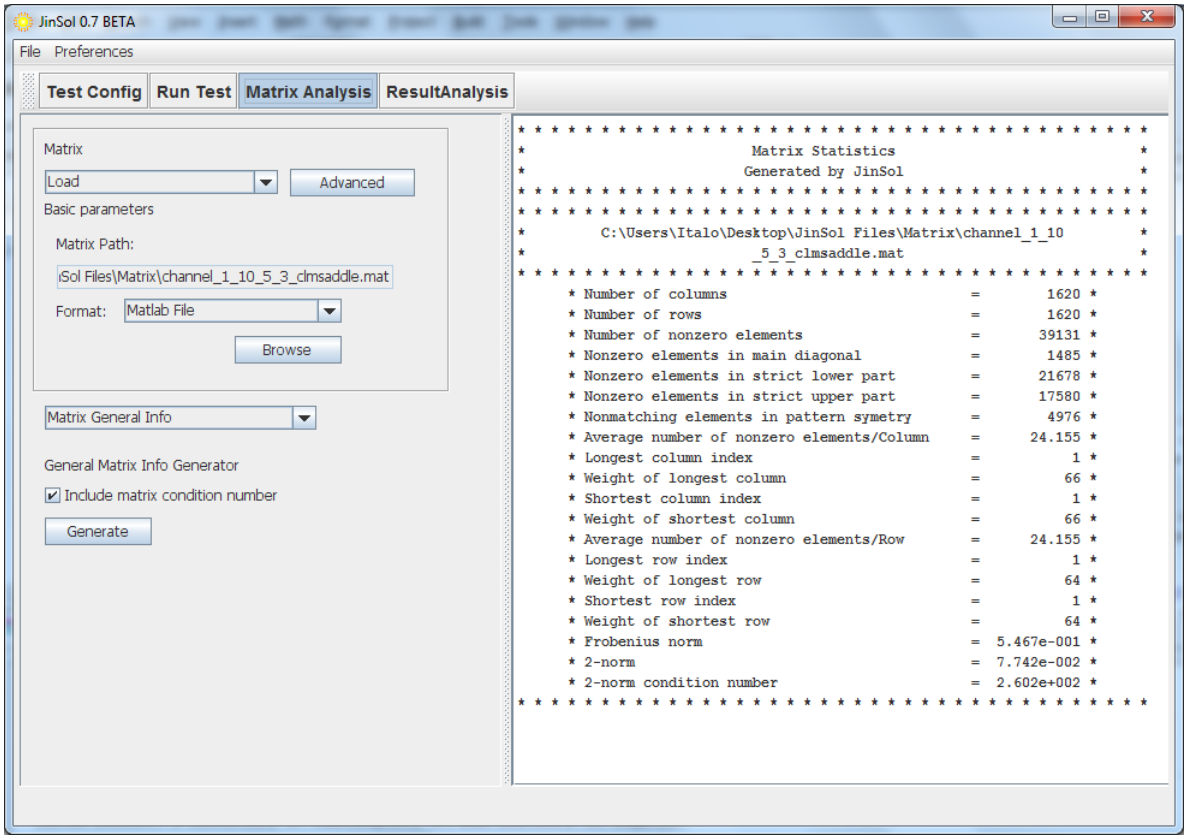


Figura 14 JinSol - Distribuição de autovalores.

16.

Afim de realizar um teste sobre o uso da interface iremos propor um caso de uso de alguns métodos e precondicionadores contidos no JinSol sobre alguns problemas que serão descritos no capítulo seguinte. Em seguida iremos apresentar os resultados obtidos nestes testes através do programa e analisar aspectos do seu uso.

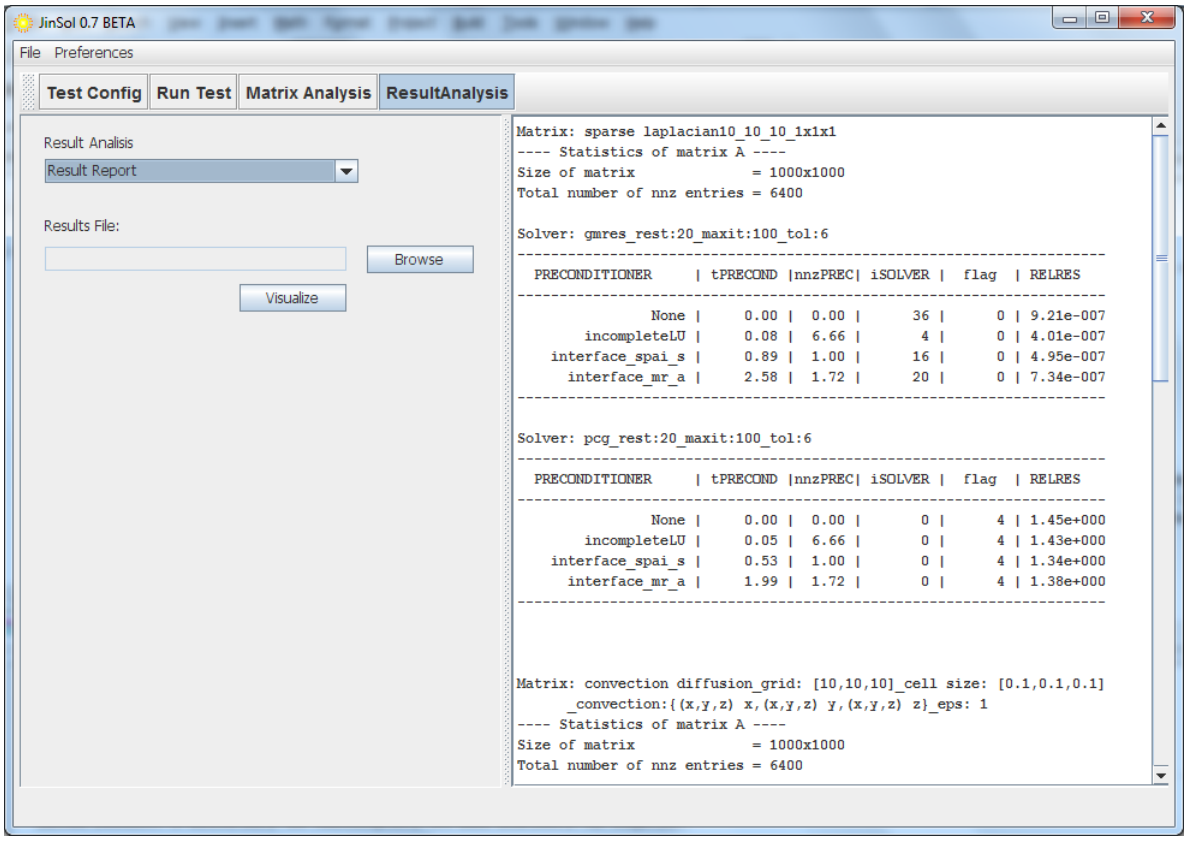


Figura 15 JinSol - Result Analysis.

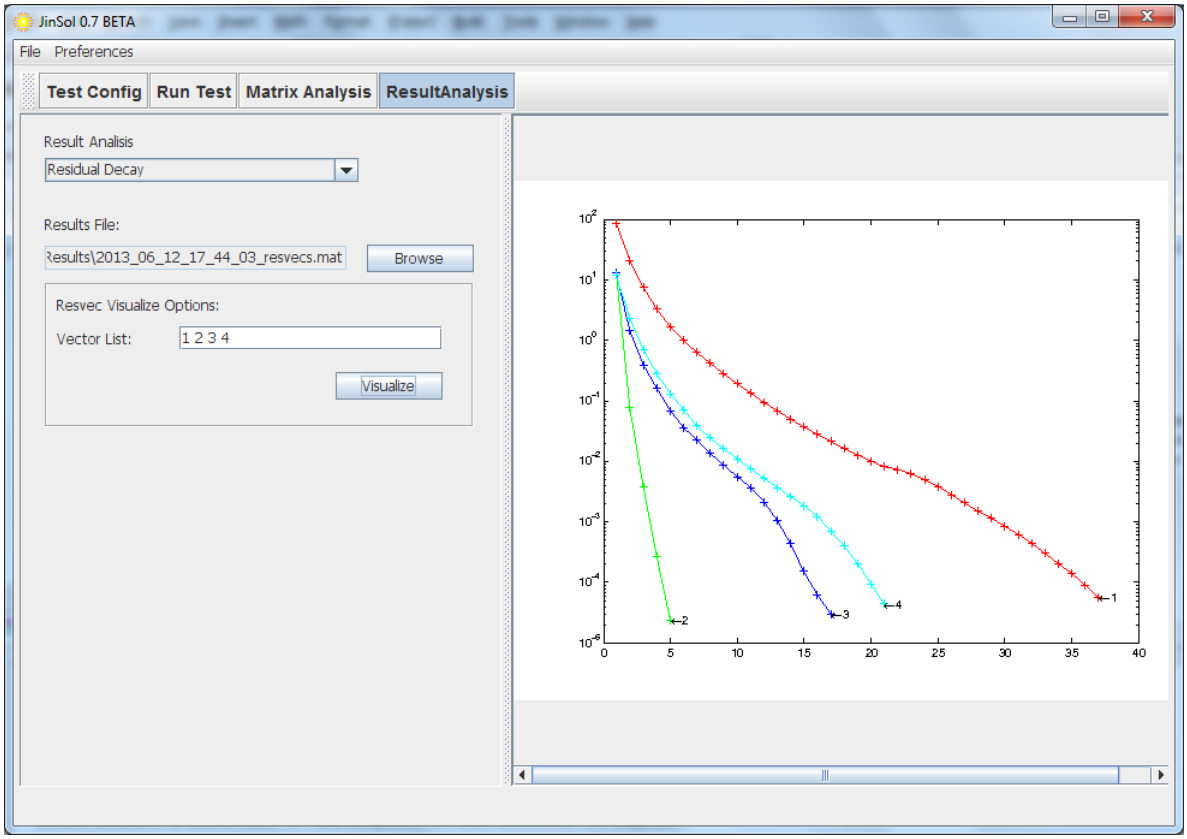


Figura 16 JinSol - Result Analysis.

3 DESCRIÇÃO DOS TESTES

Neste capítulo iremos propor alguns testes sobre problemas distintos utilizando o JinSol. A proposta é fazer um caso de uso para cada método contido no programa utilizando as múltiplas opções disponíveis.

3.1 Presol Driver Test

Usaremos neste teste uma matriz associada à modelagem tridimensional de escoamento monofásico incompressível em um meio poroso com discretização de diferenças finitas [21] do operador $\nabla \cdot K\nabla$, impondo condições de contorno de Dirichlet. O JinSol possui agregado o algoritmo de construção destas matrizes onde o usuário poderá fornecer os parâmetros definindo o tamanho da malha, o tamanho das células e o coeficiente de anisotropia. Também é possível introduzir um fator de heterogeneidade. Para este pro-

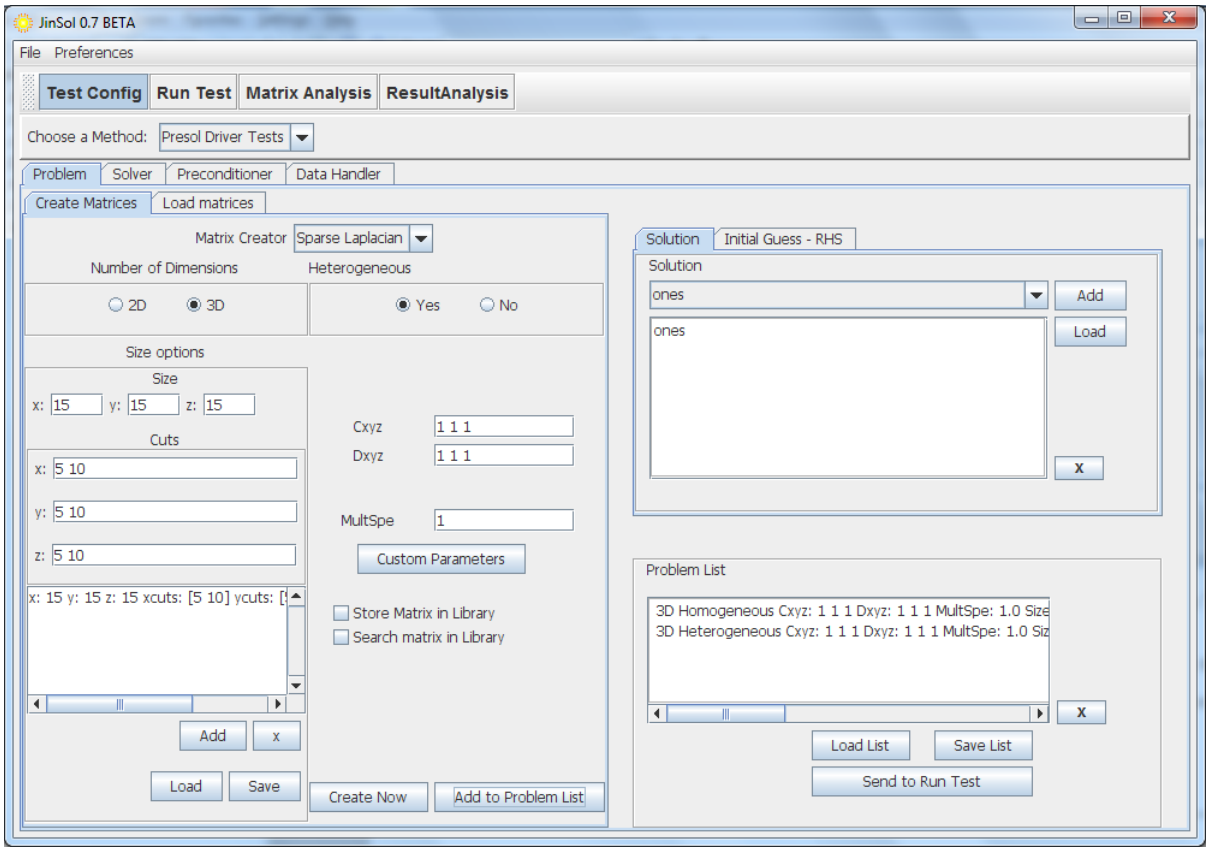


Figura 17 JinSol - Configuração do problema no método Presol Driver Test

blema usaremos o método Presol Driver Test do JinSol, precondicionado com técnicas de decomposição de domínios. Usaremos o tamanho da malha de $15 \times 15 \times 15$ com células

1x1x1 e coeficiente de anisotropia igual a 1. Dividiremos os domínio em subdomínios de $5 \times 5 \times 5$. A Configuração do problema na interface pode ser vista na Figura 17.

O *solver* utilizado é GMRES cuja configuração pode ser vista na Figura 18

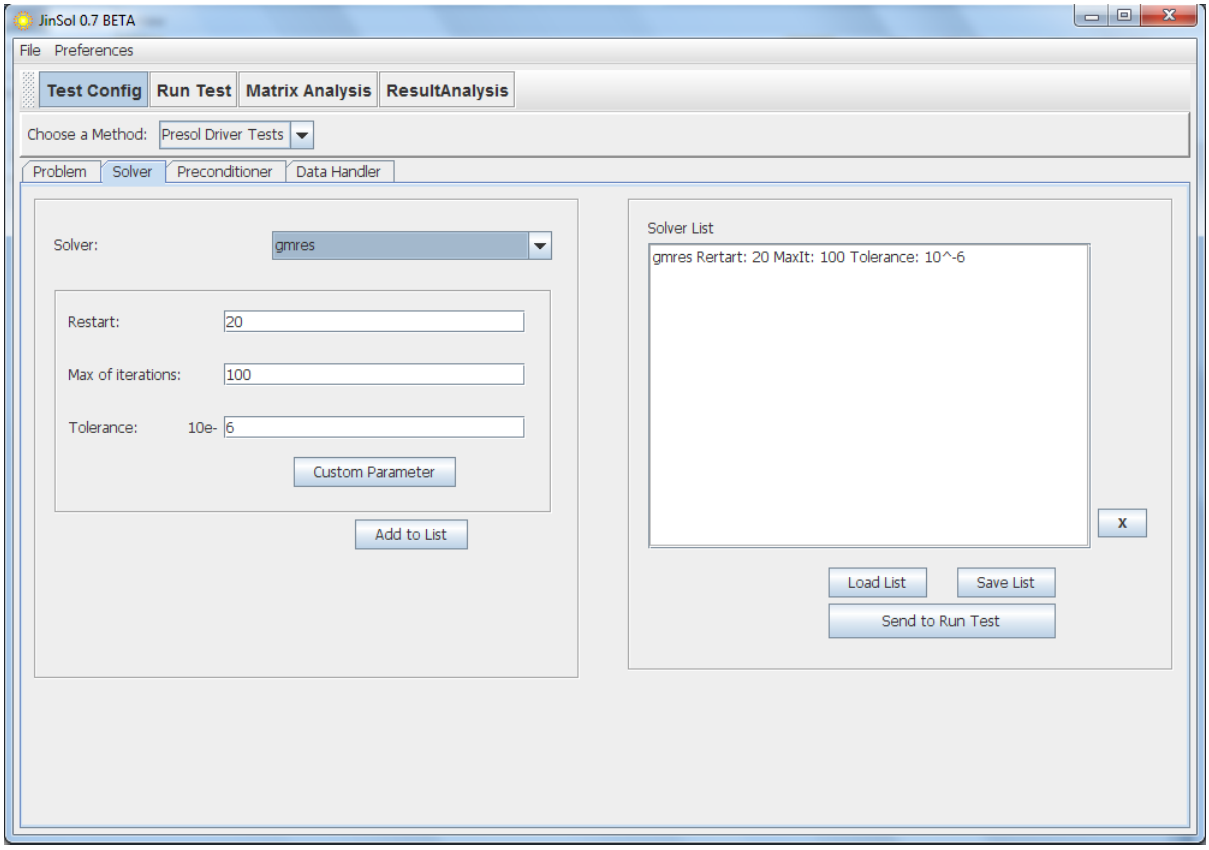


Figura 18 JinSol - Configuração do solver

Este método oferece cinco formas de precondicionamento baseados na técnica de decomposição dos domínios(Figura 19).

- Jacobi (J)

  Bloco Jacobi(inexato) com correção de espaço grosseiro e ILU(0) como o *solver* aproximado em cada subdomínio.

- Schur (S)

  Um precondicionador de dois níveis com ILU(0) como precondicionador do interior de cada subdomínio [22].

- No preconditioner (N)

  Sem precondicionador.

- Diagonal (D)

  Precondicionador diagonal aplicado no domínio completo. Isto é, sem subdomínios.

- Additive Schwarz (A)

  Schwarz aditivo com sobreposição de faixa de um nó entre os subdomínios. O precondicionador de cada subdomínio é ILU(0) [23].

Utilizaremos cada um destes precondicionadores no nosso teste com os parâmetros oferecidos como padrão pelo JinSol, como podem ser vistos na Figura 19 à direita.

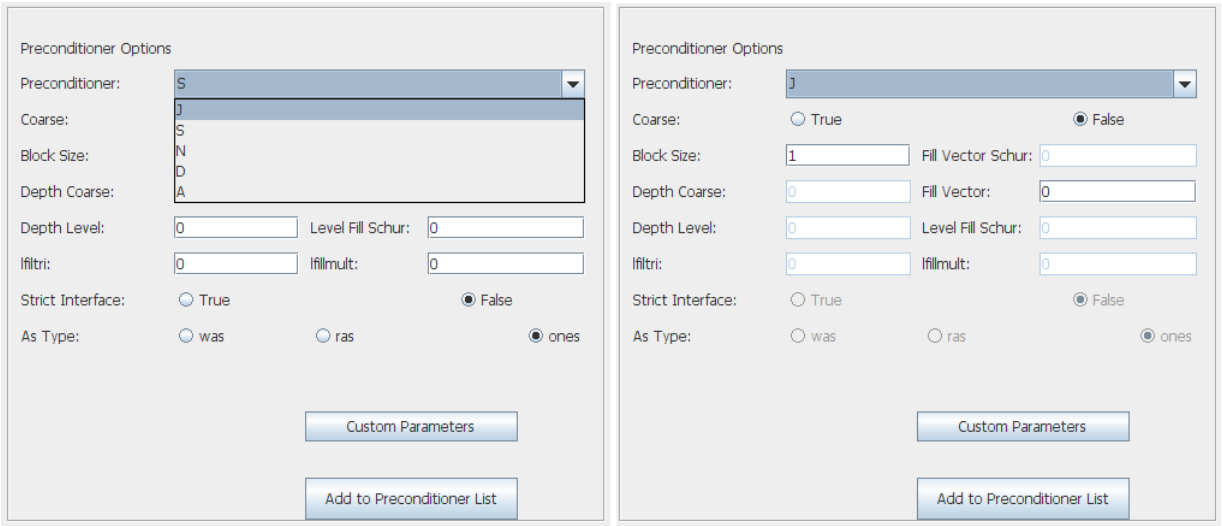


Figura 19 JinSol - Precondicionadores com decomposição de domínio

3.2 Basic Solver

Para este teste utilizaremos matrizes construídas pela opção *Sparse Convection Difusion* que constrói a matriz apartir de outro operador. Neste caso o operador sendo discretizado é $L(u) = \nabla \cdot (u\vec{v}) - \varepsilon \Delta u$ com condições de contorno de Dirichlet homogêneas. A configuração dos problemas no JinSol pode ser visto na Figura 20

Os testes realizados no método Basic Solver seguem a descrição feita no capítulo anterior. Usaremos como *solver* o GMRES precondicionado pela esquerda e o GMRES precondicionado pela direita como pode ser visto na Figura 21.

Iremos utilizar uma lista de cinco precondicionadores sendo um deles de fatoração LU incompleta e os demais de diferentes técnicas de aproximação da inversa, além da

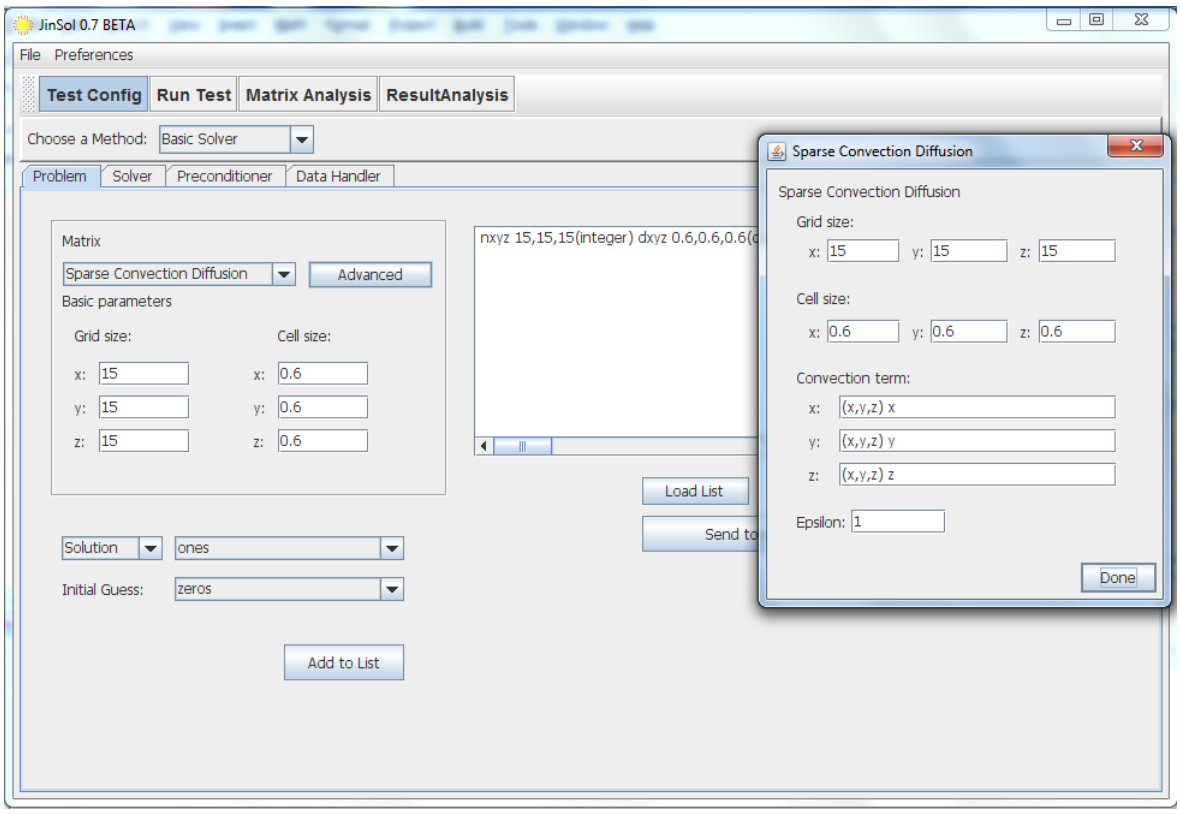


Figura 20 JinSol - Configuração dos problemas

opção de solução sem precondicionamento. Nenhum reordenamento será utilizado. A lista de precondicionamento pode ser visto na Figura 22.

3.3 Solver Ges

Os sistemas lineares utilizados neste teste foram obtidos a partir da discretização das equações de Navier-Stokes, que modelam o deslocamento de água e outros componentes durante o enchimento de um reservatório, dentro da descrição semi-Lagrangeana na simulação de escoamento de fluidos incompressíveis. Os problemas de simulação utilizados foram retirados de [20] onde descrições mais detalhadas podem ser obtidas. Buscaremos os resultados para simulações de um canal.

As matrizes geradas nestas simulações constituem problemas de ponto de sela generalizados com as seguintes características:

1. $\mathbf{A}$ é simétrica: $\mathbf{A} = \mathbf{A}^T$;

2. $\mathbf{A}$ é positiva definida: $\boldsymbol{x}^T \mathbf{A} \boldsymbol{x} > 0 \quad \forall \boldsymbol{x} \neq 0$;

3. $\mathbf{G}^T \neq \mathbf{D}$;

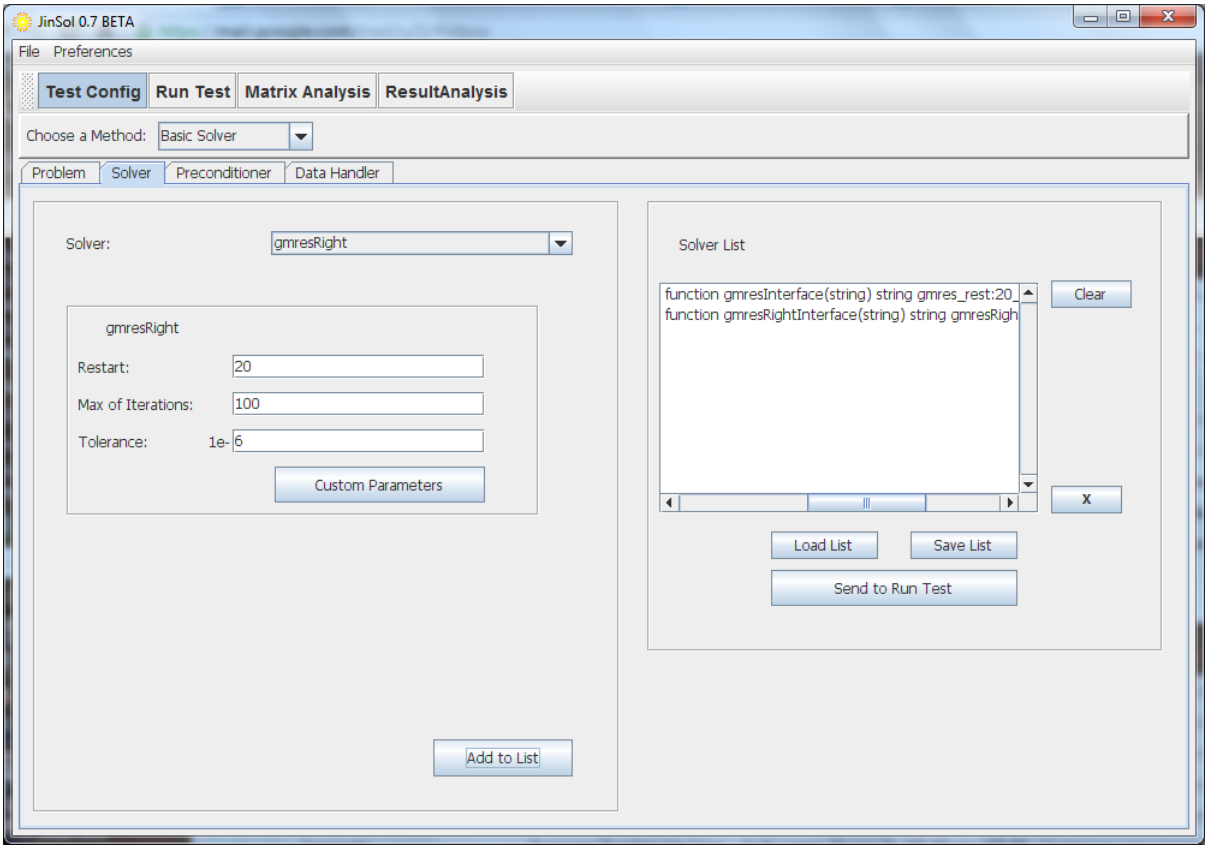


Figura 21 JinSol - Configuração dos *solvers*

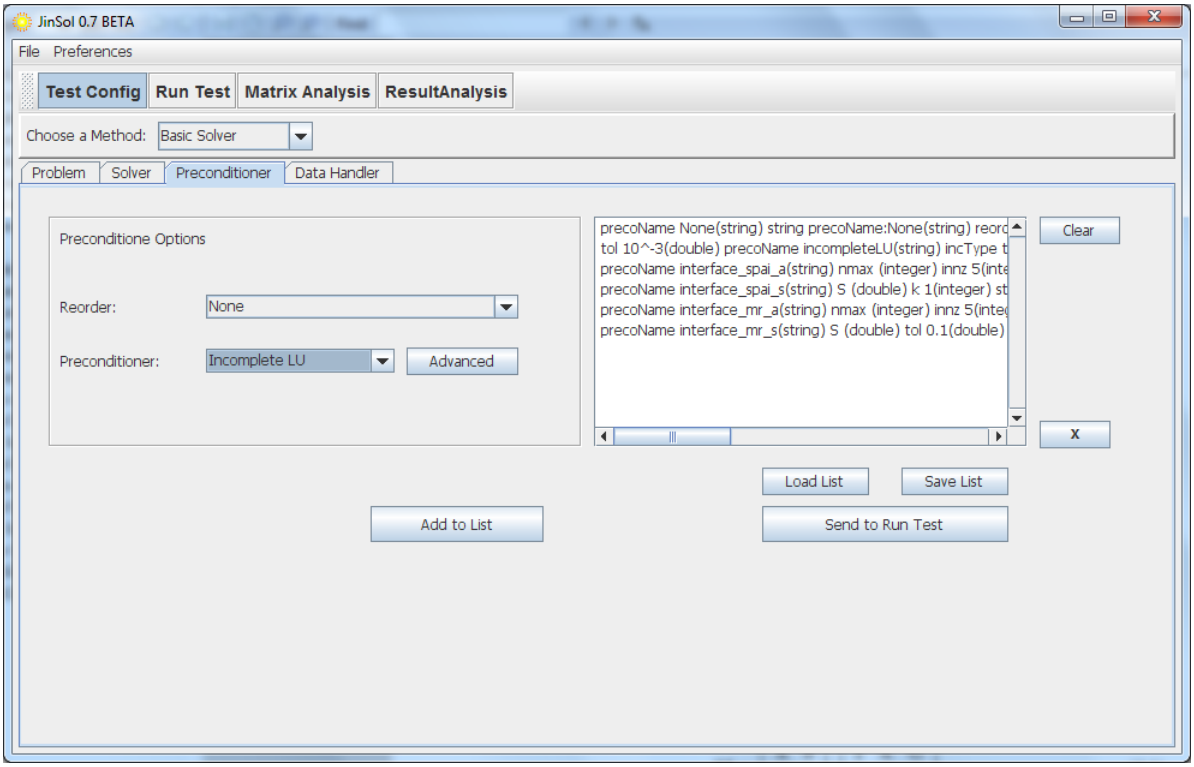


Figura 22 JinSol - Configuração da lista de precondicionamento

4. $\mathbf{C} = \mathbf{0}$ ($\mathbf{C}$ é uma matriz identicamente nula);

Este exemplo consiste em simular o escoamento de um fluido através de um canal tridimensional. As condições de entrada e saída de fluido são aplicadas nas fronteiras laterais, enquanto que nas outras fronteiras são aplicadas velocidades nulas. Considerou-se para este exemplo, um canal de dimensões: $10 \times 5 \times 3$ que gerou uma matriz A de ordem 1.620, onde 1.5% são elementos não nulos e existem 1.485 componentes de velocidade e 135 de pressão. Número de Reynolds Re=10.000 e CFL 1 e 5.

A geometria, dimensões e condições de entrada e saída podem ser vistos, a partir de um corte transversal, na Figura 23.

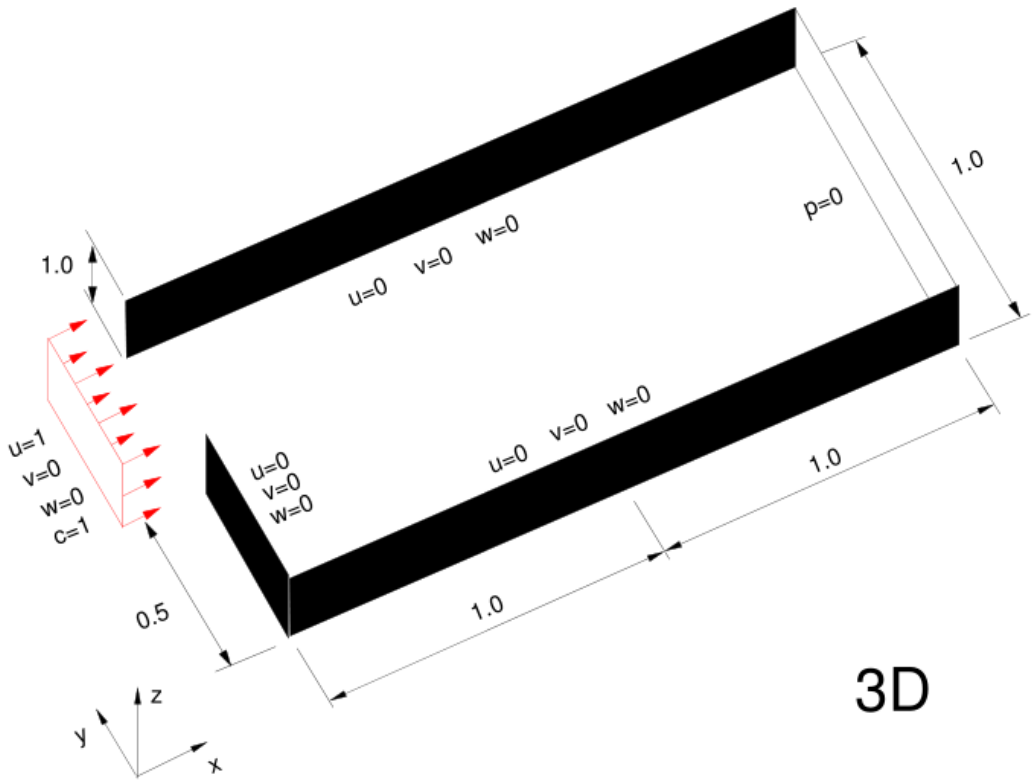


Figura 23 Domínio para o escoamento em um canal.

A configuração dos problemas no programa pode ser observada na Figura 24

Usaremos como *solver* o GMRES precondicionado pela esquerda e o GMRES precondicionado pela direita como fizemos no Basic Solver (Figura 21).

No método Solver Ges o precondicionador é obtido através de uma aproximação da fatoração LU em blocos da matriz de ponto de sela. Por exemplo, seja S o complemento de Schur de $\boldsymbol{A}$ [19] definido por $\mathbf{S} = \mathbf{C} - \mathbf{D}\mathbf{A}^{-1}\mathbf{G}$ suponha

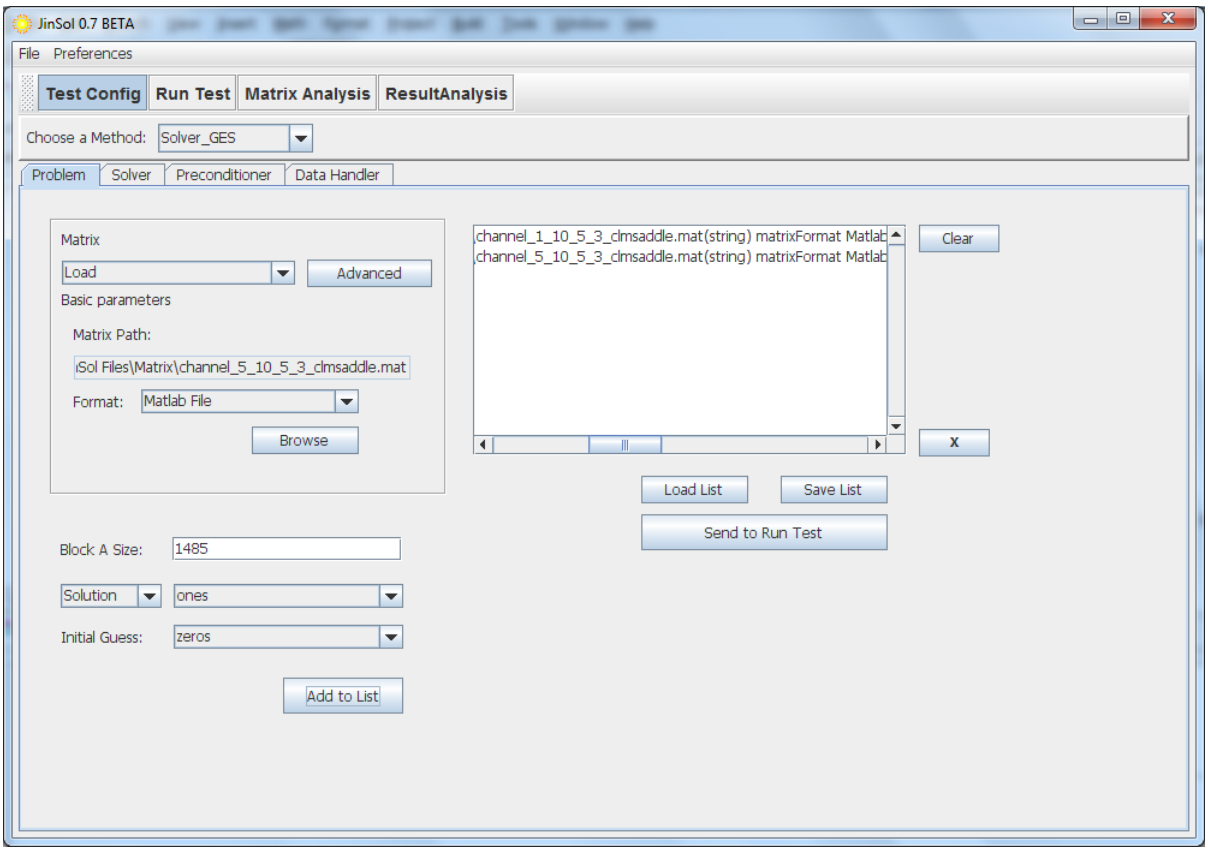


Figura 24 JinSol - Configuração do problema para o método Solver Ges

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{D} & \tilde{\mathbf{S}} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \tilde{\mathbf{A}}^{-1}\mathbf{G} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{3.1}$$

onde $\tilde{\mathbf{A}}$ é uma aproximação de $\mathbf{A}$ e $\tilde{\mathbf{S}} = -\mathbf{D}\tilde{\mathbf{A}}^{-1}\mathbf{G}$ é uma aproximação de $\mathbf{S}$. Na aplicação do precondicionador é preciso obter $r = M^{-1}Av$ ou $r = M^{-1}y$, $y = Av$. Para calcular $r = M^{-1}y$ iremos resolver o sistema $Mr = y$

$$\mathbf{M}r = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{D} & \tilde{\mathbf{S}} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \tilde{\mathbf{A}}^{-1}\mathbf{G} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \end{bmatrix} \tag{3.2}$$

$$\begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{D} & \tilde{\mathbf{S}} \end{bmatrix} \begin{bmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \end{bmatrix} \tag{3.3}$$

$$\mathbf{A}\boldsymbol{z}_1 = \boldsymbol{y}_1 \tag{3.4}$$

$$\tilde{\mathbf{S}}\boldsymbol{z}_2 = \boldsymbol{y}2 - \mathbf{D}\boldsymbol{z}_1 \tag{3.5}$$

Para resolver (3.4) utilizamos um método de Krylov precondicionado (chamado de *solver*

A no programa), aproveitando as propriedades conhecidas da matriz $\mathbf{A}$. Com z_1 obtido em (3.5) conhecemos $-\mathbf{D}\boldsymbol{z}_1$. Para resolver (3.5) utilizamos outro método de Krylov precondicionado (chamado de *solver* S no programa). Tendo calculado z_1 e z_2 iremos calcular

$$\begin{bmatrix} \mathbf{I} & \tilde{\mathbf{A}}^{-1}\mathbf{G} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{bmatrix} \tag{3.6}$$

$$\boldsymbol{r}_2 = \boldsymbol{z}_2 \tag{3.7}$$

$$\boldsymbol{z}_1 + \tilde{\mathbf{A}}^{-1}\mathbf{G}\boldsymbol{z}_2 = \boldsymbol{z}_1 \tag{3.8}$$

$$\boldsymbol{r}_1 = \boldsymbol{z}_1 - \tilde{\mathbf{A}}^{-1}\mathbf{G}\boldsymbol{z}_2 \tag{3.9}$$

que será obtido através apenas de produto, obtendo então o vetor r. Na Figura 25 podemos observar a interface do programa para configuração do teste.

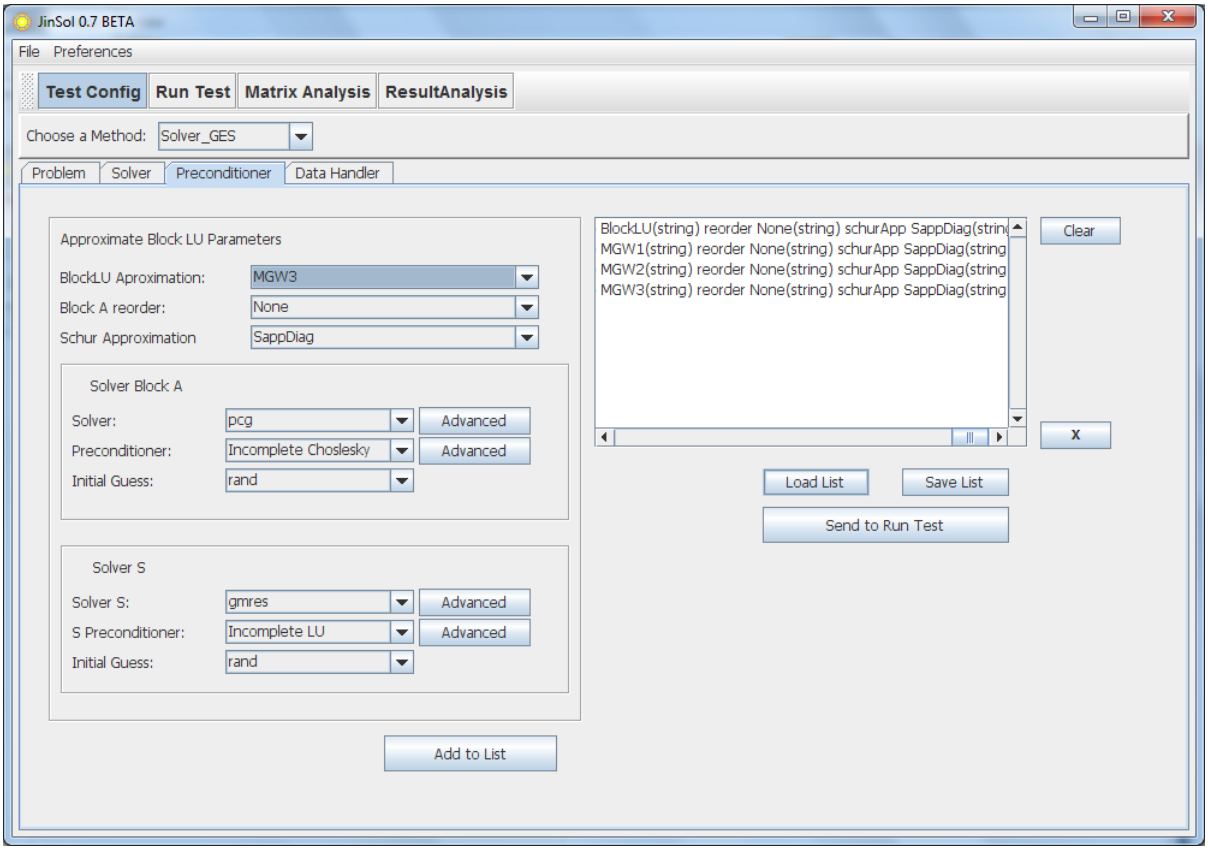


Figura 25 JinSol - Configuração dos precondicionadores do método Solver Ges

O método Solver Ges oferece quatro formas diferentes de realizar a fatoração LU em blocos.

- bluckLU

  Apresentada em (3.1)

- MGW1

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{S}} \end{bmatrix} \tag{3.10}$$

- MGW2

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{S}} \end{bmatrix} \tag{3.11}$$

- MGW3

$$\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{G} \\ \mathbf{D} & 2\tilde{\mathbf{S}} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{D}\mathbf{A}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{S}} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{A}^{-1}\mathbf{G} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{3.12}$$

Neste teste utilizaremos cada uma destas fatorações. Como visto acima o bloco $\mathbf{A}$ é simétrico postivo definido, e para fazer uso desta propriedade iremos utilizar o método PCG como *solver* A precondicionado por uma fatoração Choslesky Incompleta. Para o *solver* S será utilizado o GMRES precondicionado com uma fatoração LU incompleta. A aproximação $\tilde{\mathbf{A}} \approx \mathbf{A}$ será a diagonal da matriz $\mathbf{A}$.

4 RESULTADOS

O objetivo deste capítulo é apresentar os resultados e demais dados gerados dos testes propostos na primeira seção do capítulo anterior a partir da utilização do JinSol. Para cada método iremos começa apresentando os dados obtidos através da área de análise de matrizes e em seguida alguns dos resultados das tabelas de resultados geradas diretamente pelo programa.

4.1 Presol Driver Test

As figuras abaixo apresentam o padrão de esparsidade da matriz do problema(Figura 26) descrito no capítulo anterior, a distribuição dos seus autovalores no plano complexo (Figura 27) e outras informações e estatísticas(Figura 28) para o caso homogêneo.

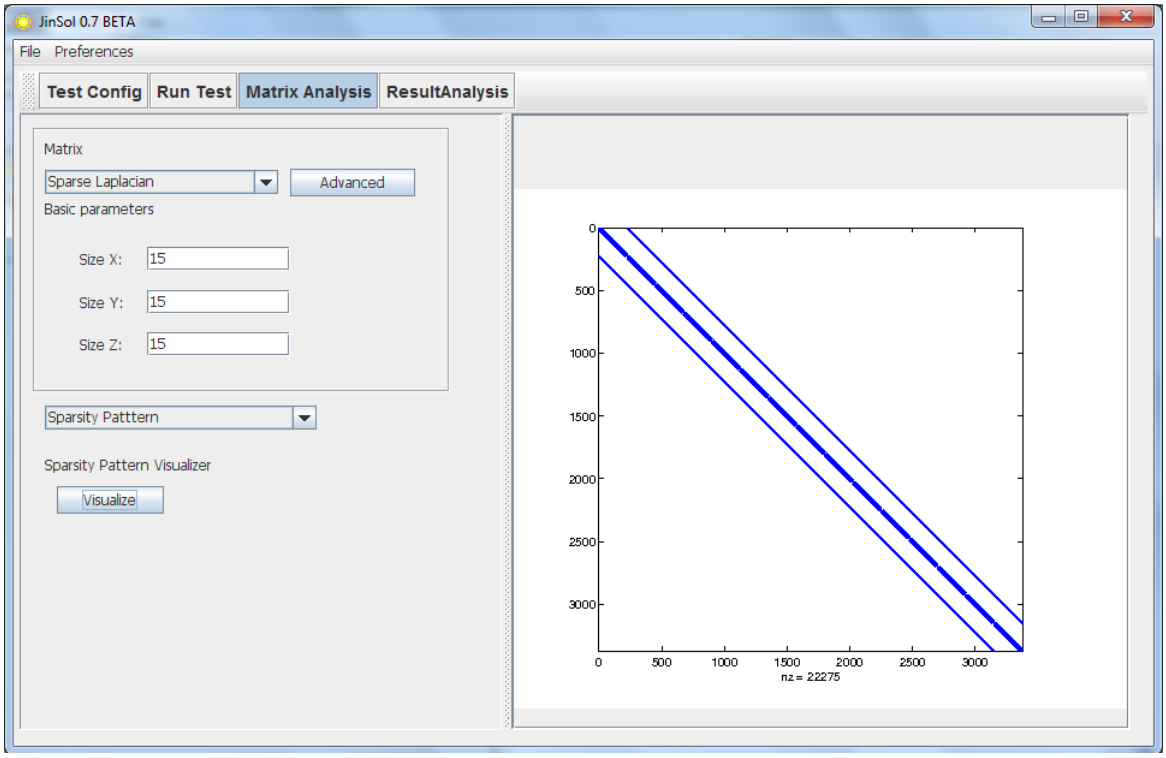


Figura 26 Presol Driver Test - Padrão de esparsidade do problema para o caso homogêneo

A Figura 29 apresenta a tabela de resultados gerada em formato csv . A Primeira linha apresenta o cabeçalho cujas colunas se referem, respectivamente, à:

A - Número do Teste

B - Nível de preenchimento da fatoração LU incompleta

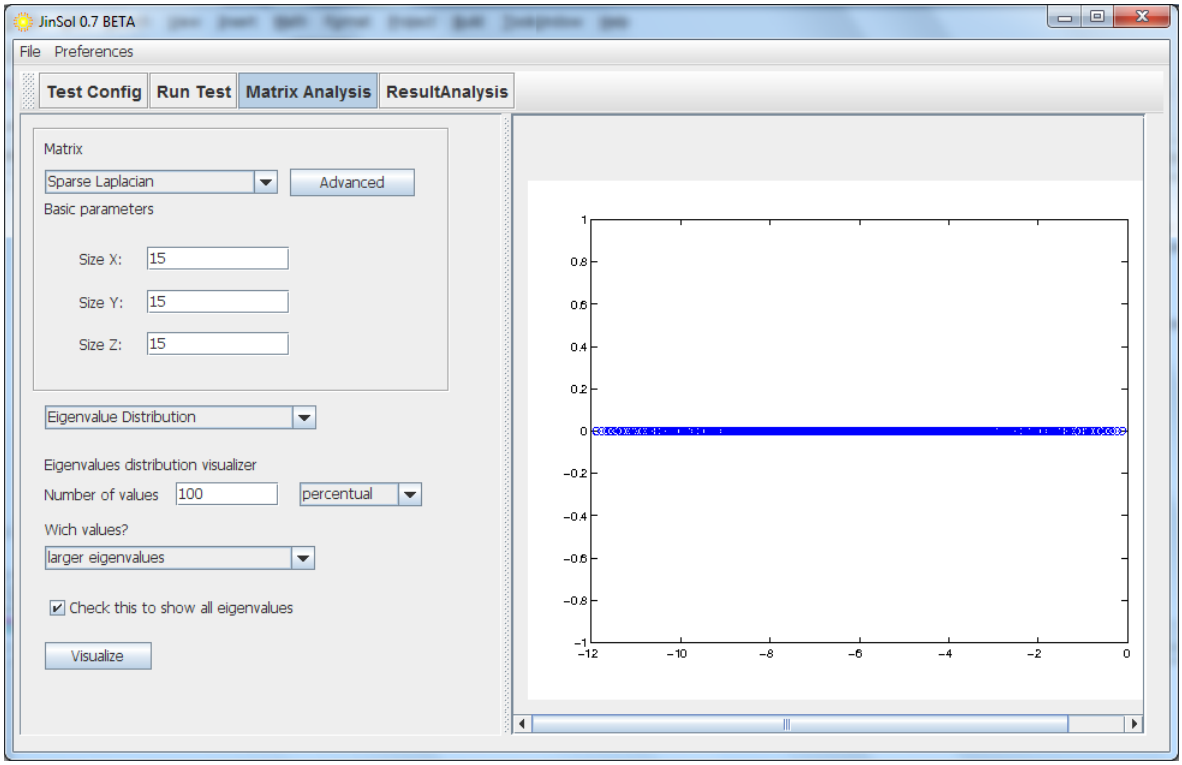


Figura 27 Presol Driver Test - Distribuição dos autovalores do problema no plano complexo para o caso homogêneo

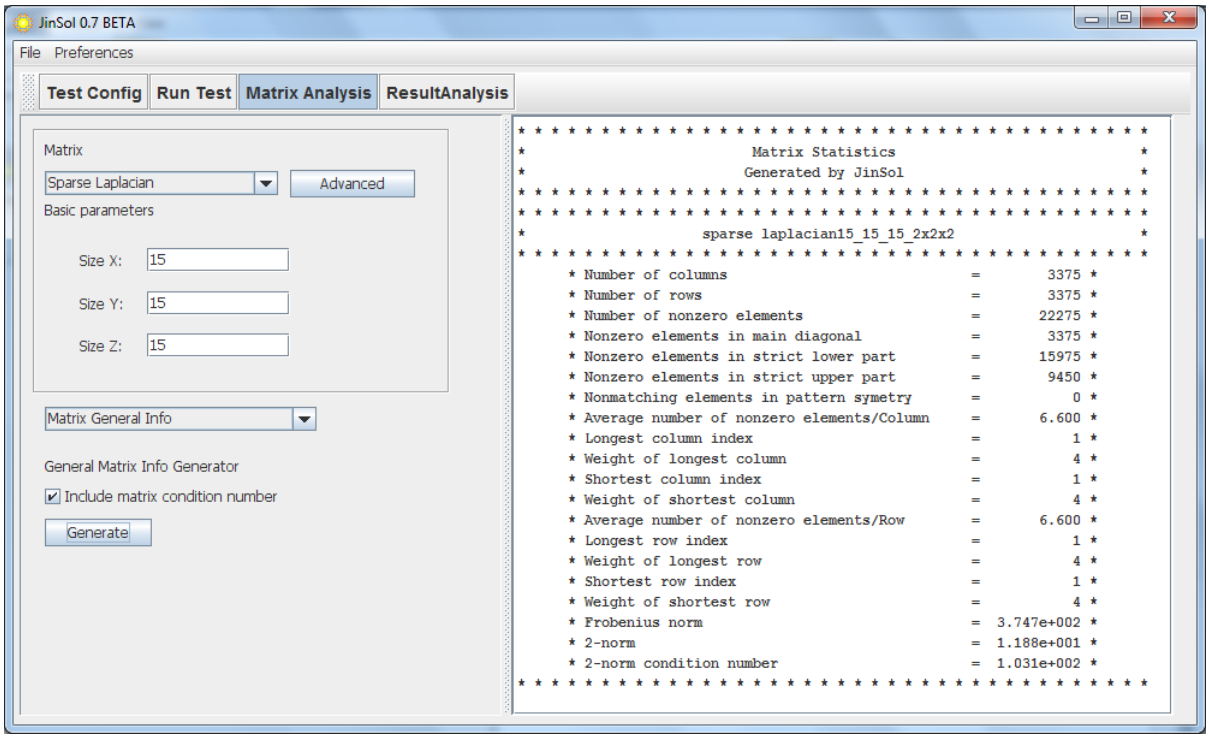


Figura 28 Presol Driver Test - Informações e estatísticas sobre a matriz do problema no caso homogêneo

C - Número de elementos não nulos na matriz

D - Razão entre o número de elementos não nulos do precondicionador e da matriz do problema

E - *Flag* de resultado do *solver*

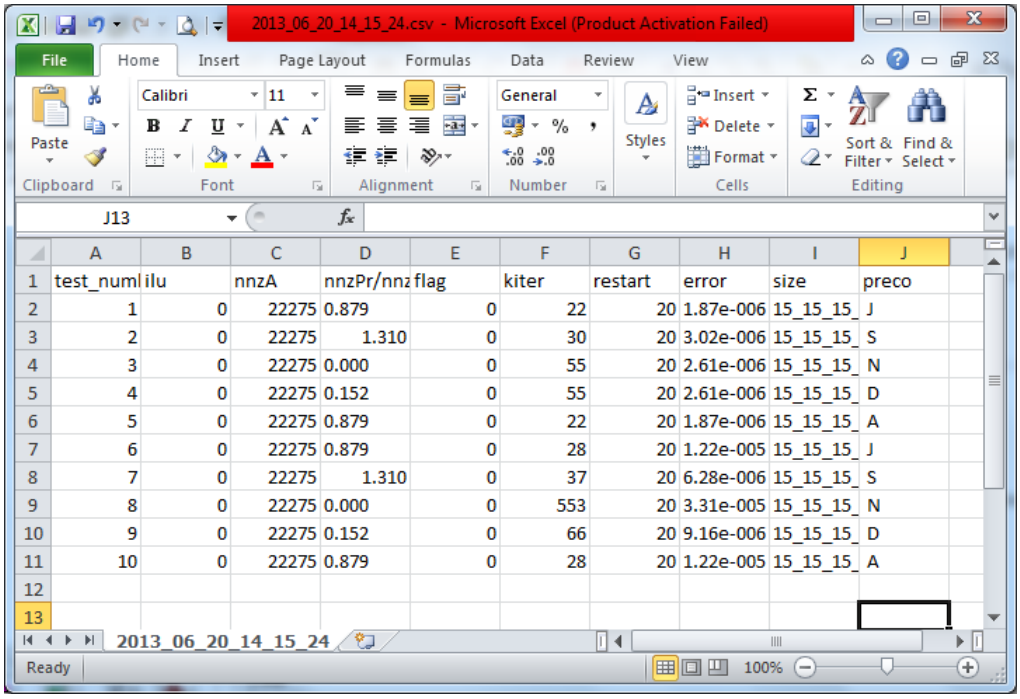
F - Número de iterações do *solver*

G - Parâmetro de restart do *solver*

H - Norma do erro

I - As dimensões do domínio

J - O Precondicionador usado



	A	B	C	D	E	F	G	H	I	J
1	test_numl	ilu	nnzA	nnzPr/nnz	flag	kiter	restart	error	size	preco
2	1	0	22275	0.879	0	22	20	1.87e-006	15_15_15_	J
3	2	0	22275	1.310	0	30	20	3.02e-006	15_15_15_	S
4	3	0	22275	0.000	0	55	20	2.61e-006	15_15_15_	N
5	4	0	22275	0.152	0	55	20	2.61e-006	15_15_15_	D
6	5	0	22275	0.879	0	22	20	1.87e-006	15_15_15_	A
7	6	0	22275	0.879	0	28	20	1.22e-005	15_15_15_	J
8	7	0	22275	1.310	0	37	20	6.28e-006	15_15_15_	S
9	8	0	22275	0.000	0	553	20	3.31e-005	15_15_15_	N
10	9	0	22275	0.152	0	66	20	9.16e-006	15_15_15_	D
11	10	0	22275	0.879	0	28	20	1.22e-005	15_15_15_	A
12										
13										

Figura 29 Presol Driver Test - Resultados obtidos no teste

4.2 Basic Solver

As figuras abaixo apresentam o padrão de esparsidade da matriz do problema(Figura 30) descrito na segunda seção do capítulo anterior, a distribuição dos seus autovalores no plano complexo (Figura 31) e outras informações e estatísticas(Figura 32).

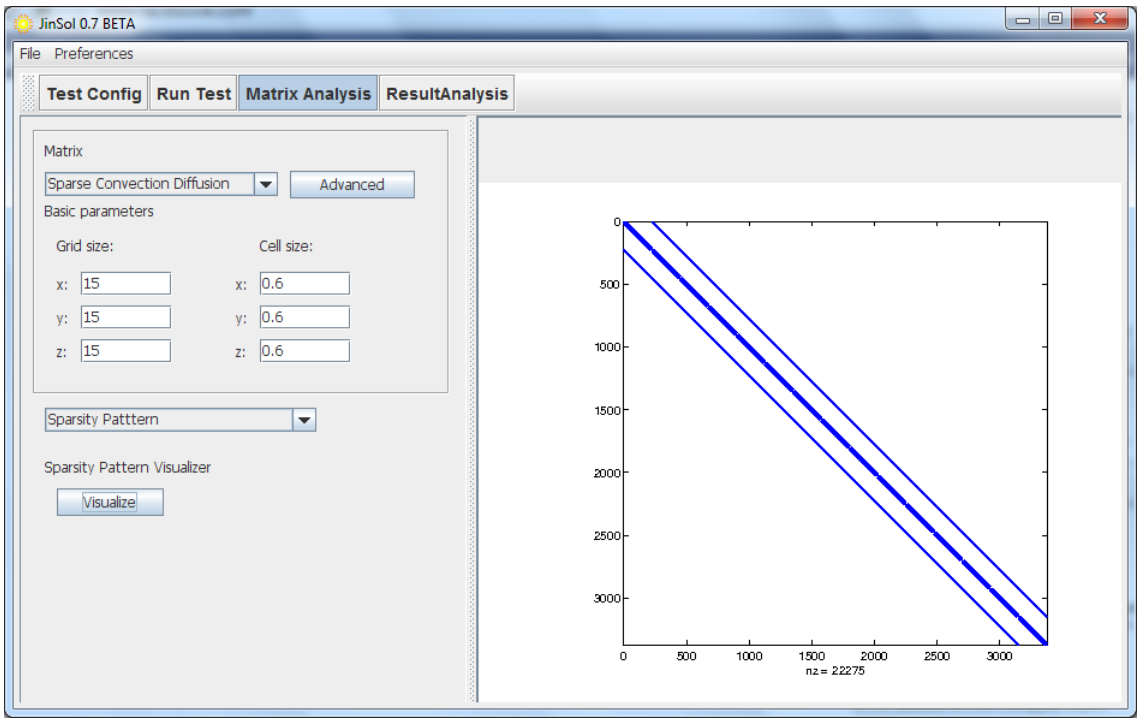


Figura 30 Basic Solver - Padrão de esparsidade do problema

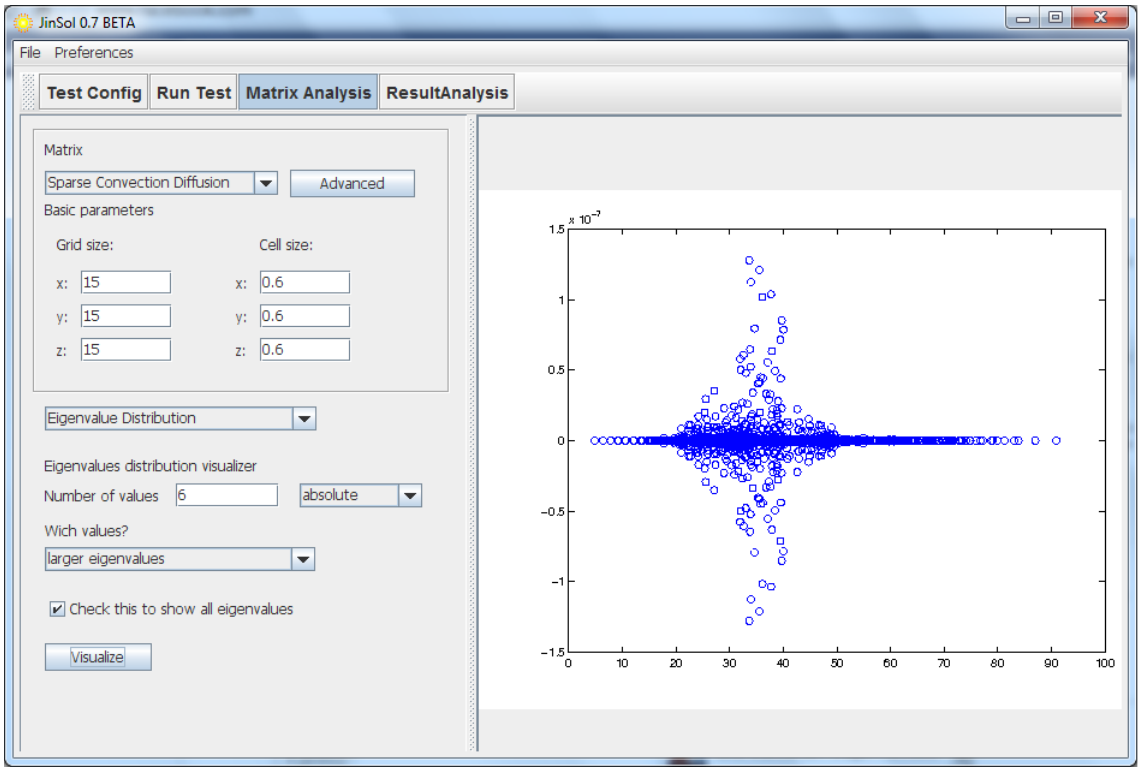


Figura 31 Basic Solver - Distribuição dos autovalores do problema no plano complexo

O método Basic Solver fornece dois formatos de relatório, um para análise visual rápida, em arquivo de texto que pode ser visualizado no próprio programa, e outro em csv, semelhante ao visto na Figura 29.

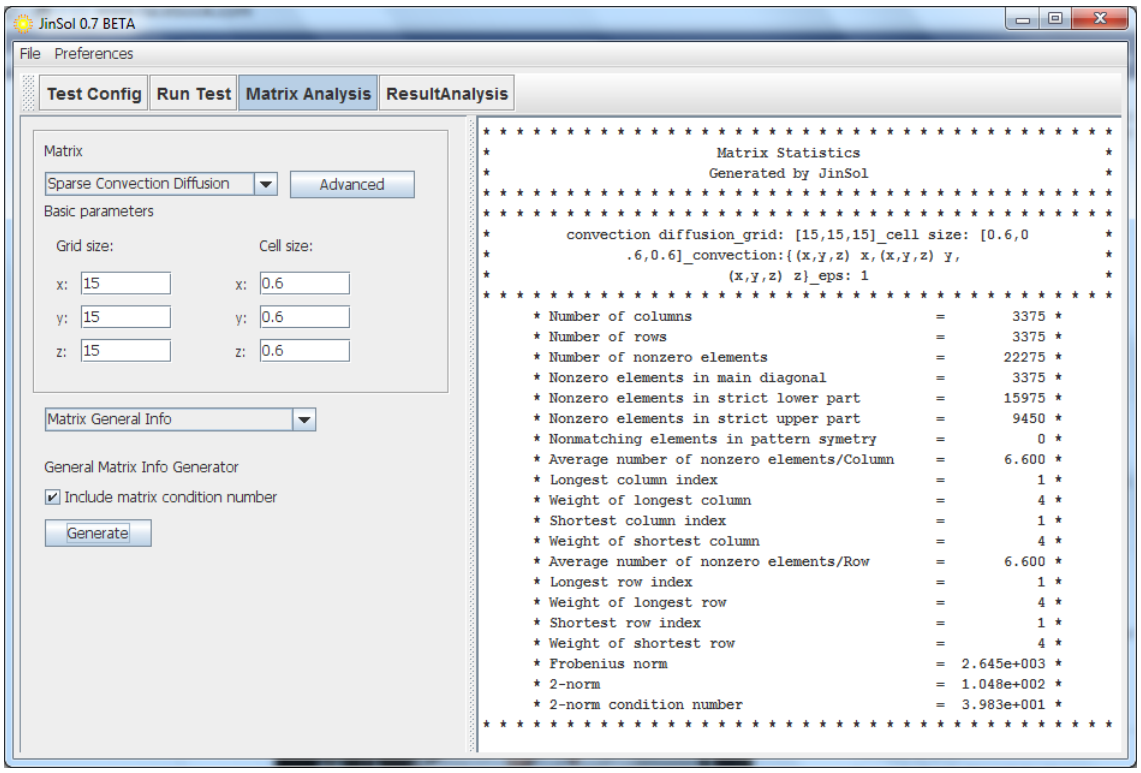


Figura 32 Basic Solver - Informações e estatísticas sobre a matriz do problema no caso homogêneo

Na Figura 33 pode ser observado o relatório de texto no JinSol. Neste relatório para cada matriz são apresentados os *solvers* com uma tabela dos resultados do teste com a matriz e o *solver* correspondente. Em cada tabela as colunas se referem espectivamente à:

1. Precondicionador
2. Tempo de construção do precondicionador
3. Razão entre o número de elementos não nulos do precondicionador e da matriz do problema
4. Número de iterações
5. *Flag* de resultado do *solver*
6. Resíduo Relativo

Na Figura 39 temos o gráfico do decaimento de resíduo para os testes com GMRES precondicionado pela esquerda. O número de cada gráfico corresponde ao número do teste na tabela de resultados respectivamente.

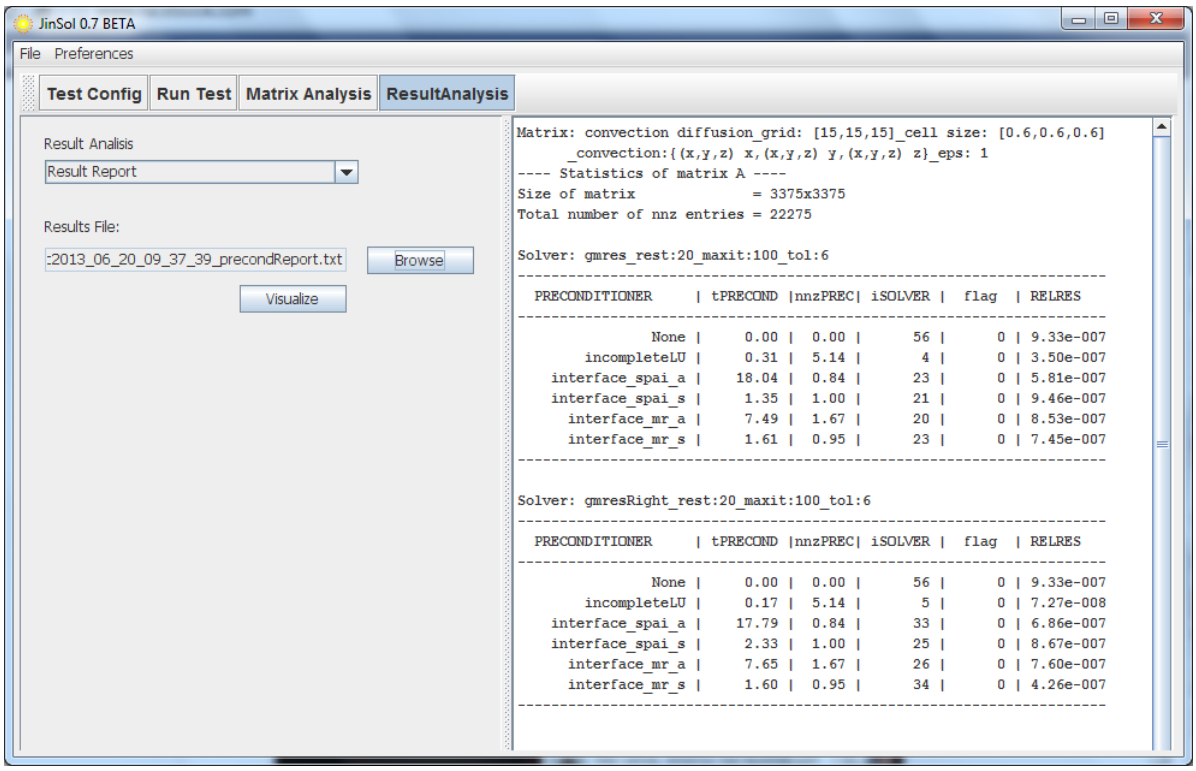


Figura 33 Basic Solver - Resultados obtidos no teste

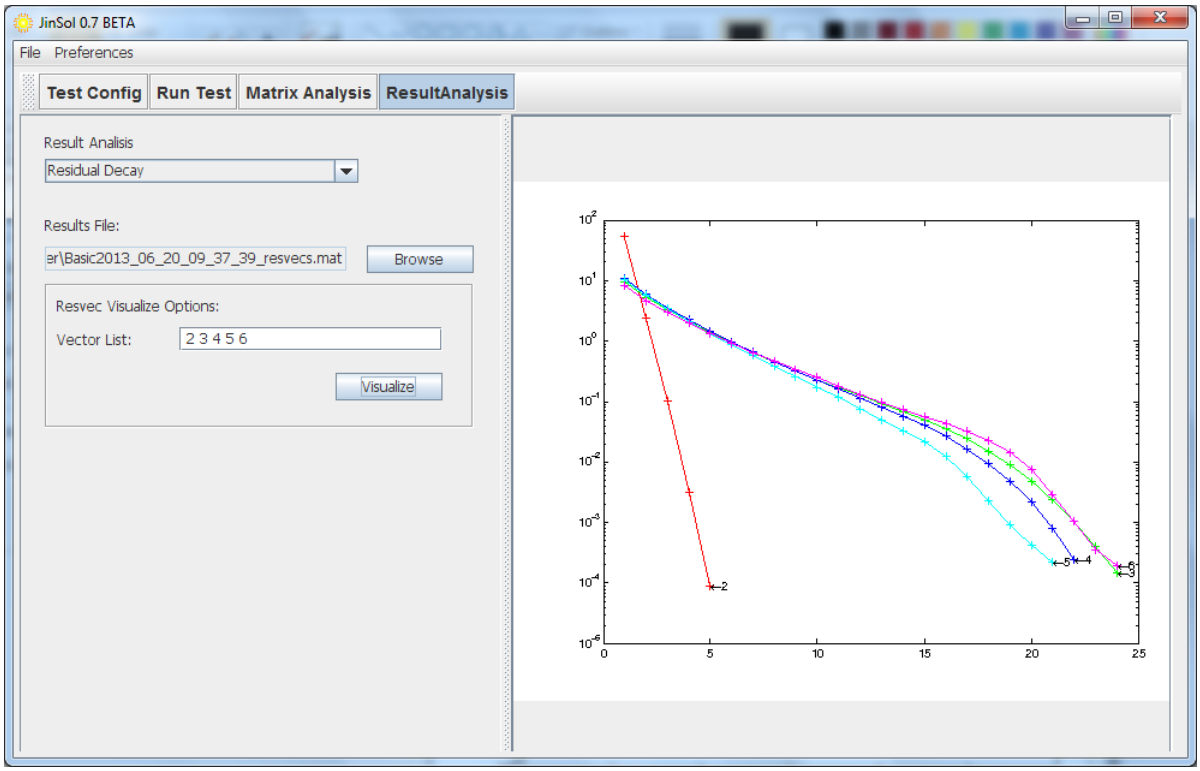


Figura 34 Basic Solver - Gráfico de decaimento do resíduo

4.3 Solver Ges

As figuras abaixo apresentam o padrão de esparsidade da matriz do problema(Figura 35) descrito na terceira seção do capítulo anterior, para o caso com CFL = 1, a distribuíção dos seus auto valores no plano complexo (Figura 36) e outras informações e estatísticas(Figura 37).

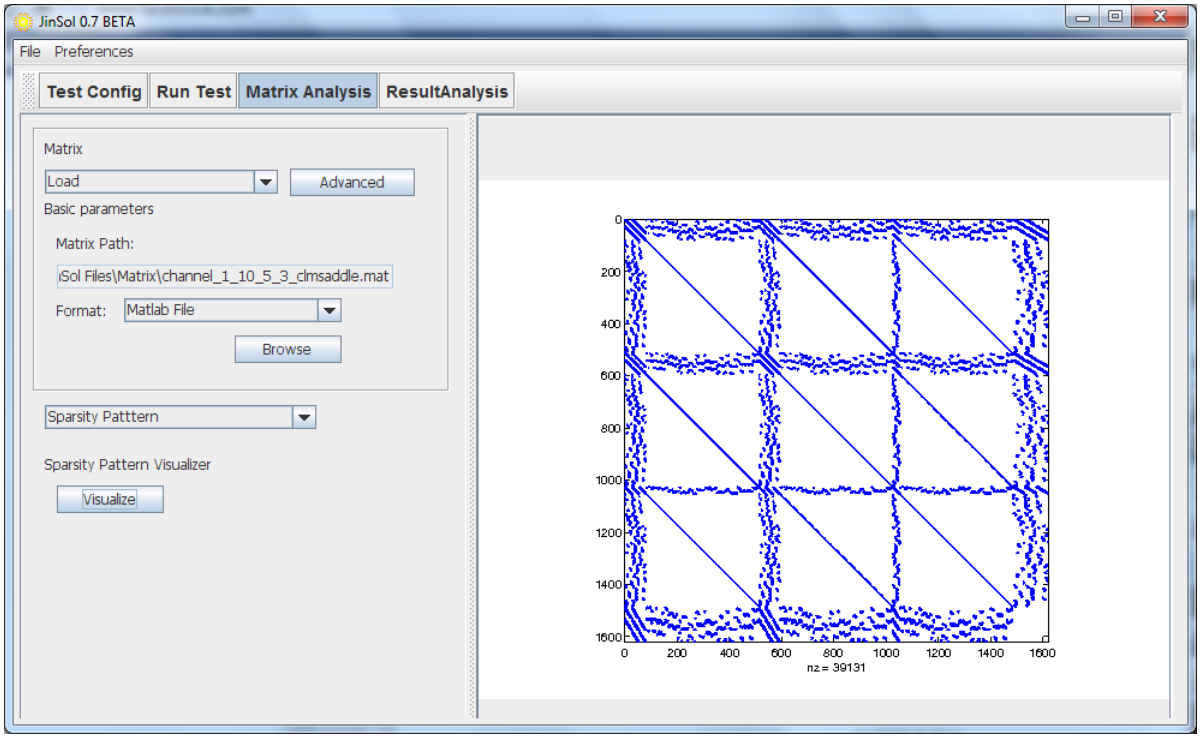


Figura 35 Solver Ges - Padrão de esparsidade do problema

O método Solver Ges fornece um relatório em formato cvs com as informações do teste e resultados. A Figura 38 apresenta o arquivo carregado em um programa de planilha eletrônica. A primeira linha apresenta o cabeçalho cujas colunas se referem, respectivamente, à:

A - Número do teste

B - Matriz do problema

C - Número de elementos não nulos na matriz

D - Dimensão da matriz

E - O *solver* do problema

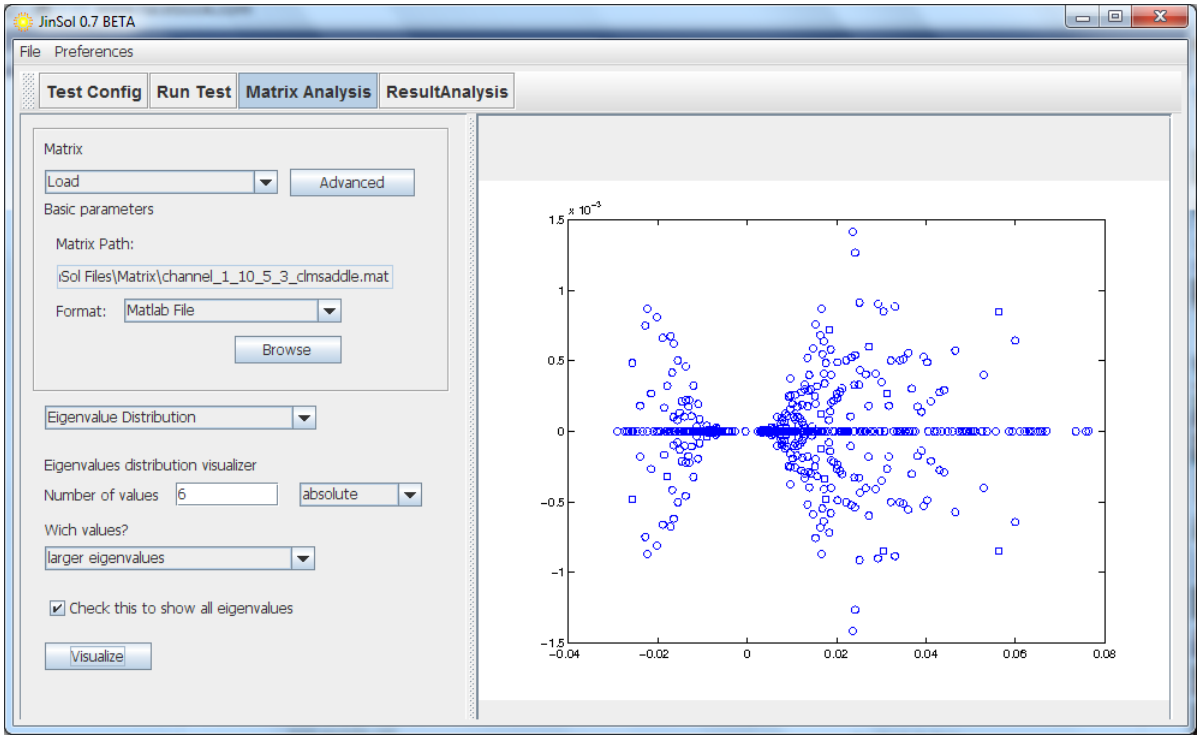


Figura 36 Solver Ges - Distribuição dos autovalores do problema no plano complexo no caso com CFL = 1

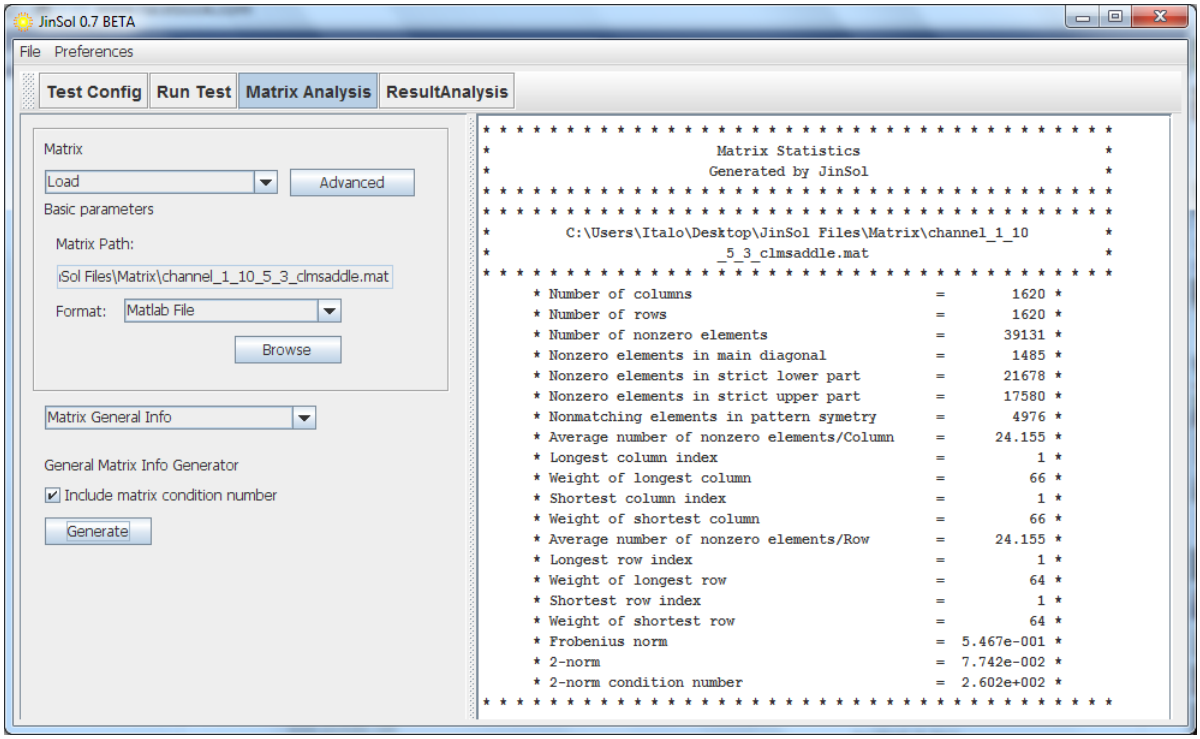


Figura 37 Solver Ges - Informações e estatísticas sobre a matriz do problema no caso com CFL = 1

F - A fatoração LU aproximada do precondicionador

G - O *solver* A

H - O precondicionador do *solver* A

I - O *solver* S

J - O precondicionador do *solver* S

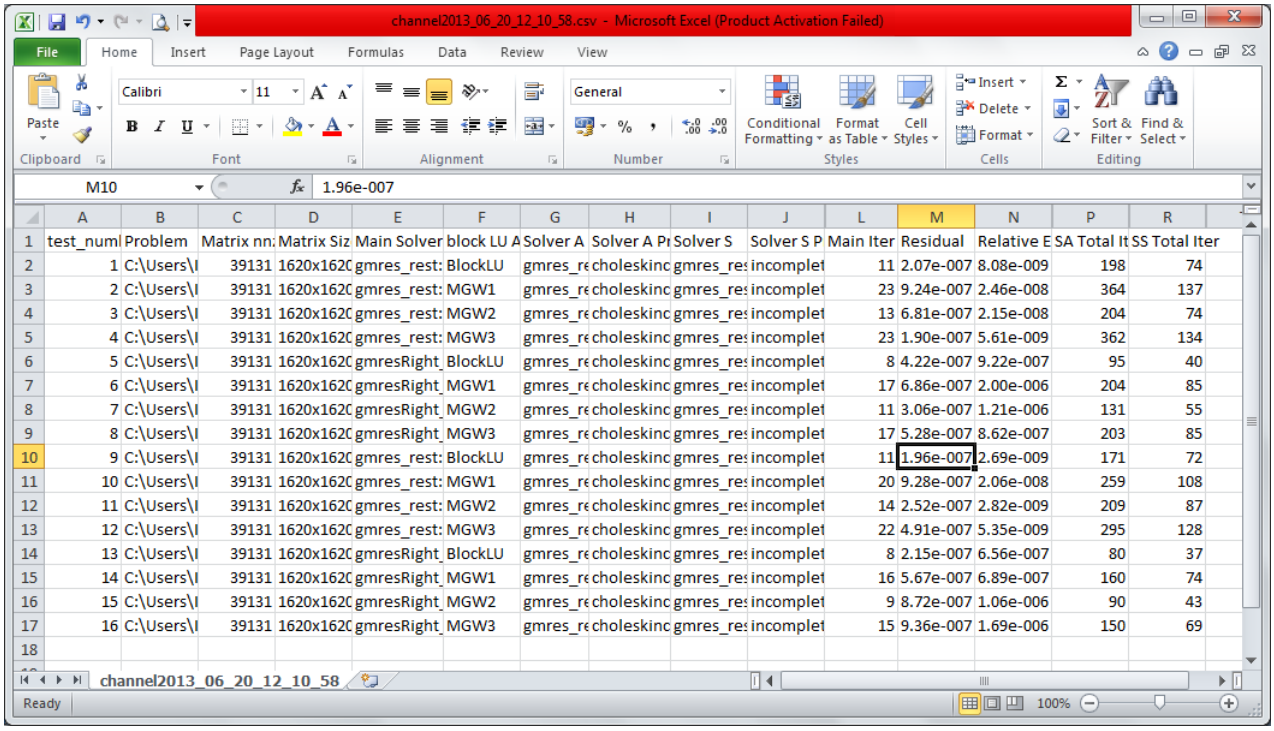
L - O número de iterações do solver do problema

M - O resíduo relativo

N - O erro relativo

P - A soma do número de iterações do *solver* A de cada iteração

R - A soma do número de iterações do *solver* S de cada iteração



test_numl	Problem	Matrix nn:	Matrix Siz	Main Solver	block LU A	Solver A	Solver A Pr	Solver S	Solver S P	Main Iter	Residual	Relative E	SA Total It	SS Total Iter
1	C:\Users\I	39131	1620x1620	gmres_rest:	BlockLU	gmres_re	choleskinc	gmres_res	incomplet	11	2.07e-007	8.08e-009	198	74
2	C:\Users\I	39131	1620x1620	gmres_rest:	MGW1	gmres_re	choleskinc	gmres_res	incomplet	23	9.24e-007	2.46e-008	364	137
3	C:\Users\I	39131	1620x1620	gmres_rest:	MGW2	gmres_re	choleskinc	gmres_res	incomplet	13	6.81e-007	2.15e-008	204	74
4	C:\Users\I	39131	1620x1620	gmres_rest:	MGW3	gmres_re	choleskinc	gmres_res	incomplet	23	1.90e-007	5.61e-009	362	134
5	C:\Users\I	39131	1620x1620	gmresRight_	BlockLU	gmres_re	choleskinc	gmres_res	incomplet	8	4.22e-007	9.22e-007	95	40
6	C:\Users\I	39131	1620x1620	gmresRight_	MGW1	gmres_re	choleskinc	gmres_res	incomplet	17	6.86e-007	2.00e-006	204	85
7	C:\Users\I	39131	1620x1620	gmresRight_	MGW2	gmres_re	choleskinc	gmres_res	incomplet	11	3.06e-007	1.21e-006	131	55
8	C:\Users\I	39131	1620x1620	gmresRight_	MGW3	gmres_re	choleskinc	gmres_res	incomplet	17	5.28e-007	8.62e-007	203	85
9	C:\Users\I	39131	1620x1620	gmres_rest:	BlockLU	gmres_re	choleskinc	gmres_res	incomplet	11	1.96e-007	2.69e-009	171	72
10	C:\Users\I	39131	1620x1620	gmres_rest:	MGW1	gmres_re	choleskinc	gmres_res	incomplet	20	9.28e-007	2.06e-008	259	108
11	C:\Users\I	39131	1620x1620	gmres_rest:	MGW2	gmres_re	choleskinc	gmres_res	incomplet	14	2.52e-007	2.82e-009	209	87
12	C:\Users\I	39131	1620x1620	gmres_rest:	MGW3	gmres_re	choleskinc	gmres_res	incomplet	22	4.91e-007	5.35e-009	295	128
13	C:\Users\I	39131	1620x1620	gmresRight_	BlockLU	gmres_re	choleskinc	gmres_res	incomplet	8	2.15e-007	6.56e-007	80	37
14	C:\Users\I	39131	1620x1620	gmresRight_	MGW1	gmres_re	choleskinc	gmres_res	incomplet	16	5.67e-007	6.89e-007	160	74
15	C:\Users\I	39131	1620x1620	gmresRight_	MGW2	gmres_re	choleskinc	gmres_res	incomplet	9	8.72e-007	1.06e-006	90	43
16	C:\Users\I	39131	1620x1620	gmresRight_	MGW3	gmres_re	choleskinc	gmres_res	incomplet	15	9.36e-007	1.69e-006	150	69

Figura 38 Solver Ges - Resultados obtidos no teste

Na Figura 39 temos o gráfico do decaimento de resíduo para os testes com GMRES precondicionado pela esquerda no caso com CFL = 1. O número de cada gráfico corresponde ao número do teste na tabela de resultados da Figura 38.

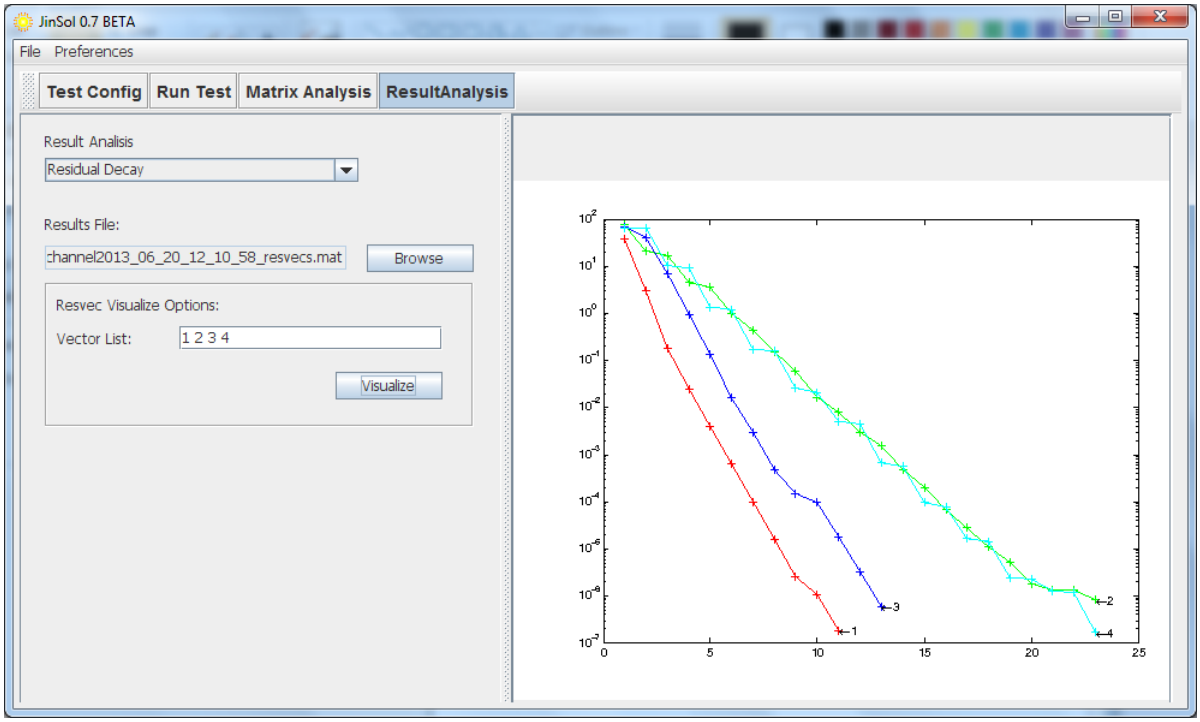


Figura 39 Basic Solver - Gráfico de decaimento do resíduo

5 CONCLUSÃO

O funcionamento da versão descrita do JinSol já passou por avaliação interna e o programa está sendo utilizado para testes em um projeto em parceria com a Petrobras no estudo de precondicionadores de inversa aproximada esparsa.

O uso do programa tem permitido que sejam implementadas melhorias baseadas na experiência dos usuários na utilização do programa, entre elas:

- Melhor organização de dados em relatórios.
- Implementações visuais que facilitam o uso do programa.
- Armazenamento de dados úteis para análises posteriores.

A instalação do programa em outros computadores se mostrou simples tanto no sistema operacional Windows quanto no Ubuntu. Esta portabilidade é um dos objetivos desde a formulação do programa e que parece ter sido alcançada com sucesso até o presente momento.

Como planos futuros iremos adicionar o suporte para que o usuário possa implantar novos *solvers*, precondicionadores e rotinas geradoras de matrizes através de arquivos xml, que poderão ser gerados através de uma interface gráfica simples e intuitiva. A implementação desta ferramenta permite que qualquer usuário possa usar a interface gráfica para auxiliar nos testes de rotinas desenvolvidas por ele próprio, tornando o JinSol uma ferramenta importante para o desenvolvimento de novos métodos e precondicionadores para sistemas lineares.

Alguns planos não muito distantes envolvem a adição de rotinas em linguagem C, C++ e Fortran e uma comunicação com os pacotes PETSc (http://www.mcs.anl.gov/petsc) e Trilinos (http://trilinos.sandia.gov/packages), entre outros. Serão adicionadas também diferentes ferramentas de análise das informações oriundas do problema e um banco de dados de testes estruturado para evitar a execução de testes repetidos desnecessariamente, perda de informações, e facilitar outros testes comparativos. A adição de novas ferramentas de análise também está em processo de implantação como gerar dados sobre a matriz após aplicação do precondicionador e o cálculo dos valores singulares da matriz [16].

Outro objetivo é permitir testes automatizados de comparação afim de encontrar, dentro de um certo aspecto, o melhor método para os problemas desejados, utilizando métodos heurísticos e métodos de descida local para rodar um número reduzido de testes.

REFERÊNCIAS

[1] CARVALHO, L. M.; GRATTON, S. *Avanços em métodos de Krylov para solução de sistemas lineares de grande porte*. Segunda edição. São Carlos - SP, Brasil: SBMAC, 2012.

[2] SAAD, Y. *Iterative Methods for Sparse Linear Systems*. [S.l.]: SIAM, 2003.

[3] IPSEN, I.; MEYER, C. D. The idea behind krylov methods. *Amer. Math*, Monthly, 105(10), p. 889–899, 1998.

[4] SAAD, Y.; SCHULTZ, M. H. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific Computing*, Vol. 7, No. 3, p. 856–869, 1986.

[5] ARNOLDI, W. E. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quart of Appl Math*, v. 9, p. 17–29, 1951.

[6] HESTENES, M. R.; STIEFEL, E. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau Standards*, v. 49, p. 409–436, 1952.

[7] SAAD, Y. Krylov subspace methods for solving large unsymmetric linear systems. *Mathematics of Computation*, v. 37, p. 105–126, 1981.

[8] STEWART, G. W. *Matrix Algorithms. Volume II: Eigensystems*. [S.l.]: SIAM, 2001.

[9] MEURANT, G. *Computer Solution of Large Linear Systems*. North-Holland: Elsevier, 1999.

[10] STRANG, G. *Introduction To Linear Algebra*. Third edition. Wellesley - Massachusetts: Wellesley-Cambridge Press, 2003.

[11] MEYER, C. D. *Matrix analysis and applied liner algebra*. [S.l.]: SIAM, 2000.

[12] STEWART, G. W. *Introduction to Matrix Computations*. New York: Academic Press, 1973.

[13] HOUSEHOLDER, A. S. Unitary triangularization of a nonsymmetric matrix. *Journal ACM*, v. 5(4), p. 339–342, 1958.

[14] BARRETT, R. et al. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. 2nd edition. ed. Philadelphia: SIAM press, 1994.

[15] TREFETHEN, L. N.; BAU, I. D. *Numerical linear algebra*. [S.l.]: SIAM, 1997.

[16] IPSEN, I. C. F. *Numerical Matrix Analysis: Linear systems and least squares*. Raleigh, North Carolina: SIAM, 2009.

[17] BENZI, M. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, v. 182, p. 418–477, 2002.

[18] NIEVINSKI_LIMA, I. C.; CARVALHO, L. M. Programa com interface gráfica para testes e estudo de soluções para problemas de ponto de sela. In: SBMAC. *Anais do CNMAC 2012 v.4*. Águas de Lindoia - SP: SBMAC, 2012.

[19] BENZI, M.; GOLUB, G. H.; LIESEN, J. Numerical solution of saddle point problems. *Cambridge University Press*, v. 14, n. 1, p. 1–137, 2005.

[20] FORTES, W. R. *Precondicionadores e solucionadores para resolução de sistemas lineares obtidos de simulação de enchimento de reservatórios*. Dissertação (Mestrado) — Universidade do Estado do Rio de Janeiro, 2008.

[21] ZIENKIEWICZ, O. C.; TAYLOR, R. L.; ZHU, J. Z. *The Finite Element Method: Its Basis and Fundamentals: Its Basis and Fundamentals*. [S.l.]: Butterworth-Heinemann, 2005.

[22] CARVALHO, L. M.; GIRAUD, L.; TALLEC, P. L. Algebraic two-level preconditioners for the schur complement method. *SIAM Journal on Scientific Computing*, SIAM, v. 22, n. 6, p. 1987–2005, 2001.

[23] CAI, X.-C.; SARKIS, M. A restricted additive schwarz preconditioner for general sparse linear systems. *SIAM Journal on Scientific Computing*, SIAM, v. 21, n. 2, p. 792–797, 1999.

6 APÊNDICE A - DADOS DE TESTES

Neste apêndice se encontram os dados e resultados dos testes apresentados no terceiro capítulo que por questão de organização foram omitidos no quarto capítulo. A legenda irá identificar de qual teste a imagem é proveniente e à que ela se refere.

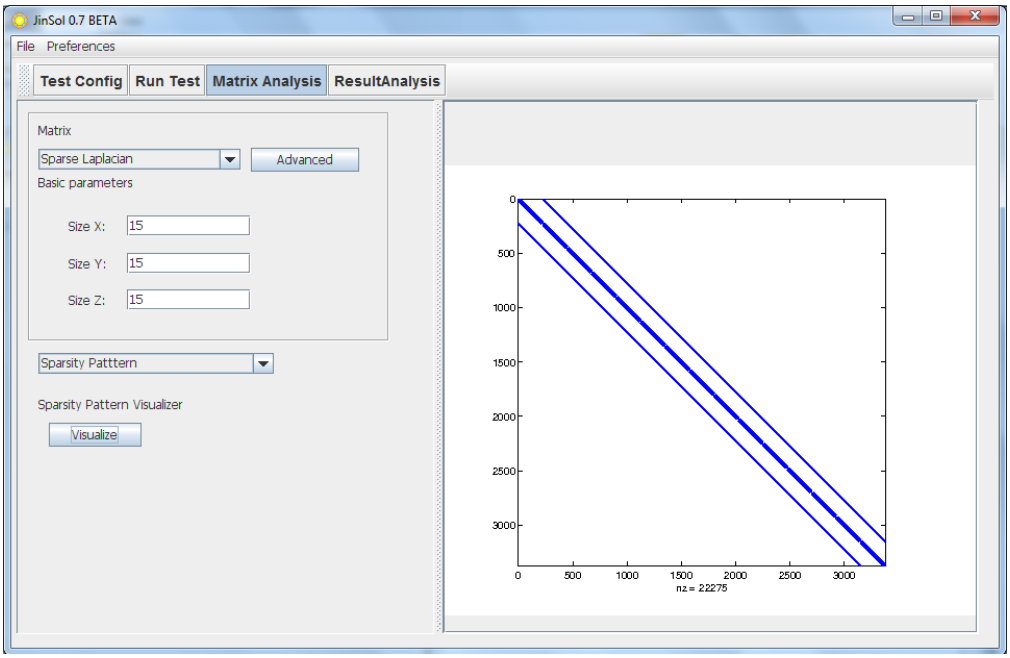


Figura 40 Presol Driver Test - Padrão de esparsidade do problema para o caso heterogêneo

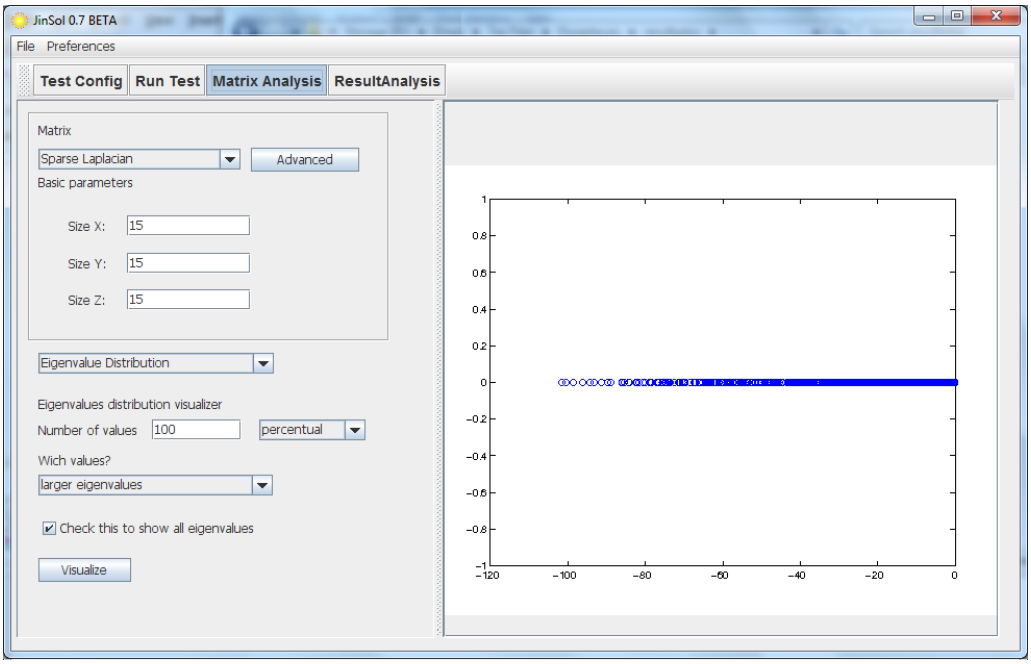


Figura 41 Presol Driver Test - Distribuição dos autovalores do problema no plano complexo para o caso heterogêneo

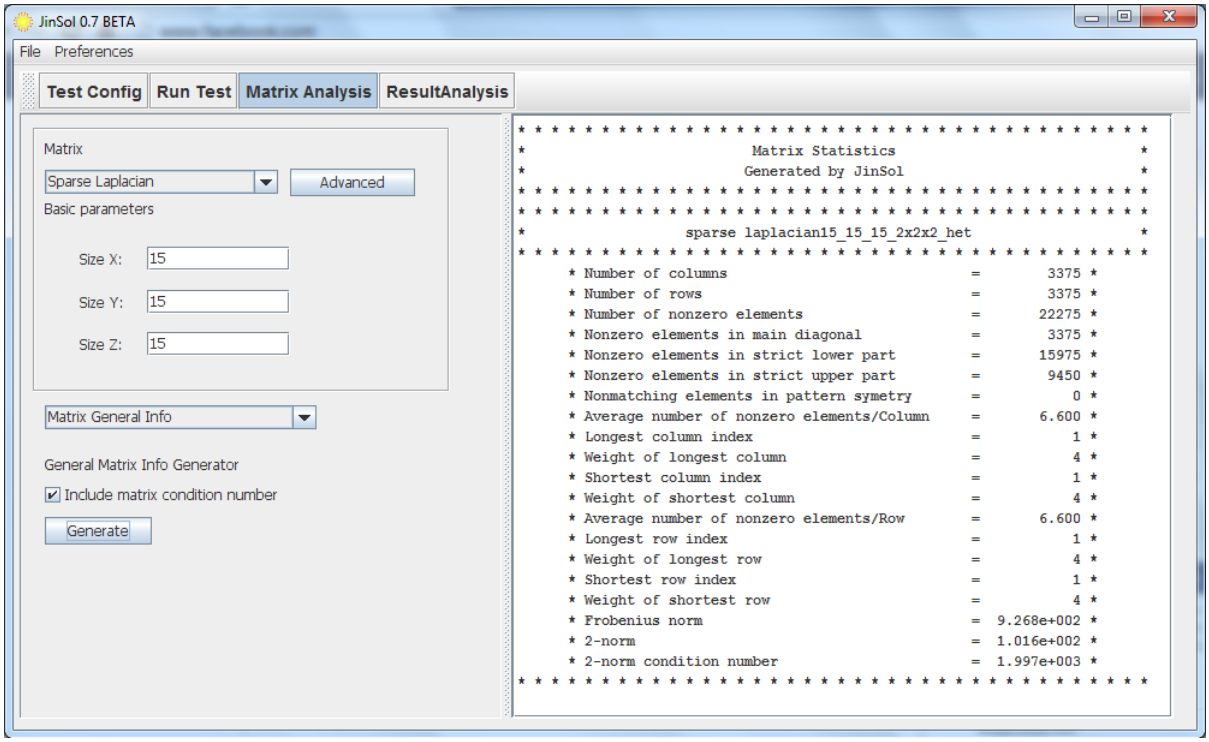
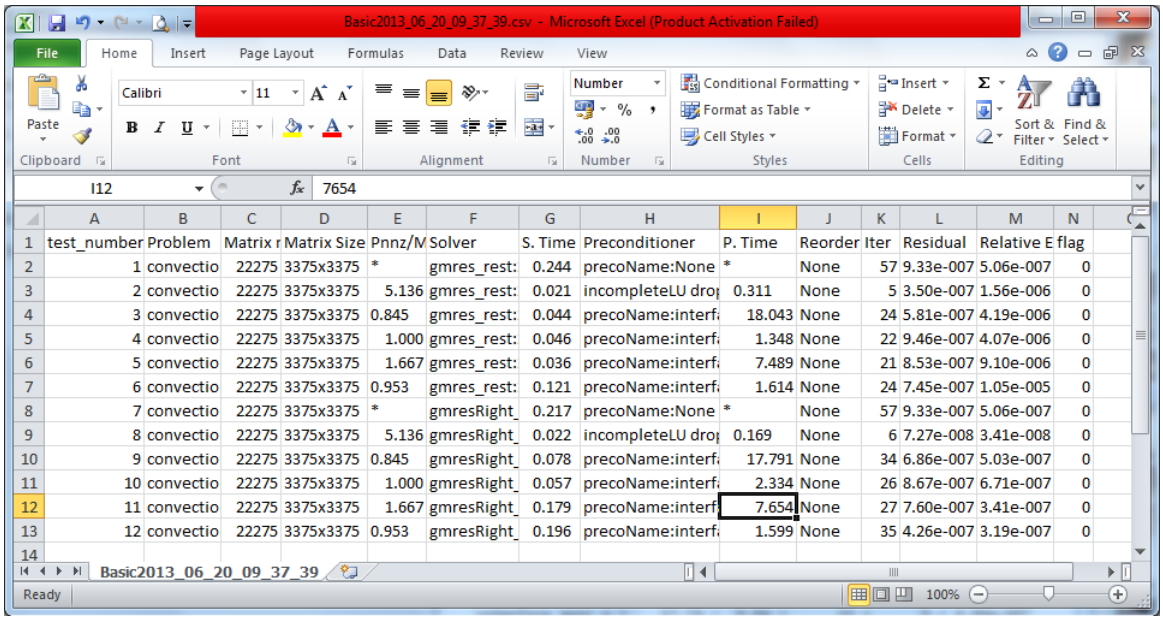


Figura 42 Presol Driver Test - Informações e estatísticas sobre a matriz do problema no caso heterogêneo



test_number	Problem	Matrix r	Matrix Size	Pnnz/M	Solver	S. Time	Preconditioner	P. Time	Reorder	Iter	Residual	Relative E	flag
1	convectio	22275	3375x3375	*	gmres_rest:	0.244	precoName:None	*	None	57	9.33e-007	5.06e-007	0
2	convectio	22275	3375x3375	5.136	gmres_rest:	0.021	incompleteLU dro	0.311	None	5	3.50e-007	1.56e-006	0
3	convectio	22275	3375x3375	0.845	gmres_rest:	0.044	precoName:interf	18.043	None	24	5.81e-007	4.19e-006	0
4	convectio	22275	3375x3375	1.000	gmres_rest:	0.046	precoName:interf	1.348	None	22	9.46e-007	4.07e-006	0
5	convectio	22275	3375x3375	1.667	gmres_rest:	0.036	precoName:interf	7.489	None	21	8.53e-007	9.10e-006	0
6	convectio	22275	3375x3375	0.953	gmres_rest:	0.121	precoName:interf	1.614	None	24	7.45e-007	1.05e-005	0
7	convectio	22275	3375x3375	*	gmresRight_	0.217	precoName:None	*	None	57	9.33e-007	5.06e-007	0
8	convectio	22275	3375x3375	5.136	gmresRight_	0.022	incompleteLU dro	0.169	None	6	7.27e-008	3.41e-008	0
9	convectio	22275	3375x3375	0.845	gmresRight_	0.078	precoName:interf	17.791	None	34	6.86e-007	5.03e-007	0
10	convectio	22275	3375x3375	1.000	gmresRight_	0.057	precoName:interf	2.334	None	26	8.67e-007	6.71e-007	0
11	convectio	22275	3375x3375	1.667	gmresRight_	0.179	precoName:interf	7.654	None	27	7.60e-007	3.41e-007	0
12	convectio	22275	3375x3375	0.953	gmresRight_	0.196	precoName:interf	1.599	None	35	4.26e-007	3.19e-007	0

Figura 43 Basic Solver - Relatório obtidos a partir dos testes no formato csv carregado em um programa de planilha eletrônica.

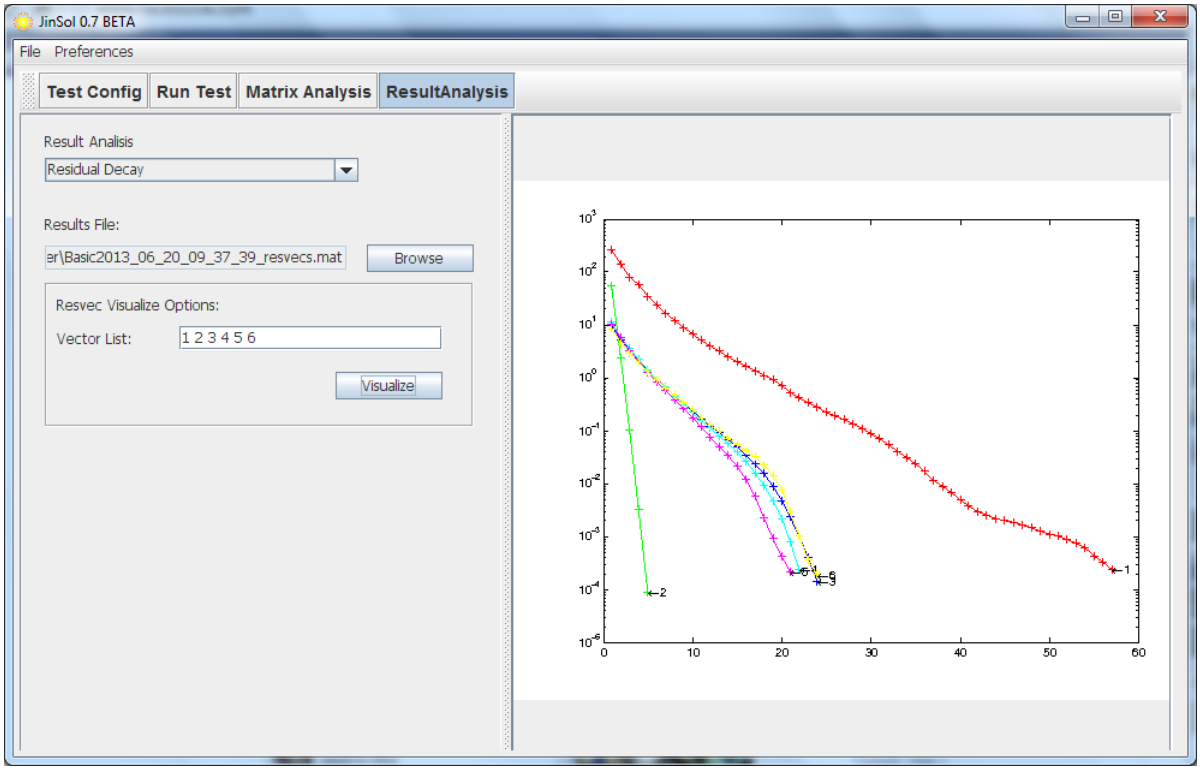


Figura 44 Basic Solver - Gráfico de decaimento do resíduo. Os números no gráfico referem-se ao número do teste na Figura 43

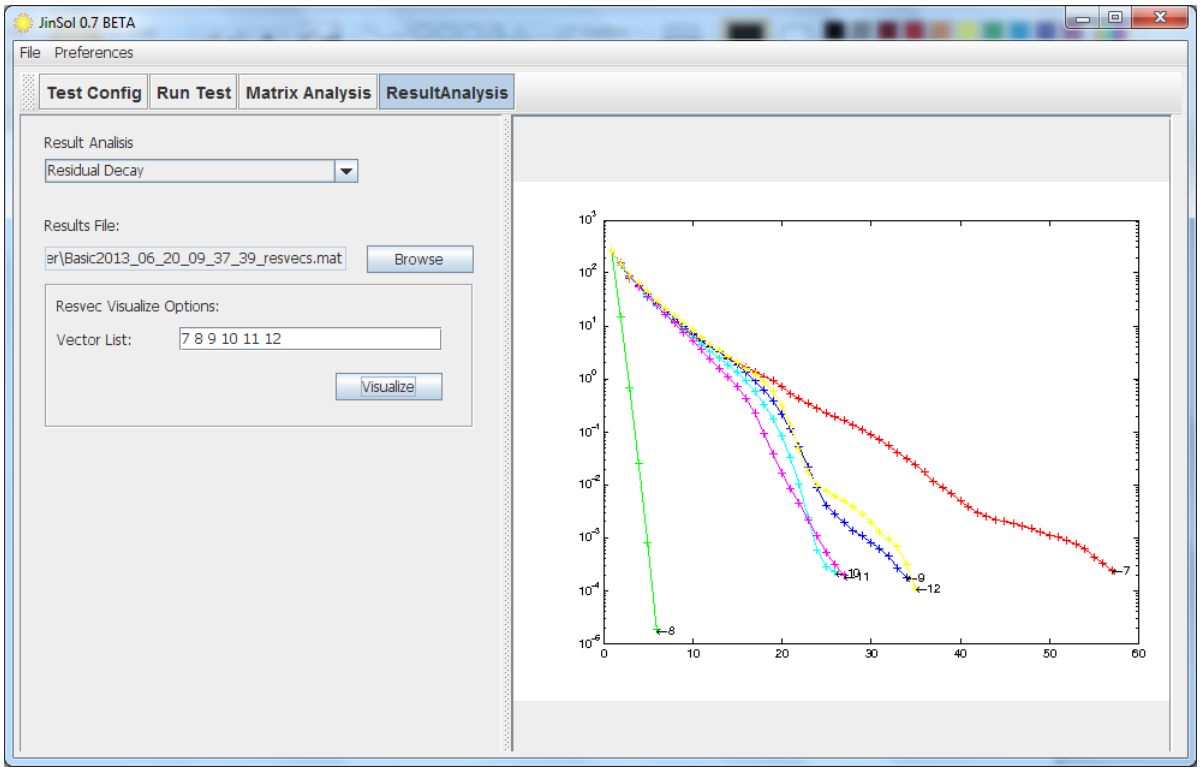


Figura 45 Basic Solver - Gráfico de decaimento do resíduo. Os números no gráfico referem-se ao número do teste na Figura 43

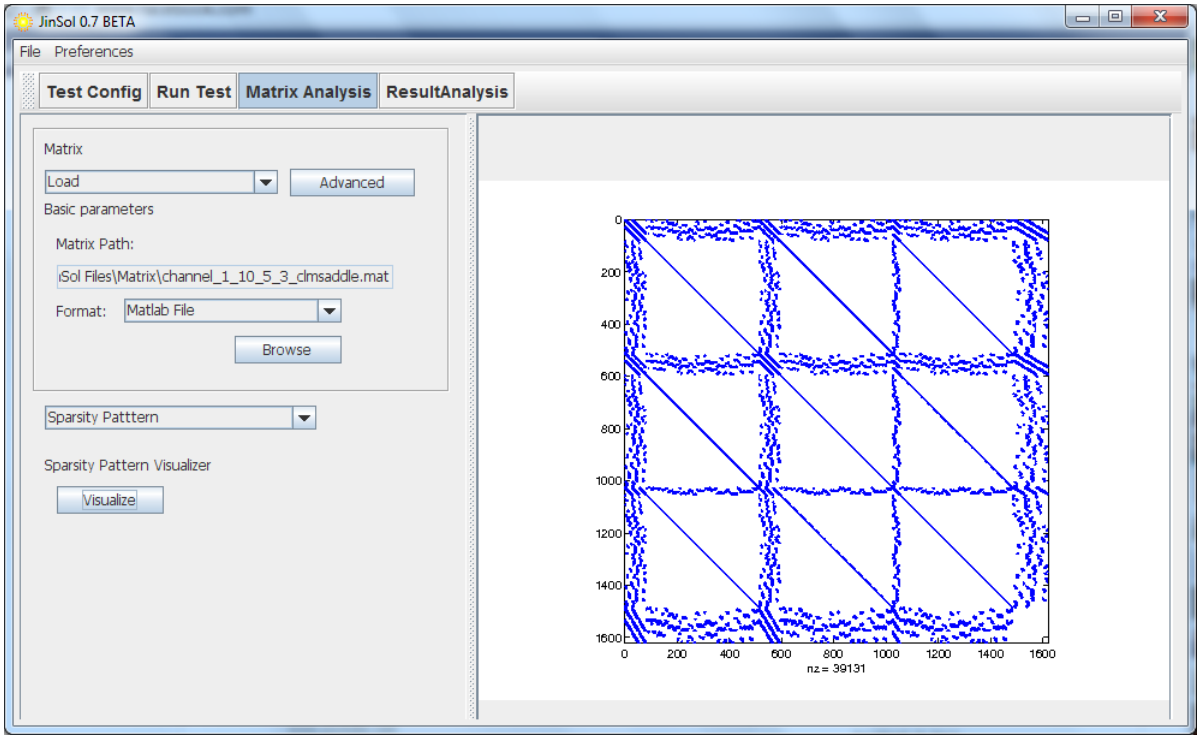


Figura 46 Solver Ges - Padrão de esparsidade do problema com CFL = 5

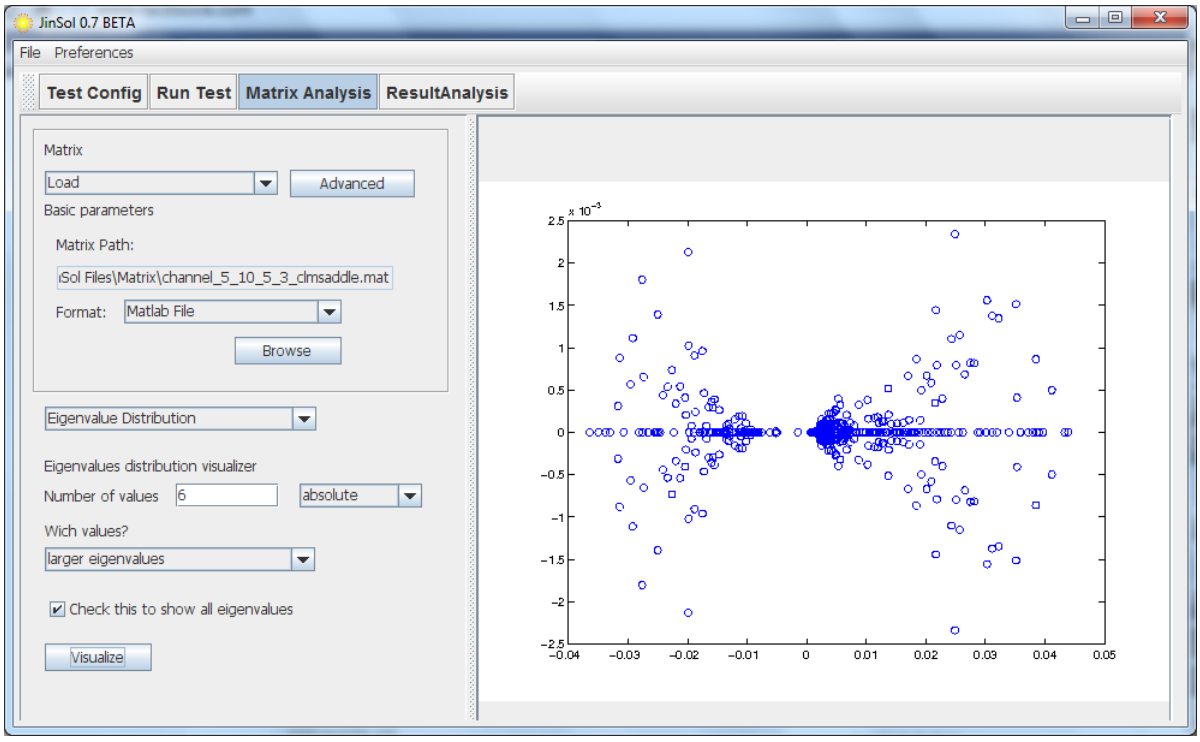


Figura 47 Solver Ges - Distribuição dos autovalores do problema no plano complexo no caso com CFL = 5

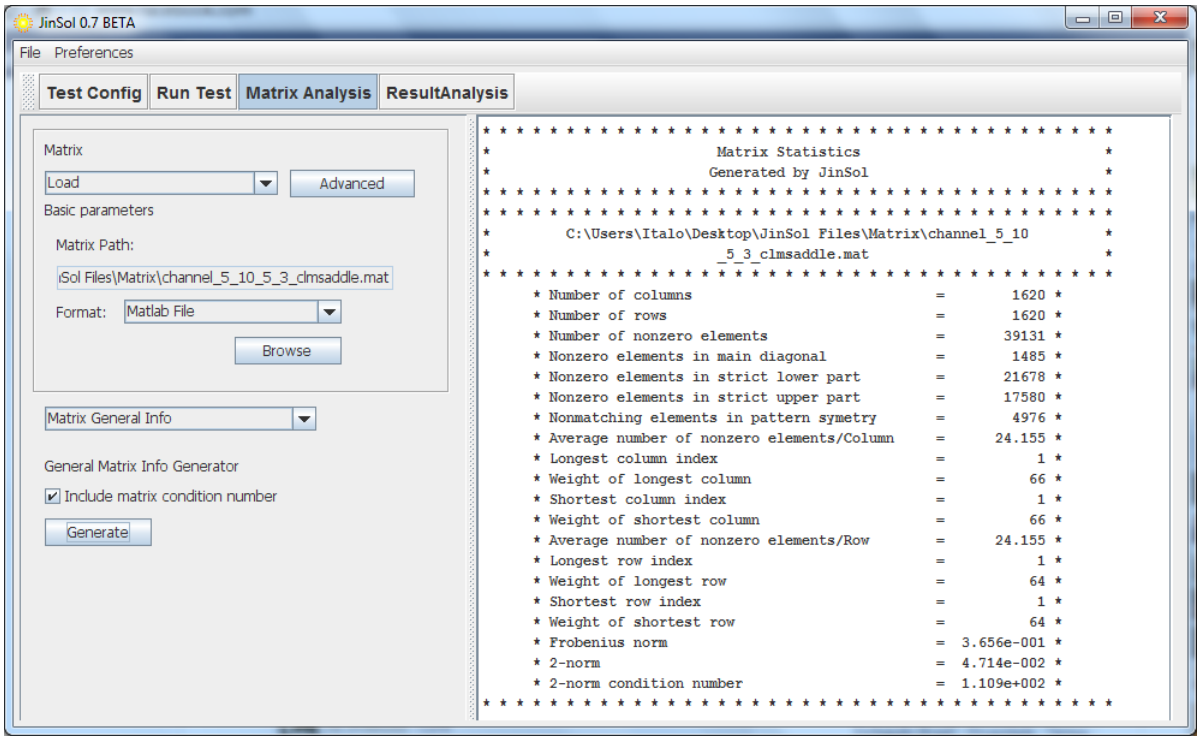


Figura 48 Solver Ges - Informações e estatísticas sobre a matriz do problema no caso com CFL = 5

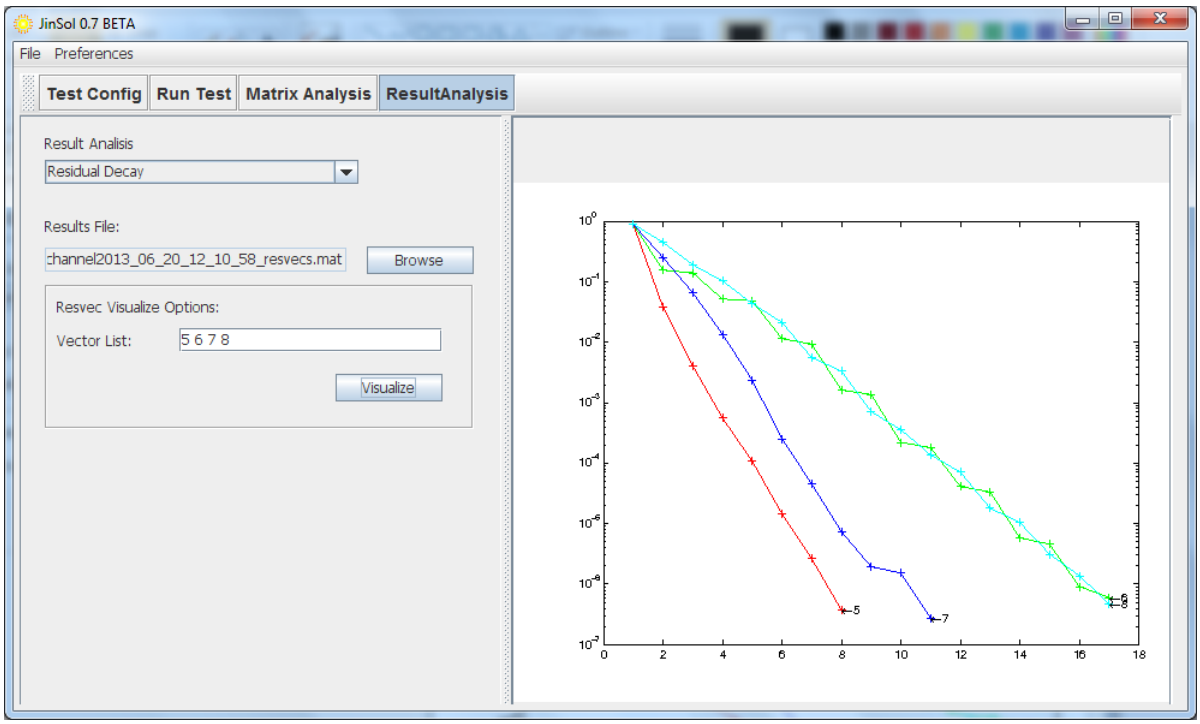


Figura 49 Basic Solver - Gráfico de decaimento do resíduo com GMRES precondicionado pela direita no caso com CFL = 1. O número de cada gráfico corresponde ao número do teste na tabela de resultados da Figura 38

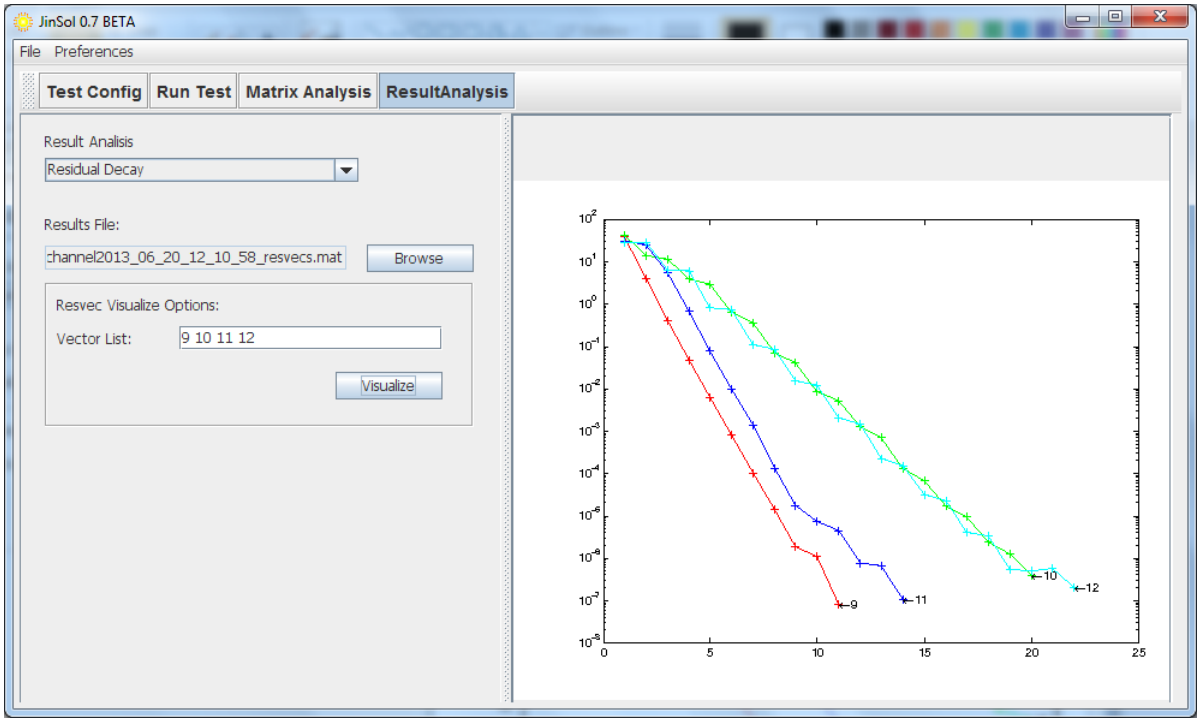


Figura 50 Basic Solver - Gráfico de decaimento do resíduo com GMRES precondicionado pela esquerda no caso com CFL = 5. O número de cada gráfico corresponde ao número do teste na tabela de resultados da Figura 38

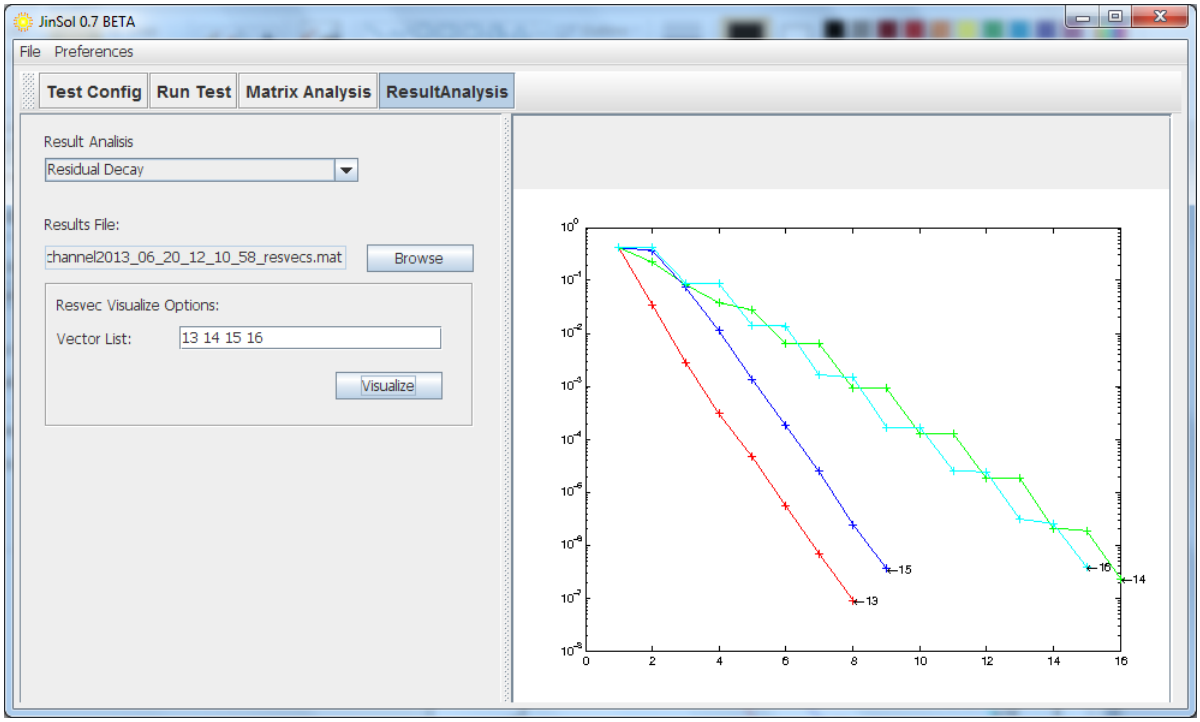


Figura 51 Basic Solver - Gráfico de decaimento do resíduo com GMRES precondicionado pela direita no caso com CFL = 5. O número de cada gráfico corresponde ao número do teste na tabela de resultados da Figura 38

7 APÊNDICE B - DADOS DO DESENVOLVIMENTO

Neste apêndice encontram-se algumas informações sobre o desenvolvimento e especificações técnicas do JinSol.

Especificações de desenvolvimento:

- Paradigma de programação: Orientação a objetos
- Linguagem: Java JDK 7
- IDE utilizada: Eclipse SDK
- Pacote de interface gráfica: Swing
- Sistemas operacionais testados: Windows e Linux
- Requerimentos: JRE 7 ou superior e Matlab R2008 ou superior.
- Idioma: Inglês

As imagens abaixo apresentam o diagrama de classes dos principais painéis da interface JinSol.

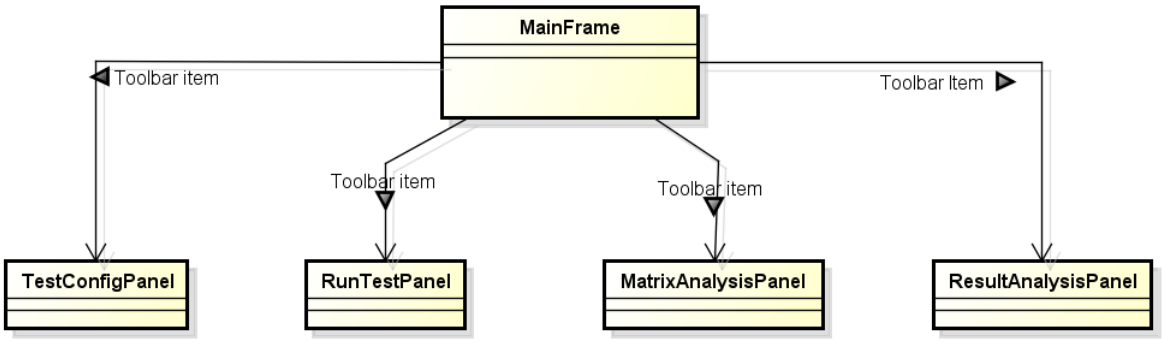


Figura 52 Diagrama básico de painéis da interface do JinSol

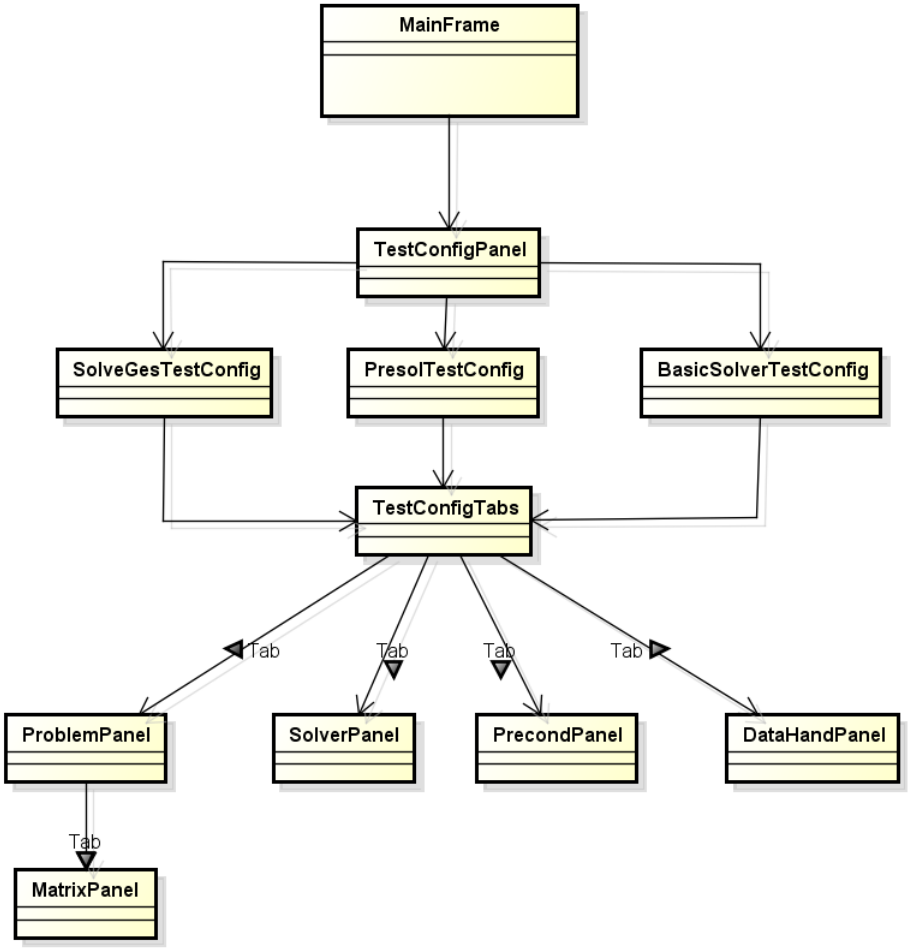


Figura 53 Diagrama extendido de painéis da interface do JinSol para configuração de testes

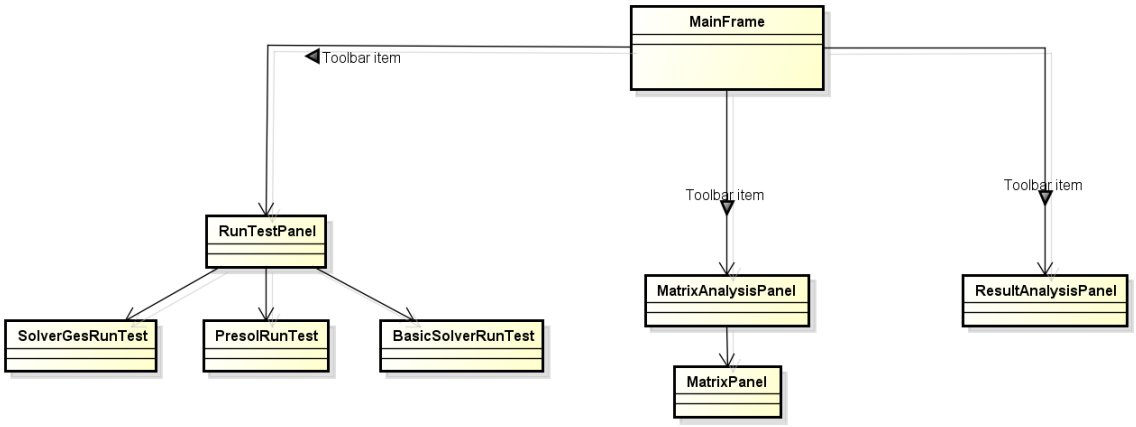


Figura 54 Diagrama básico de painéis da interface do JinSol sem configuração de testes