Solver de Alto Desempenho
para Problemas
na Forma Angular Blocada Aninhada

por
Fábio Nakano

Agradecimentos

A Julio Stern, amigo e orientador nas mais diversas áreas, e
M. Taube,
J. M. Martinez,
A. Fridlander,
C. A. B. Pereira,
C. Humes,
F. Coutinho,
J. F. Perez,
A. Plonski,
C. O. Ribeiro e
R. Milidui,
por muitas razões claras e outras igualmente importantes.

Conteúdo

1. Introdução	5
1.1. Sumário	5
1.1.1. Escopo do Projeto	5
1.2. O NOPEF	5
1.3. A Empresa UniSoma	6
1.4. Os Problemas de Otimização Multimisturas e Multiperíodo	7
1.5. ESPARSIDADE E ESTRUTURA	8
1.5.1. Esparsidade	8
1.5.2. Estrutura	8
1.5.3. Métodos de Decomposição vs. Métodos Estruturais	8
1.6. Evolução Técnica de Hardware e Software	9
1.7. Direitos Autorais	9
1.8. Avaliação do Mercado	10
1.9. Importância Estratégica e Riscos do Projeto	10
1.10. Quadro Geral das Tarefas de Desenvolvimento e Implementação	10
1.11. Algumas Considerações sobre o Desenvolvimento de Software de Alto Desempenho, e sua Inserção no Programa de Inovação Tecnológica	12
2. Definição dos problemas de otimização de misturas	14
2.1. Otimização de Múltiplas Mistura:	14
2.2. Otimização Multiperíodo	16
3. Método simplex	18
4. Fatoração QR	19
5. Forma Angular Blocada por linhas	22
5.1. fatoração QR	26
5.2. removendo linhas	27
5.3. inserindo linhas	30
5.4. reinversão	32
5.5. Simplex QR	32
5.6. implementação MATLAB do solver para a forma angular bloqueada	32
6. Forma Angular bloqueada aninhada	34
6.1. Fatoração QR	34
6.2. Inserindo linhas	39
6.3. Removendo linhas	42
6.4. implementação MATLAB	46
6.4.1. Reentrante	46
6.4.2. Angular bloqueada estendida	49
7. Codificação em linguagem C	52
7.1. densa	53
7.2. linear	54
7.3. QR	55
7.4. xrbaf	55
7.5. nrbafe	56
7.6. vetor	58
7.7. ivetor	60
8. Testes	61
9. Extensões ao trabalho apresentado	69
10. Bibliografia Básica	70
11. Bibliografia de Referência	70

Tabela de figuras

Figura 1 - Rotações aplicadas para obter R triangular superior	21
Figura 2 - Forma angular bloqueada por linhas	22
Figura 3 - Forma angular bloqueada por colunas	24

Figura 4 - Fatoração QR da forma angular blocada.....	26
Figura 5 - Estrutura da linha de Q	28
Figura 6 - Remoção de linha de A	29
Figura 7 - Inserção de linha	31
Figura 8 - Representação CBAF.....	33
Figura 9 - Fatoração dos blocos internos	35
Figura 10 - Fatoração do bloco diagonal externo.....	36
Figura 11 - Fatoração de todos os blocos diagonais externos.	37
Figura 12 - Fim do processo de fatoração da matriz RBAF.....	38
Figura 13 - Inserção de linha.....	39
Figura 14 - Triangularização da forma fatorada.....	40
Figura 15 - Triangularização da forma fatorada.....	41
Figura 16 - Remoção de linha.....	43
Figura 17 - Remoção de linha	44
Figura 18 - Remoção de linha	45
Figura 19 - Representação Reentrante	47
Figura 20 - Representação angular blocada estendida	50
Figura 21 - Diagrama da representação densa por linhas.....	54
Figura 22 - Diagrama da representação XRBAF	56
Figura 24 - Diagrama da representação Vetor	58
Figura 25 - Produto entre matriz NRBAF e vetor.....	59
Figura 26 - Diagrama da representação vetor de índices	61
Figura 27 - Padrão de esparsidade de um problema testado	62

1. Introdução

1.1. Sumário

Este trabalho foi desenvolvido como um projeto na área de software no Programa de Inovação Tecnológica da FAPESP. Este projeto, FAPESP-96/2341-2, teve como parceiros o NOPEF-USP, Núcleo de Otimização e Processos Estocásticos Aplicados à Economia e Finanças da Universidade de São Paulo, e a empresa cliente UniSoma. O Pesquisador responsável pelo projeto, Prof.Dr. Julio M. Stern, do DCC-IME-USP, e o Prof.Dr. Miguel Taube-Netto do IMECC-UNICAMP e também Presidente da UniSoma, co-orientaram o Engenheiro Fábio Nakano no desenvolvimento e adaptação dos algoritmos necessários, sua programação eficiente e sua implementação. Parte dos resultados obtidos são também apresentados como a Dissertação de Mestrado do Engenheiro Fábio Nakano no DCC-IME-USP.

1.1.1. Escopo do Projeto

Nosso objetivo é o desenvolvimento de solvers de Programação Matemática especialmente adaptados para Otimização de problemas com estrutura (forma) angular blocada por linhas (RBAF - Row Block Angular Form), e forma angular blocada por linhas aninhada (NRBAF - Nested Row Block Angular Form).

Este projeto adapta e implementa métodos estruturais desenvolvidos pelo pesquisador responsável. Estes métodos são apresentados em:

- J.M.Stern and S.A.Vavasis, *Active Set Methods for Problems in Column Block Angular Form*, *Comp. Appl. Math* v.12, pp.199-226, 1994.
- J.M.Stern, *Esparsidade Estrutura, Estabilidade e Escalonamento em Álgebra Linear Computacional*, IX Escola de Computação, Recife, 1994.

Estes solvers irão substituir solvers comerciais (OSL-IBM) nos softwares *MultiFor* e *DyFor* da empresa UniSoma. Estes softwares de aplicação industrial resolvem problemas de Otimização de Múltiplas Misturas, com estrutura RBAF, e problemas de Otimização Multiperíodo, com estrutura NRBAF. Outros pesquisadores já estiveram envolvidos no desenvolvimento de solvers para a UniSoma, como:

- L.R. de Souza Amaral, J. M. Martínez Pérez. *Um Algoritmo Estável para Resolução do Problema de Otimização de Rações*. IMECC-UNICAMP, 1987.
- F.A.M.Gomes, A.Fridlander, *O Método Simplex com Decomposição LU Aplicado a Problemas Blocos-Angulares*. *Pesquisa Operacional*, V-12 pp.63-74, 1994.

1.2. O NOPEF

O NOPEF - Núcleo de Otimização e Processos Estocásticos Aplicados à Economia e Finanças - é vinculado à Pró-Reitoria de Pesquisa, funcionando como órgão de apoio à pesquisa e à difusão de conhecimentos e de prestação de serviços à comunidade em geral. O NOPEF é um núcleo interdisciplinar,

congregando pesquisadores do Instituto de Matemática (IME), da Faculdade de Economia e Administração (FEA) e da Escola Politécnica (EP) da Universidade de São Paulo (USP).

São objetivos regimentais do NOPEF:

1. Desenvolver pesquisas em Economia, Finanças e Pesquisa Operacional que utilizem métodos de Programação Matemática, Otimização e análise estocástica em seus desenvolvimentos teóricos e práticos.
2. Fomentar a divulgação de resultados de pesquisas, bibliografias especializadas, artigos científicos, livros, relatórios e outros informes especializados.
3. Prestar serviços à comunidade, sob a forma de cursos, boletins, seminários, congressos, promoção de eventos, assessoria e consultoria, “softwares” e manuais explicativos das técnicas matemáticas utilizadas.

Capacitação para Projetos de Desenvolvimento Tecnológico

O NOPEF tem desenvolvido vários projetos em parceria com o setor privado. A nosso ver, o sucesso obtido pelo NOPEF com estes projetos de parceria deve-se em parte a observância de algumas normas:

1. Admitir apenas projetos em que haja substancial desenvolvimento e transferência de tecnologia, evitando projetos de consultoria baseadas em tecnologias relativamente bem conhecidas e difundidas.
2. Desenvolver o “produto” da consultoria até que este esteja em condições de ser realmente assimilado pelo cliente. Em especial um software deve apresentar uma interface “amigável” para o usuário, manual, help, e todos os demais requisitos que facilitem a boa aceitação do produto.
3. Estabelecer metas e cronogramas realistas, que obviamente devem ser cumpridos.

Um exemplo deste tipo de projeto foi o Convênio *BM&F* - NOPEF custeado inteiramente pela *Bolsa Mercantil e de Futuros* que resultou no desenvolvimento, por parte do Prof. Stern, do software *Critical-Point for Windows*. A versão estudantil deste software está disponível por *ftp* em www.ime.usp.br/~jstern. A versão profissional deste software é um produto comercial de sucesso, sendo utilizado por vários fundos de pensão e bancos de investimento no Brasil e no exterior.

1.3. A Empresa UniSoma

A UniSoma é uma empresa de consultoria e desenvolvimento de sistemas aplicados à logística, ao planejamento e programação (scheduling) da produção. Ela foi fundada em 1984 por pesquisadores de Pesquisa Operacional, Matemática Aplicada, Estatística e Computação da UNICAMP após longa experiência de seus fundadores na área acadêmica e em consultoria para empresas.

A capacitação principal da UniSoma é o desenvolvimento de modelos matemáticos para a tomada de decisões complexas em logística, no planejamento e programação da produção, com implementações computacionais próprias, condicionadas ao ambiente organizacional e técnico de cada cliente. A ação da UniSoma complementa sistemas comerciais de informática através da utilização matemática das informações disponíveis nestes sistemas. Esta abordagem se aplica a amplo espectro de situações de manufatura, indústrias de processo, agroindústria, etc.

Área de Atuação	Clientes
Planejamento integrado da produção no setor avícola	Sadia
Formulação otimizada de misturas (rações animais, coque em siderúrgicas, ligas, minérios, suco de laranja)	Sadia, Granja Saito, Perdigão, M.Cassab, Anderson Clayton, Usiminas, CST, Citropectina, etc.
Planejamento e programação da produção industrial (metalúrgica, poliéster, eletrônica, produtos de papel)	Brazaço-Mapri, Tilibra, Polyenka, Thomson, Tupe Components BHZE
Sequenciamento e dimensionamento de lotes de produção no setor automotivo	Fiat Automóveis
Minimização de perdas em cortes industriais (papel, aço)	Ripasa, Rigesa, Klabin, Papéis Simão, Pisa, Mannesmann
Planejamento de operações de transporte e distribuição (minério, suínos, pintinhos)	CVRD, Granja Rezende, Agrocerec
Planejamento florestal	CVRD, Belgo-Mineira, Cenibra

Reconhecimento Internacional

O trabalho desenvolvido na Sadia resultou na nomeação e classificação da Sadia como finalista do *1994 ORSA PRIZE* oferecido anualmente pela Operations Research Society of America.

Além disso, em 1995, a Sadia e a UniSoma foram ganhadores do prêmio *Franz Edelman Award*. Este prêmio, criado em homenagem a *Franz Edelman*, pioneiro no uso de métodos de *Management Science* na RCA, é concedido desde 1972. Entre os premiados incluem-se a *Tata Iron and Steel Ltd* (1994), a *AT&T* (1993), o *New Haven Health Department AIDS Division* (1992), e a *American Airlines* (1991).

Após a apresentação dos seis trabalhos premiados em 1995 perante uma comissão julgadora, foram distinguidos entre os ganhadores a *HARRIS CORPORATION*, que teve suporte metodológico da Universidade da Califórnia (Berkeley), e a *SADIA (UniSoma)*. Outros ganhadores em 1995 foram a *Força Aérea de Israel*; *Sainsbury's*, rede de supermercados da Inglaterra com faturamento anual de US\$ 15 bilhões; *NYNEX*, empresa telefônica de New York que opera em toda a região nordeste dos Estados Unidos; *Corporation*, produção de semicondutores; e *Key Corp*, um banco americano com 1300 agências.

1.4. Os Problemas de Otimização Multimisturas e Multiperíodo

O problema de determinar os percentuais dos ingredientes de uma mistura (ração, minério, sucos cítricos, etc.) sujeitos a restrições de qualidade (níveis nutricionais, teores de metais, teores de açúcares, acidez, cor), com otimização do valor (custo) de uma unidade de mistura, é uma aplicação clássica de programação linear.

Problemas agro-industriais típicos de multi-rações apresentam da ordem de 25 restrições por ração para 40 rações, perfazendo 1.000 restrições. Quando formulamos o problema multi-período temos tipicamente 12 períodos, ou da ordem de 12.000 restrições. Considerando um número de 30 ingredientes, diferenciados por origem em 10 regiões, temos da ordem de 150.000 variáveis. O detalhamento destes problemas é apresentado adiante.

No projeto desenvolvido pela UniSoma na Sadia Alimentos, o gasto anual com rações para aves na empresa era da ordem de 5E8US\$ (meio bilhão de Dólares). A formulação multi-mistura acarretou reduções

de custo da ordem de 3%, e a formulação multi-período da ordem de 3% adicionais, ou seja uma economia anual de 3E7US\$ (30 milhões de Dólares).

1.5. ESPARSIDADE E ESTRUTURA

1.5.1. Esparsidade

Grande parte do processamento de dados em ciência envolve computação numérica, e a maior parte desta computação numérica são rotinas básicas de álgebra linear. Várias aplicações em engenharia, física, pesquisa operacional, administração ou matemática, geram problemas lineares cujas matrizes são esparsas (entre tantos outros: resolução de equações diferenciais, análise de sistemas, circuitos ou estruturas, programação linear e não linear, otimização de fluxos em redes, etc...). Ademais, observa-se que a densidade destas matrizes decresce com a dimensão do problema: tipicamente apenas da ordem $n \cdot \log(n)$ dos n^2 elementos na matriz são não nulos. Assim, quanto maiores e mais complexas os problemas (e usuários sempre querem resolver modelos maiores), mais importante se torna usar eficientemente a esparsidade e a estrutura das matrizes envolvidas. Por exemplo, na área de otimização, problemas de aplicação industrial hoje considerados de médio porte envolvem milhares de equações, sendo a maioria destes problemas altamente estruturados, muito esparsos, e usando mais de 99% do tempo de resolução em rotinas básicas de álgebra linear.

1.5.2. Estrutura

No desenvolvimento de algoritmos especializados cabe distinguir dois conceitos:

1. Esparsidade: Como resolver eficientemente sistemas cujas matrizes tenham baixa densidade de elementos não nulos.
2. Estrutura: Como resolver eficientemente sistemas esparsos cujos elementos não nulos estão dispostos com uma certa regularidade na matriz de coeficientes.

Solvers comerciais, como por exemplo o OSL (Optimization Subroutine Library da IBM), utilizam eficientemente a esparsidade geral dos problemas de médio e grande porte; Todavia, a estrutura de um problema só pode ser bem aproveitada por um solver especialmente desenvolvido para uma classe de problemas com aquela estrutura. Ademais, o solver terá de saber a priori, ou então reconhecer automaticamente, detalhes desta estrutura (ex. dimensão dos blocos e sub-blocos) antes de iniciar a fase de resolução numérica do problema.

1.5.3. Métodos de Decomposição vs. Métodos Estruturais

Existem na literatura duas abordagens clássicas para resolução de problemas de otimização estruturados (especialmente para estrutura bloco angular): os Métodos de Decomposição e os Métodos Estruturais, cada qual apresentando vantagens distintas:

Métodos de Decomposição: Estes métodos mapeiam o problema original em vários sub-problemas acoplados. A estrutura do problema original é refletida na estrutura do acoplamento. No caso dos problemas bloco angulares, temos um sub-problema principal, ou “mestre” que devemos otimizar em função da solução

dos demais sub-problemas, os problemas “auxiliares”. Uma série de soluções iterativas mestre-auxiliares converge para a solução do problema original. Exemplos desta abordagem são os métodos de Dantzig-Wolf e Rosen; Este último foi utilizado no sistema *MultiFor* para resolver problemas bem menores que os resolvidos no sistema *DyFor* pelo solver OSL. Uma coletânea recente de trabalhos nesta área pode ser encontrada em - J.B.Rosen, Editor, 1990, *Supercomputers in Large-Scale Optimization*, Balzer AG, Basel.

Métodos Estruturais: Estes métodos resolvem diretamente o problema original. A estrutura do problema é aproveitada num nível mais baixo, preservando a estrutura e a esparsidade da “fatoração da base” no algoritmo de restrições ativas.

Comparativamente, os métodos de decomposição são bastante fáceis de implementar, sendo basicamente um “batch” ou “loop exterior” invocando um solver padrão. Todavia o procedimento iterativo, que costuma ser bastante eficiente nos primeiros estágios, tende a convergir lentamente no final. Também o “overhead” das chamadas ao solver no loop externo contribuem para diminuir a eficiência destes métodos. Os métodos estruturais são extremamente eficientes, porém de implementação muito trabalhosa, dependendo inclusive de sub-rotinas especiais de álgebra linear computacional. Métodos estruturais são especialmente adequados para problemas de Análise Paramétrica e Reotimização (“partida quente”) que corresponde a situação de uso típica na aplicação deste projeto.

1.6. Evolução Técnica de Hardware e Software

Em 1984 a UniSoma começou o desenvolvimento do sistema *MultiFor* para multiformulação de rações nos microcomputadores da época, com implementação própria do algoritmo de Rosen para problemas de programação linear, com estruturas bloco angulares. Com o aumento da capacidade computacional dos microcomputadores e workstations, novas perspectivas se abriram para a otimização de misturas ao longo de horizontes de vários períodos, com a finalidade de abranger o gerenciamento de compras de ingredientes, cujos preços variam ao longo do tempo. Foi, então, desenvolvido o sistema *DyFor* (“Dynamic Formulation”), que é uma extensão do *MultiFor*. O sistema *DyFor* utiliza presentemente um solver comercial para Programação Linear: a biblioteca OSL (Optimization Subroutine Library) da IBM. O projeto em tela visa substituir o uso do solver OSL por um solver desenvolvido especificamente para acomodar a estrutura do problema de otimização de misturas.

A evolução dos solvers dos sistemas *MultiFor* e *DyFor* acompanharam a evolução da tecnologia de hardware e software disponíveis. No início da década de 80 microcomputadores com coprocessadores dedicados para operações de ponto flutuante (o Intel 8087, FPU do Intel 8086 foi colocado no mercado em 1980) e com 640KB de memória principal permitiram a implementação de algoritmos simples de programação linear. Neste ambiente a solução tecnicamente viável para resolver problemas estruturados foi um método de decomposição: o algoritmo de Rosen. Na década seguinte, o aumento de performance dos processadores, a crescente sofisticação dos sistemas de gerenciamento de memória dos microcomputadores, e principalmente a sofisticação das ferramentas de desenvolvimento de software, como a programação orientada a objetos, permitiram implementar solvers baseados em métodos estruturais.

1.7. Direitos Autorais

Os direitos autorais da biblioteca de classes e funções de álgebra linear computacional para matrizes esparsas e estruturadas, bem como do solver, pertencem exclusivamente ao pesquisador responsável (respeitado o regimento da USP). A UniSoma terá o direito de livremente comercializar executáveis do solver

incorporados aos softwares *MultiFor* e *DyFor*, ou produtos que venham a substituí-los. Os direitos autorais e patentes dos softwares *MultiFor* e *DyFor*, ou produtos que venham a substituí-los, pertencem exclusivamente a empresa UniSoma.

1.8. Avaliação do Mercado

Os sistemas de formulação de rações *MultiFor* e *DyFor* vem tendo excelente aceitação no mercado. O sistema *MultiFor* tem atualmente 26 cópias vendidas a clientes no setor agro-industrial, e o sistema *DyFor*, recentemente desenvolvido, 3 cópias. Outra característica interessante destes sistemas é a possibilidade de tratar outros problemas de misturas, como a recente aplicação desenvolvida para a Companhia Vale do Rio Doce, otimizando a formação de pilhas de estocagem por camadas de minérios. É grande o potencial de colocar o *DyFor* no mercado internacional, por apresentar inovações inéditas em relação aos sistemas concorrentes, já estando em estudo o marketing para o mercado Norte-Americano. A substituição do solver externo (IBM-OSL) pelo solver integrado a ser desenvolvido neste projeto ampliará sobremaneira o mercado potencial do sistema e em muito facilitará sua comercialização.

1.9. Importância Estratégica e Riscos do Projeto

Consideramos o projeto de **importância estratégica para o país**, dada a dependência em relação a muito poucos fornecedores de solvers de alto desempenho, nenhum deles nacional. Consideramos **grande o poder fertilizante** do projeto nas áreas de Pesquisa Operacional, Economia, Finanças, Matemática Aplicada e Engenharia de Software, com possíveis desdobramentos na área de Computação Paralela. Presentemente existem vários programas comerciais ou de domínio público implementando métodos de decomposição para problemas RBAF. Todavia conhecemos apenas protótipos de solvers estruturais para problemas RBAF, enquanto desconhecemos sequer protótipos para problemas NRBAF. Assim, serão **grandes as inovações tecnológicas** decorrentes da utilização dos novos conceitos de métodos estruturais.

Desenvolver este solver específico visando alcançar ou superar o desempenho do solver IBM-OSL, desenvolvido ao longo de décadas (antigo IBM-MPSX), é um grande desafio (**alto risco tecnológico**). Visto que os produtos *MultiFor* e *DyFor* já são comercializados com grande sucesso, o êxito da comercialização dos mesmos com um solver integrado está garantido (**baixo risco de comercialização**). As estruturas RBAF e NRBAF ocorrem em importantes aplicações financeiras, como a otimização multi-período de múltiplos portfólios, e é nossa intenção futuramente desenvolver produtos para este mercado, inclusive para exportação (**alto poder germinativo de novos negócios**).

1.10. Quadro Geral das Tarefas de Desenvolvimento e Implementação

As Tarefas Básicas de programação tiveram a seguinte organização e cronograma:

1. Programação da biblioteca de classes e funções para matrizes esparsas e-ou estruturadas em C++.
2. Programação dos solvers de otimização por métodos de restrições ativas para problemas de programação matemática com estrutura RBAF e NRBAF em Matlab e C++

O cronograma físico das tarefas básicas 1 e 2 é detalhado a seguir. Cada uma das tarefas básicas foi dividida em sub-tarefas, para efeito de detalhamento e medição dos trabalhos, e a evolução planejada dos trabalhos é apresentada em 10 trimestres.

- 1.1- Definição de classes para matrizes com representação: densa, simétrica, triangular inferior e superior, Hessemberg, estática e dinâmica, por linha e coluna, blocada e bloco-aninhada.
- 1.2- Funções para fatoração LU densa e pivoteamento parcial e escalamento automático.
- 1.3- Funções para pré-posicionamento de pivos, P3.
- 1.4- Funções de partição e P4.
- 1.5- Funções de eliminação assimétrica esparsa.
- 1.6- Fatoração de Cholesky e QR densas.
- 1.7- Funções para ordenamento por dissecção.
- 1.8- Funções de eliminação simétrica e ortogonais esparsas.
- 1.9- Heurísticas para critérios de entrada na base do simplex.
- 2.1- Protótipo Matlab para a função Simplex, com especificação de base inicial, reinversão, e regras para criação da lista de entrada e funções para mudança de base densa.
- 2.2- Função simplex densa na biblioteca de classes e funções C++
- 2.3- Protótipo Matlab da função simplex blocada.
- 2.4- Funções para mudança de base blocada.
- 2.5- Função simplex blocada da biblioteca C++.
- 2.6- Funções de criação de listas de entrada e testes com problemas multimisturas.
- 2.7- Funções de mudança de base bloco-aninhada.
- 2.8- Funções para mudança de base e fatoração blocada-esparsa.
- 2.9- Função simplex bloco-aninhada e teste com problemas multiperíodo.

Cronograma Físico de Sub-Tarefas por Trimestre

	1	2	3	4	5	6	7	8	9	10
1.1	x	x	x							
1.2		x	x							
1.3			x	x						
1.4				x	x					
1.5					x	x	x			
1.6					x	x				
1.7							x	x		
1.8							x	x	x	
1.9									x	x
2.1	x									
2.2	x	x								
2.3		x	x	x						
2.4		x	x	x						
2.5				x	x					
2.6					x	x	x			
2.7					x	x	x	x	x	
2.8							x	x	x	
2.9								x	x	x

1.11. Algumas Considerações sobre o Desenvolvimento de Software de Alto Desempenho, e sua Inserção no Programa de Inovação Tecnológica

Projetos de inovação tecnológica na área de software tem características próprias, nem sempre comparáveis com outras áreas. É oportuno ressaltar e tecer alguns comentários a respeito de duas destas características:

1. Esclarecer o grande caminho a ser percorrido desde um algoritmo fruto de um projeto de pesquisa acadêmica até o desenvolvimento de um produto comercial.
2. A posição “labor intensive” da indústria de software, e o conseqüente interesse estratégico na aplicação de recursos para inovação tecnológica em projetos de desenvolvimento de software.

O desenvolvimento de um produto de software é distinto da implementação de um protótipo, e ainda mais distinto de um programa de alto nível para teste numérico de um algoritmo fruto de pesquisa acadêmica. O programa de alto nível para efeito de teste numérico de um algoritmo é usualmente o trabalho de pesquisa de um aluno de pós-graduação levando a uma tese de mestrado ou a um artigo conforme a sofisticação do algoritmo e sua implementação, e o grau de inovação introduzido. Obviamente isto não inclui o desenvolvimento do algoritmo em si, usualmente resultado da pesquisa de um aluno de doutorado ou do orientador.

A transformação do programa para testes numéricos em um protótipo inclui a definição e uso de estruturas de dados eficientes, gerenciamento eficiente de entrada e saída de dados, gerenciamento eficiente de memória e caches, etc. No projeto em tela esta transformação também inclui a troca da linguagem de alto nível na qual foi programado o algoritmo para testes numéricos, Matlab, para linguagens adequadas a implementação do protótipo, no caso C e C++. Nesta fase, há alto envolvimento do pesquisador sênior para definição e orientação das linhas mestras da implementação, gerando uma enorme quantidade de trabalho de programação a ser realizado. Devido a sofisticação existente nestas definições e sua codificação, espera-se que este trabalho seja parte central de uma dissertação de mestrado.

Finalmente, temos que considerar a transformação do protótipo em produto. Nesta fase, fora do escopo do presente projeto, o código do protótipo deve ser otimizado, considerando inclusive os detalhes de arquitetura da plataforma (hardware), compilador, etc, assim como devem ser definidas e desenvolvidas as diversas interfaces entre módulos (ou softwares) que compõe o produto, interface de usuário, recursos para manipulação de dados, visualização e análise, etc. Ao contrário do que alguns pesquisadores em métodos numéricos podem crer, esta é a fase mais demorada e de maior custo de mão de obra. Esta mão de obra é de perfil distinto do perfil acadêmico, porém de alta qualificação. Esta é também a fase em que ocorre a maior transferência tecnológica da instituição de pesquisa para a indústria. Esta fase demanda um grande esforço de integração, de modo a tornar harmoniosa a absorção da nova tecnologia ao produto ou processo previamente existente na indústria. Nesta fase é vital manter sob controle as várias tarefas exigidas nesta absorção, embora inevitavelmente este processo demande uma grande (em relação as fases mais acadêmicas) quantidade de mão de obra e recursos. Esta fase será objeto de um projeto complementar, uma vez concluído com sucesso o protótipo do solver.

O coordenador deste projeto já teve uma experiência muito bem sucedida no desenvolvimento de um produto de software (Critical-Path for Windows) para o mercado financeiro. O desenvolvimento deste produto foi inteiramente financiado pela BM&F - Bolsa de Mercadorias e de Futuros - ao longo de 3 anos. É interessante notar que o protótipo do algoritmo básico do produto (um solver de programação quadrática paramétrica suficientemente estável para trabalhar com problemas estruturados de grande porte com matrizes de covariância com autovalores muito pequenos) foi programado em alto nível (Matlab) em 4 meses pelo coordenador, e prototipado em C++ em aproximadamente 18 meses por 1 programador. Todavia este produto só obteve aceitação comercial quando as interfaces de dados e de usuário corresponderam às expectativas dos

profissionais do mercado financeiro. O desenvolvimento destas interfaces demandaram 2 programadores por 3 anos, consumindo da ordem de 70% dos recursos do projeto.

Sabidamente, a indústria de software tem algumas características marcantes:

1. A indústria de software é “labor intensive”, e não “capital intensive”.
2. A qualificação da mão de obra é fator determinante para o sucesso desta indústria.
3. mercado de software de uso geral é dominado pelos detentores dos melhores canais de distribuição, notadamente empresas multinacionais.

As características 1 e 2 explicam o sucesso, real ou potencial, de países cuja mão de obra tem baixo coeficiente custo/benefício. Pelo lado do custo nota-se uma tendência mundial pelo deslocamento de um grande número de posições de trabalho para fora de países do 1º mundo. Pelo lado do benefício vemos a concentração destas posições em países que por tradição cultural (ex. Índia e países do leste Europeu) ou consistente política de desenvolvimento (ex. Brasil) tem mão de obra qualificada em relativa abundância. No caso do Brasil tivemos em décadas passadas um represamento deste potencial devido a proteção de mercado na área de hardware, que em contraste com a área de software, está no extremo “capital intensive”. A característica 3 explica a existência de um nicho de mercado particularmente promissor na área de softwares com grau de especificidade médio, com os softwares de nossa empresa parceira, a UniSoma. Não por acaso a UniSoma começa a exportar com seus softwares agro-industriais para o mercado Americano.

No projeto em tela desenvolveu-se um protótipo que, uma vez incorporado aos softwares de modelagem matemática e otimização *MultiFor* e *DyFor*, aumentará o potencial competitivo dos produtos da UniSoma por melhorar seu desempenho e por torna-los independentes de fornecedores de componentes (solvers) de tecnologia fechada.

2. Definição dos problemas de otimização de misturas

2.1. Otimização de Múltiplas Mistura:

Este é o problema resolvido pelo programa *MultiFor*

Dados

j	índice dos ingredientes	
i	índice dos nutrientes	
l	índice das equações de nutrientes	
p	índice das equações de ingredientes	
k	índice das rações	
NR	conjunto das rações	
c _j	preço do ingrediente j	
R _k	quantidade a ser produzida da ração k	
a _{ij}	quantidade do nutriente i proveniente do ingrediente j (dados da	matriz nutricional)
B _{jk}	quantidade máxima do ingrediente j na ração k	
b _{jk}	quantidade mínima do ingrediente j na ração k	
Beta _{ik}	quantidade máxima do nutriente i na ração k	
Alfa _{ik}	quantidade mínima do nutriente i na ração k	
F _i	fator que multiplica o nutriente i da equação de nutriente	
F _j	fator que multiplica o ingrediente j da equação de ingrediente	
e _{lk}	quantidade máxima da equação de nutriente l na ração k	
d _{lk}	quantidade mínima da equação de nutriente l na ração k	
e _{pk}	quantidade máxima da equação de ingrediente p na ração k	
d _{pk}	quantidade mínima da equação de ingrediente p na ração k	
e _p	quantidade máxima da equação de ingrediente p no lote	
d _p	quantidade mínima da equação de ingrediente p no lote	
E _j	quantidade máxima total do ingrediente j	
D _j	quantidade mínima total do ingrediente j	

Variável de decisão

x_{jk} quantidade do ingrediente j na ração k

Restrições

1. Quantidade a ser fabricada da ração k

$$\sum_{j=1}^n x_{jk} = R_k \quad \forall k$$

1. Quantidade do ingrediente j na ração k (máxima e mínima)

$$100 \cdot \frac{x_{jk}}{\sum_{j=1}^n x_{jk}} \leq B_{ik} \quad \forall j, k$$

$$100 \cdot \frac{x_{jk}}{\sum_{j=1}^n x_{jk}} \geq b_{ik} \quad \forall j, k$$

3. Quantidade do nutriente i na ração k (máxima e mínima)

$$\frac{\sum_{j=1}^n a_{ij} x_{jk}}{\sum_{j=1}^n x_{jk}} \leq Beta_{ik} \quad \forall i, k$$

$$\frac{\sum_{j=1}^n a_{ij} x_{jk}}{\sum_{j=1}^n x_{jk}} \geq Alfa_{ik} \quad \forall i, k$$

4. Equação de nutrientes (quantidade máxima e mínima)

$$\frac{\sum_{i_1 \in l} F_{i_1} \sum_{j=1}^n a_{i_1 j} x_{jk}}{\sum_{i_2 \in l} F_{i_2} \sum_{j=1}^n a_{i_2 j} x_{jk}} \leq e_{lk} \quad \forall l, k$$

$$\frac{\sum_{i_1 \in l} F_{i_1} \sum_{j=1}^n a_{i_1 j} x_{jk}}{\sum_{i_2 \in l} F_{i_2} \sum_{j=1}^n a_{i_2 j} x_{jk}} \geq d_{lk} \quad \forall l, k$$

5. Equação de ingredientes (quantidade máxima e mínima) na fórmula

$$\frac{\sum_{j_1 \in p} F_{j_1} x_{j_1 k}}{\sum_{j_2 \in p} F_{j_2} x_{j_2 k}} \leq e_{pk} \quad \forall p, k$$

$$\frac{\sum_{j_1 \in p} F_{i_1} x_{j_1 k}}{\sum_{j_2 \in p} F_{j_2} x_{j_2 k}} \geq d_{pk} \quad \forall p, k$$

6. Equação de ingredientes (quantidade máxima e mínima) no lote (conjunto de fórmulas)

$$\frac{\sum_{j_1 \in p} F_{i_1} x_{j_1 k} \sum_{k=1}^{NR} x_{j_1 k}}{\sum_{j_2 \in p} F_{j_2} x_{j_2 k} \sum_{k=1}^{NR} x_{j_2 k}} \leq e_p \quad \forall p$$

$$\frac{\sum_{j_1 \in p} F_{i_1} x_{j_1 k} \sum_{k=1}^{NR} x_{j_1 k}}{\sum_{j_2 \in p} F_{j_2} x_{j_2 k} \sum_{k=1}^{NR} x_{j_2 k}} \geq d_p \quad \forall p$$

7. Disponibilidade do ingrediente no lote (máxima e mínima)

$$\sum_{k=1}^{NR} x_{jk} \leq E_j \quad \forall j$$

$$\sum_{k=1}^{NR} x_{jk} \geq D_j \quad \forall j$$

Função Objetivo: Minimizar o custo total

$$c = \min \left(\sum_{j=1}^n c_j \left(\sum_{k=1}^{NR} x_{jk} \right) \right)$$

2.2. Otimização Multiperíodo

Este é o problema resolvido pelo programa *DyFor*

As definições e as restrições descritas acima para Otimização de Mistura valem também para otimização Multiperíodo. Abaixo, citamos apenas os dados, as variáveis e as restrições que devem ser adicionadas ou alteradas, aquelas já citadas.

Dados

t	índice dos períodos
T	conjunto dos períodos
c_{jt}	preço do ingrediente j no período t
s_{jt}	custo de estocagem do ingrediente j no período t
S	conjunto dos ingredientes estocáveis
θ	taxa de desconto anual
Δt	número de dias decorridos desde o início do planeamento
ymx_{jt}	compra máxima do ingrediente j no período t
ymn_{jt}	compra mínima do ingrediente j no período t
wmx_{jt}	estoque máximo do ingrediente j no período t
wmn_{jt}	estoque mínimo do ingrediente j no período t
Gmx_t	gasto máximo do período t

Variáveis de decisão

x_{jkt}	quantidade do ingrediente j na ração k no período t
y_{jt}	compra do ingrediente estocável j no período t
w_{jt}	estoque do ingrediente j no período t

Restrições

1. Balanceamento de ingredientes estocáveis

$$w_{j,(t-1)} + y_{jt} = \sum_{k=1}^{NR} x_{jkt} + w_{jt} \quad \forall j \in S, t$$

2. Quantidade de compra de ingredientes (máxima e mínima)

$$\begin{aligned} y_{jt} &\leq ymx_{jt} & \forall j \in S, t \\ y_{jt} &\geq ymn_{jt} & \forall j \in S, t \end{aligned}$$

3. Quantidade de estoque de ingredientes (máxima e mínima)

$$\begin{aligned} w_{jt} &\leq wmx_{jt} & \forall j \in S, t \\ w_{jt} &\geq wmn_{jt} & \forall j \in S, t \end{aligned}$$

4. Gasto máximo por período

$$\sum_{j \in S} \left(c_{jt} \sum_{k=1}^{NR} x_{jkt} \right) + \sum_{j \in S} (c_{jt} y_{jt} + s_{jt} w_{jt}) \leq Gmx_t \quad \forall t$$

Função Objetivo: Minimizar o custo total

$$c = \min \left(\frac{\sum_{t=1}^T \left(\sum_{j \in S} \left(c_{jt} \sum_{k=1}^{NR} x_{jkt} \right) + \sum_{j \in S} (c_{jt} y_{jt} + s_{jt} w_{jt}) \right) \right)}{(1+\theta)^{\Delta t}} \right)$$

3. Método Simplex

Os problemas descritos na seção anterior são problemas de programação linear. Para fixar a notação segue uma descrição resumida do problema e do algoritmo que vamos implementar: Métodos de restrições ativas para problemas não lineares podem também ser construídos sobre a mesma estrutura básica.

minimizar uma função linear dentro de um poliedro padrão:

$$\min(cx), x \geq 0 \mid Ax = d$$

que, permutando a fim de agrupar as colunas da base pode ser visto como:

$$\min \left(\begin{bmatrix} c^b & c^r \end{bmatrix} \begin{bmatrix} x^b \\ x^r \end{bmatrix} \right), x \geq 0 \mid \begin{bmatrix} B & R \end{bmatrix} \begin{bmatrix} x^b \\ x^r \end{bmatrix} = d$$

usando como notação $\tilde{d} \equiv B^{-1}d$, $\tilde{R} \equiv B^{-1}R$, se mudarmos um elemento $x_j^r \in x^r$, a solução básica, x^b torna-se:

$$x^b = \tilde{d} - \tilde{R}x^r = \tilde{d} - x_j^r \tilde{R}_{\cdot,j}$$

que é viável enquanto não negativa. Pela hipótese de não degenerescência, $\tilde{d} > 0$, podemos aumentar o valor de x_j^r , mantendo a solução básica viável até um patamar $\varepsilon(j) > 0$.

O valor da função objetivo desta solução básica é:

$$\begin{aligned} cx &= c^b x^b + c^r x^r = \\ &= c^b B^{-1}(d - R x^r) + c^r x^r = \\ &= c^b \tilde{d} + (c^r - c^b \tilde{R}) x^r \equiv \\ &\equiv \varphi - z x^r = \varphi - z_j x_j^r \end{aligned}$$

onde z é o custo reduzido na base.

e o algoritmo é o seguinte:

1-) Procure um índice residual j tal que $z_j > 0$;

2-) Compute, para $k \in K \equiv \{l \mid \tilde{R}_{l,j} > 0\}$, $\varepsilon_k = \frac{\tilde{d}_k}{\tilde{R}_{k,j}}$, e $i = \arg \min_{k \in K} (\varepsilon_k)$

3-) Faça x_j^r básica e x_i^b residual.

4-) Volte ao passo 1

O algoritmo não pode prosseguir se $z \leq 0$ ou K for vazio. O primeiro caso corresponde a termos achado a solução ótima e o segundo a termos um problema ilimitado.

A cada passo do simplex precisamos re-obter a inversa da base. A cada passo do simplex, a base é modificada mediante uma troca de colunas, uma pequena alteração. "Atualizar a inversa da base" significa, com pouco trabalho adicional, recomputar a inversa da nova base, a partir da inversa da base anterior. Podemos atualizar a inversa de base diretamente usando, por exemplo, as formulas de Sherman-Morrisson. Todavia, devido as más propriedades de estabilidade numérica desta abordagem, é preferível manter, ao invés da inversa, uma fatoração da base como $B=LU$ ou $B=QR$, onde L (U) indica uma matriz triangular inferior (superior) e Q uma matriz ortogonal. Mantendo estas fatorações podemos fazer todos os cálculos necessários no simplex, isto é resolver sistemas lineares cuja matriz de coeficientes é a base ou sua transposta. Por abuso de linguagem freqüentemente nos referimos à fatoração da base como sua "inversa", uma vez que a fatoração é usada como sua sucedânea.

Nas próximas seções mostraremos como a manter eficientemente a fatoração QR de uma base na forma angular blocada, e também como preservar esta estrutura ao atualizar a fatoração ao longo das sucessivas mudanças de base. A possibilidade de preservar a estrutura da base em seus fatores, e também as boas qualidades de estabilidade numérica desta fatoração, justificam o seu uso.

4. Fatoração QR

Dada uma matriz real $A(m \times n), m \geq n$, existe uma matriz Q , ortogonal (ie. $Q^{-1} = Q'$) tal que $A = Q \begin{bmatrix} R \\ 0 \end{bmatrix}$, onde R é uma matriz quadrada e triangular superior. Esta decomposição é uma fatoração QR de

A. Para chegar a essa decomposição, considere a rotação de um vetor por um ângulo θ no $\mathbb{R}^2$:

$$rot(\theta)x = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

note que a rotação é uma transformação ortogonal:

$$\begin{aligned} rot(\theta)rot(\theta)' &= \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} = \\ &= \begin{bmatrix} \cos^2(\theta) + \sin^2(\theta) & \cos(\theta)\sin(\theta) - \sin(\theta)\cos(\theta) \\ \cos(\theta)\sin(\theta) - \sin(\theta)\cos(\theta) & \cos^2(\theta) + \sin^2(\theta) \end{bmatrix} = I_2 \end{aligned}$$

e o produto de transformações ortogonais continua ortogonal:

$$\begin{aligned} Q &= Q_1 Q_2 \cdot \\ QQ' &= Q_1 Q_2 Q_2' Q_1' = I \end{aligned}$$

logo podemos usar um conjunto de rotações para levar A a triangular superior.

A rotação de Givens estende a operação de rotação para o $\mathbb{R}^n$:

$$G(i, j, \theta) = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & \cos(\theta) & & \sin(\theta) & \\ & & & \ddots & & \\ & & -\sin(\theta) & & \cos(\theta) & \\ & & & & & \ddots \\ & & & & & & 1 \end{bmatrix}$$

dizemos que $G(i, j, \theta)A$ roda as linhas i e j de A de um ângulo θ .

Podemos aplicar uma sequência de rotações de Givens a A de forma a anular elementos de A , obtendo R , triangular superior. Como ilustramos na Figura 1.

Observe que permutações de linhas são um caso particular do operador de rotação. Outras fatorações de matriz utilizam outras operações básicas para obter um fator triangular (ex. Operações elementares na fatoração LU). A simplicidade de incluir permutações no fator ortogonal da fatoração QR é a essência do tratamento de problemas estruturados que adotamos.

Como Q é uma matriz ortogonal, não precisamos dela para resolver um sistema linear $Ax = d$ ou, explicitando a fatoração, $QRx = d \Rightarrow x = B^T U^{-1} U^{-T}$. Logo, é desnecessário armazenar essa matriz e recalculá-la seus elementos quando inserimos ou removemos uma linha da base e ainda podemos calcular qualquer linha de Q quando for necessário.

Figura 1 - Rotações aplicadas para obter R triangular superior

5. Forma Angular Blocada

Descreveremos nesta seção a forma angular blocada, BAF, por linhas e por colunas, RBAF e CBAF, e mostraremos como esta estrutura surge naturalmente em problemas de otimização. Além de problemas que apresentam esta estrutura naturalmente, existem heurísticas que, por permutações, aproximam matrizes esparsas, aparentemente não BAF, de matrizes BAF [Stern3]. Mais adiante veremos como esta estrutura aparece também de forma aninhada, tanto por linhas quanto por colunas, NRBAF e NCBAF.

Uma matriz está na forma angular blocada por linhas quando contém uma série de blocos diagonais (independentes) e um conjunto de linhas acoplando esses blocos (Figura 2).

Figura 2 - Forma angular blocada por linhas

Existem muitos problemas que podem ser representados nesta forma em sua maioria compostos por sub-sistemas conectados. Como exemplo de problema RBAF citamos o problema de produção e estoque [Lasdon1] onde um industrial produz dois produtos com previsão trimestral de demanda e o industrial tem que determinar quanto produzir e estocar durante o ano minimizando o custo de produção e estoque. Existem restrições quanto à capacidade de trabalho e estoque disponíveis. A formulação do problema é a seguinte:

$x_{i,t}$ quantidade do i -ésimo produto produzida no período t .

$y_{i,t}$ quantidade do i -ésimo produto estocada no período t .

$s_{1,t}$ estoque ocioso no período t .

$s_{2,t}$ trabalho ocioso no período t .

as equações que definem o problema para um período são:

I-) Quantidade de trabalho:

$$b_1 x_{1,t} + b_2 x_{2,t} + s_{2,t} = l_t$$

onde b é o trabalho necessário para produzir uma unidade do produto x e o trabalho disponível no período é l .

II-) Capacidade de estoque:

$$a_1 x_{1,t} + a_2 x_{2,t} + s_{1,t} = f_t$$

onde a é o espaço necessário para estocar o produto x e o espaço total disponível no período é f .

III-) estoque acopla os períodos:

$$y_{i,t} = y_{i,t-1} + x_{i,t} - d_{i,t}$$

d é a demanda estimada do produto.

IV-) custo:

$$z_t = v_{1,t} y_{1,t} + v_{2,t} y_{2,t} + c_{1,t} b_1 x_{1,t} + c_{2,t} b_2 x_{2,t}$$

v é o custo unitário de estoque e c é o preço da unidade de trabalho.

O sistema que descreve o problema para um período fica:

$$\begin{bmatrix} & & & b_1 & b_2 & 1 \\ & a_1 & a_2 & & & 1 \\ -1 & & 1 & & -1 & \\ & -1 & & 1 & & -1 \\ & & & & & \end{bmatrix} \begin{bmatrix} y_{1,t-1} \\ y_{2,t-1} \\ x_{1,t} \\ x_{2,t} \\ y_{1,t} \\ y_{2,t} \\ s_{1,t} \\ s_{2,t} \end{bmatrix} = \begin{bmatrix} l_t \\ f_t \\ -d_{1,t} \\ -d_{2,t} \end{bmatrix}$$

rearranjando as colunas temos:

$$\begin{bmatrix} b_1 & b_2 & 1 & & & & & \\ & & & a_1 & a_2 & 1 & & \\ -1 & & & 1 & & & -1 & \\ & -1 & & & 1 & & & -1 \end{bmatrix} \begin{bmatrix} y_{1,t} \\ y_{2,t} \\ s_{2,t} \\ x_{1,t} \\ x_{2,t} \\ s_{1,t} \\ y_{1,t-1} \\ y_{2,t-1} \end{bmatrix} = \begin{bmatrix} l_t \\ f_t \\ -d_{1,t} \\ -d_{2,t} \end{bmatrix}$$

Podemos trabalhar também com a forma tranposta: angular blocada por colunas(Figura 3). Precisamos nos preocupar com a fatoração QR de uma matriz na forma angular blocada por colunas e o update da forma fatorada na inserção/remoção de linhas nessa matriz.

Figura 3 - Forma angular blocada por colunas

Como exemplo de problema nesta forma citamos um problema genérico de programação estocástica [Lasdon1] com a forma:

$$Ax + By = b, x \geq 0, y \geq 0 \quad (I)$$

onde b é um vetor com elementos de valores aleatórios. A seguinte sequência de eventos é assumida:

1. x é escolhido,
2. b ocorre com uma certa distribuição de probabilidade e
3. y é escolhido para tornar a relação I possível. O vetor y pode ser visto como um “vetor de recurso” que usamos para fazer a relação $Ax = b, x \geq 0$ possível.

dado x , observado b e considerando o custo de uma solução y : $d'y$, o problema passa a ser determinístico:

$$\min\{d'y \mid By = b - Ax, y \geq 0\}$$

assumimos que x é escolhido de maneira a sempre existir um y que faça (I) ser verdadeira para qualquer b possível ou seja, x é escolhido do conjunto $F = \{x \mid (\exists y) By = b - Ax, y \geq 0, \forall b\}$.

O resultado depende de b e de x e portanto chamaremos $\phi(x, b)$. Já que b é um vetor de variáveis aleatórias, $\phi(x, b)$ também o é e o problema fica:

$$\min\{c'x + E_b(\min\{d'y \mid By = b - Ax, y \geq 0\})\}$$

geralmente $E_b\phi(x, b)$ é não linear, mas um caso especial em que b pertence a um conjunto finito: $\{b_1, b_2, \dots, b_n\}$ com probabilidade de ocorrência $\{p_1, p_2, \dots, p_n\}$ então x é solução se e somente se o sistema

$$\begin{bmatrix} A & B \\ A & B \\ \vdots & \ddots \\ A & B \end{bmatrix} \begin{bmatrix} x \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}, x \geq 0, y_i \geq 0$$

tiver solução. E os dois estágios do problema podem ser resolvidos simultaneamente, minimizando a função:

$$z = c'x + \sum_{i=1}^n p_i d' y_i$$

5.1. Fatoração QR

Podemos fatorar cada sub-sistema de um problema angular bloqueado independentemente:

1. triangularizando o bloco diagonal do sub-sistema, e aplicando as mesmas rotações sobre o bloco de acoplamento, ou seja, aplicando rotações de Givens tais que $D_i = Q \begin{bmatrix} R_i \\ 0 \end{bmatrix}$ e $C_i = Q \begin{bmatrix} W_i \\ Z_i \end{bmatrix}$;
2. permutando (rodando) os blocos Z_i para o final da matriz e
3. triangularizando a matriz Z , resultado da concatenação dos blocos Z_i .

como ilustrado na Figura 4.

Figura 4 - Fatoração QR da forma angular bloqueada

5.2. Removendo Linhas

Mostraremos agora como atualizar a forma fatorada, quando removemos uma linha da base. Informalmente podemos dizer que dado $A=QR$ temos que desfazer as rotações em R de forma a reconstituir a linha que queremos remover, e então removê-la.

Vamos remover uma linha $a = [0 \quad d \quad 0 \quad c] \in A = QR$, linha do i -ésimo sub-sistema de A e fazer a atualização da forma fatorada, obtendo $\tilde{A} = \tilde{Q}\tilde{R}$. Para simplificar, vamos supor a a primeira linha do i -ésimo bloco, ou k -ésima linha, de A . Queremos reconstituir a linha a em R , ou seja, chegar à situação em que:

$$A = \begin{bmatrix} A'' \\ a \\ A' \end{bmatrix} = \begin{bmatrix} {}^1Q & {}^2Q \\ & 1 \\ {}^1Q & {}^2Q \end{bmatrix} \begin{bmatrix} R'' \\ a \\ R' \end{bmatrix} \text{ onde } {}^1Q = Q_{1:k-1,1:k-1}, {}^2Q = Q_{1:k-1,k+1:n}, {}^1Q = Q_{k+1:n,1:k-1}, {}^2Q = Q_{k+1:n,k+1:n}.$$

e

$$A = \begin{bmatrix} A'' \\ a \\ A' \end{bmatrix} \text{ onde } A'' = A_{1:k-1,\bullet} \text{ e } A' = A_{k+1:n,\bullet},$$

$$R = \begin{bmatrix} R'' \\ a \\ R' \end{bmatrix} \text{ onde } R'' = R_{1:k-1,\bullet} \text{ e } R' = R_{k+1:n,\bullet} \text{ e}$$

$$Q = \begin{bmatrix} {}^1Q \\ q \\ {}^2Q \end{bmatrix} \text{ onde } {}^1Q = Q_{1:k-1,\bullet}, {}^2Q = Q_{k+1:n,\bullet}$$

Então podemos remover a linha e obter $\tilde{A} = \tilde{Q}\tilde{R}$

Vamos calcular as rotações que precisamos fazer para reconstituir a . Considere $A = QR = Q\hat{Q}'\hat{Q}R$ onde $\hat{Q}$

$$\text{são rotações de Givens tais que } Q\hat{Q}' = \begin{bmatrix} {}^1Q & {}^2Q \\ & 1 \\ {}^1Q & {}^2Q \end{bmatrix}.$$

Como $\hat{Q}$ são rotações nas colunas de Q , precisamos computar a linha q de Q que corresponde a a em A :

$$R'q' = a'$$

note que q tem a mesma estrutura de a , como podemos ver na Figura 5. Precisamos apenas dos blocos do i -ésimo sub-sistema e do bloco Z para calcularmos q .

Figura 5 - Estrutura da linha de Q

determinamos então $\hat{Q}$ tal que $\hat{Q}q^t = I_{\bullet,k}$ e fazemos as rotações nas linhas de R .

O algoritmo é:

1. Computar q tq. $R^t q^t = a^t$
2. Aplicar rotações $\hat{Q}$ sobre q e R de forma a obter $\hat{Q}q^t = I_{\bullet,k}$. Após esta operação a primeira linha do i -ésimo bloco de R será a .
3. Remover a

Na Figura 6 mostramos as rotações feitas em R para reconstituir a linha a . Para minimizar os preenchimentos e obter a como primeira linha do i -ésimo sub-sistema, precisamos fazer as rotações a partir da última linha dos sub-sistemas envolvidos. O primeiro quadro mostra as rotações e preenchimentos no bloco Z (setas 1, 2, 3). Na rotação 3 ocorre o preenchimento de um elemento fora de Z . A primeira linha de Z é permutada para o i -ésimo sub-sistema (segundo quadro) e fazemos as rotações no i -ésimo sub-sistema, reconstituindo a (terceiro quadro), que é removida (quarto quadro).

Figura 6 - Remoção de linha de A

5.3. Inserindo Linhas

Aqui vamos tratar o problema de dada uma fatoração $A=QR$, A , R angulares bloqueadas, inserir uma linha $a = [0 \ d \ 0 \ c]$ como a primeira linha do i -ésimo bloco, ou k -ésima linha, de A e atualizar R aproveitando a fatoração prévia, obtendo $\tilde{A} = \tilde{Q}\tilde{R}$. Usaremos a seguinte notação:

$$A = \begin{bmatrix} A'' \\ A' \end{bmatrix} \text{ onde } A'' = A_{1:k-1,\bullet} \text{ e } A' = A_{k:n,\bullet},$$

$$R = \begin{bmatrix} R'' \\ R' \end{bmatrix} \text{ onde } R'' = R_{1:k-1,\bullet} \text{ e } R' = R_{k:n,\bullet} \text{ e}$$

$$Q = \begin{bmatrix} {}^1_1 Q & {}^2_1 Q \\ {}^1_2 Q & {}^2_2 Q \end{bmatrix} \text{ onde } {}^1_1 Q = Q_{1:k-1,1:k-1}, {}^2_1 Q = Q_{1:k-1,k:n}, {}^1_2 Q = Q_{k:n,1:k-1}, {}^2_2 Q = Q_{k:n,k:n}.$$

Inserindo a linha a como a primeira linha do i -ésimo sub-sistema obtemos:

$$\tilde{A} = \begin{bmatrix} A'' \\ a \\ A' \end{bmatrix} = \begin{bmatrix} {}^1_1 Q & {}^2_1 Q \\ & 1 \\ {}^1_2 Q & {}^2_2 Q \end{bmatrix} \begin{bmatrix} R'' \\ a \\ R' \end{bmatrix}$$

e neste caso, o i -ésimo bloco de R é Hessenberg superior e precisamos fazer apenas algumas rotações para ter $\tilde{R}$, triangular superior.

Os passos do algoritmo são os seguintes (Figura 7):

1. Inserir a linha no sub-sistema correspondente (quadro 2);
2. Triangularizar o bloco diagonal (quadro 3);
3. Permutar a última linha do sub-sistema como a primeira linha do bloco Z (quadro 4) e
4. Triangularizar o bloco Z (quadro 5).

Figura 7 - Inserção de linha

5.4. Reinversão

O algoritmo de atualização da fatoração da base apresentado tem uma grande eficiência resultante do aproveitamento da fatoração anterior preservando nos fatores a estrutura NRBAF da base. Conseqüentemente temos grande parcimônia no uso de operações de ponto flutuante, FLOPs. Uma consequência indireta da diminuição do número de FLOPs é a diminuição dos inevitáveis erros de arredondamento acumulados, que paulatinamente degradam a qualidade da fatoração da base. No entanto, mesmo considerando as boas qualidades de estabilidade numérica da fatoração QR, a acumulação de erros de arredondamento no obriga a, periodicamente, interromper o processo de atualização da base, recalculando a fatoração. Este periódico recálculo completo da fatoração QR da base corrente é chamado “reinversão”.

Para controlar a necessidade de reinversão, monitoramos o erro no processo de remoção de linha da base. Definimos este erro como sendo a norma da diferença entre a reconstituição da linha que estamos removendo obtida durante o processo de atualização, e a linha original. A constante de tolerância para este erro, que uma vez atingida dispara o processo de reinversão, é ajustada empiricamente.

5.5. Simplex QR

Vamos detalhar um pouco o método simplex para matrizes RBAF implementado nesta fase do projeto. A formulação do problema é a mesma, mas é importante estudar o cálculo das variáveis do método.

Como trabalhamos com inserção/remoção de linhas e o método simplex faz troca de colunas, temos que trabalhar com as transpostas. O cálculo da inversa da base (que nunca é realmente calculada) fica:

$$B^t = QU \Rightarrow Q = B^t U^{-1} \quad (I)$$

$$B^t = QU \Rightarrow B^{-1} = QU^{-t} \quad (II)$$

$$\text{de I e II: } B^{-1} = B^t U^{-1} U^{-t} \text{ e } B^{-t} = U^{-1} U^{-t} B$$

Cálculo do custo reduzido na base:

$$z = c^r - c^b \tilde{R}$$

como o resultado é um vetor e estamos interessados em um elemento dele, podemos trabalhar com sua transposta. Usaremos as mesmas letras para os vetores para não carregar demais a notação.

$$\begin{aligned} z &= c^r - \tilde{R}^t c^b = \\ &= c^r - R^t U^{-1} U^{-t} B c^b \end{aligned}$$

avaliando o segundo membro a partir do vetor, minimizamos o número de operações necessárias. Analogamente:

$$\begin{aligned} \tilde{d} &= B^{-1} d = B^t U^{-1} U^{-t} d \\ \tilde{R}_{\bullet,j} &= B^{-1} R_{\bullet,j} = B^t U^{-1} U^{-t} R_{\bullet,j} \end{aligned}$$

Com este nível de detalhe podemos partir para a implementação em alguma linguagem de programação.

5.6. Implementação MATLAB do Solver para Forma Angular Blocada

Foram feitas duas implementações MATLAB deste *solver*. A primeira, na versão MATLAB-4 precisava operar sobre nomes de variáveis para a representação dos blocos das matrizes aninhadas (por exemplo o primeiro bloco diagonal tinha o nome BD1, o segundo BD2 e assim por diante e os blocos de acoplamento tinham nome B*C_i*). Isto gerava código difícil de entender e manter. Mesmo assim esse protótipo serviu de base para o desenvolvimento do código C já usando o objeto matriz de matrizes.

Usando *cell arrays*, uma nova estrutura de dados disponível no MATLAB-5 para matriz objetos, reparamos as deficiências existentes na versão MATLAB-4.

Cell array é uma estrutura de dados que pode ser vista como uma matriz de objetos. No nosso caso, usamos um *cell array* de duas colunas e tantas linhas quanto blocos em nossa matriz blocada (fig.8)

Figura 8 - Representação CBAF

o *cell array* fica:

$$CBAF = \begin{bmatrix} [D_1] & [C_1] \\ [D_2] & [C_2] \\ \vdots & \vdots \\ [D_{n-1}] & [C_{n-1}] \\ [D_n] & [C_n] \end{bmatrix}$$

Na versão MATLAB-5, modificamos o vetor de índices da base (que eram dois vetores distintos) e agora usamos um vetor de duas colunas, a primeira corresponde ao índice do sub-sistema e a segunda ao índice da linha dentro do sub-sistema. Isto facilita a leitura do código.

6. Forma Angular Blocada Aninhada

Apresentaremos agora procedimentos de fatoração e atualização para a estrutura RBAF aninhada. NRBAF. Estes procedimentos são extensões de seus correspondentes para a forma não aninhada, e das considerações apresentadas anteriormente.

6.1. Fatoração QR

Na figura 9 ilustramos a fatoração dentro de um bloco interno, destacando as linhas do final dos blocos de acoplamento interno, que na figura 10 são permutados para o final do bloco externo (setas). Na figura 11 os todos os blocos internos já estão fatorados e destacamos as linhas finais dos blocos de acoplamento externo, que são permutadas para o final da matriz angular blocada e o bloco resultante é triangularizado, como ilustrado na figura 12.

Figura 9 - Fatoração dos blocos internos

Figura 10 – Fatoração do bloco diagonal externo.

Figura 11 - Fatoração de todos os blocos diagonais externos.

Figura 12 – Fim do processo de fatoração da matriz RBAF

6.2. Inserindo Linhas

A inserção de uma linha na forma fatorada transforma o bloco diagonal interno em *Hessenberg* superior (fig. 13). Para ter a matriz toda triangular superior temos que fazer rotações envolvendo esse bloco (consequentemente nos blocos de acoplamento correspondentes), o último bloco de acoplamento interno desse bloco diagonal externo (que é sub-produto da fatoração QR do bloco externo), como feito para matrizes BAF (fig. 14) e o último bloco de acoplamento externo (sub-produto da fatoração da matriz) (fig.15)

Figura 13 – Inserção de linha.

Figura 14 – Triangularização da forma fatorada

Figura 15 – Triangularização da forma fatorada

6.3. *Removendo Linhas*

Remover uma linha da base na forma fatorada implica em “desfazer” todas as rotações que envolveram essa linha, reconstituindo a linha e então removê-la. Sabemos, por construção, que tais rotações ocorreram entre linhas do bloco interno a que a linha pertence, linhas do último bloco de acoplamento interno do bloco externo a que a linha pertence e linhas do último bloco de acoplamento externo. A determinação do ângulo de rotação é dada na seção 3. Na figura 16 mostramos as rotações sobre o último bloco de acoplamento externo (1) e a permutação da primeira linha deste bloco para o i -ésimo bloco (2). Na figura 17 as rotações dentro do último bloco interno do i -ésimo bloco externo (1) a permutação da linha para o bloco interno de onde a linha tem que ser removida (2) e as rotações no bloco interno (3). Na figura 18, a matriz com a linha que queremos remover reconstituída. Note que a matriz continua triangular superior.

Figura 16 - Remoção de linha

Figura 17 – Remoção de linha

Figura 18 – Remoção de linha

6.4. Implementação MATLAB

Baseados na estrutura adotada para a forma angular blocada, vamos criar uma representação no MATLAB para a forma aninhada que possibilite a reutilização do código já existente e que proporcione mínimo *overhead*. Basicamente estudamos duas representações:

6.4.1. Reentrante

Uma reprodução da representação interna, onde temos blocos diagonais externos constituídos por matrizes na forma angular blocada e um bloco de conexão externo, denso, correspondente (fig.19).

Figura 19 - Representação Reentrante

Cuja representação como *cell array* é:

$$NRBAF = \begin{bmatrix} \begin{bmatrix} [Di_1] & [Ci_1] \\ [Di_2] & [Ci_2] \\ \vdots & \vdots \\ [Di_{n-1}] & [Ci_{n-1}] \\ [Di_n] & [Ci_n] \end{bmatrix}_1 & [Ce_1] \\ \begin{bmatrix} [Di_1] & [Ci_1] \\ [Di_2] & [Ci_2] \\ \vdots & \vdots \\ [Di_{n-1}] & [Ci_{n-1}] \\ [Di_n] & [Ci_n] \end{bmatrix}_2 & [Ce_2] \\ \vdots & \vdots \\ \begin{bmatrix} [Di_1] & [Ci_1] \\ [Di_2] & [Ci_2] \\ \vdots & \vdots \\ [Di_{n-1}] & [Ci_{n-1}] \\ [Di_n] & [Ci_n] \end{bmatrix}_{m-1} & [Ce_{m-1}] \\ \begin{bmatrix} [Di_1] & [Ci_1] \\ [Di_2] & [Ci_2] \\ \vdots & \vdots \\ [Di_{n-1}] & [Ci_{n-1}] \\ [Di_n] & [Ci_n] \end{bmatrix}_m & [Ce_m] \end{bmatrix}$$

Esta representação tem a vantagem de se aproveitar todo o código da etapa anterior, já que os blocos diagonais externos estão na forma angular blocada. No entanto para proceder as rotações no bloco de acoplamento externo precisamos dos ângulos usados na fatoração dos blocos internos. Além disso precisaremos particionar os blocos de acoplamento externo na granularidade dos blocos internos, antevemos portanto alguns problemas de implementação e desempenho:

Uma variação interessante dessa representação é particionar os blocos de acoplamento externo na granularidade dos blocos internos. Isto nos guia à segunda forma, que é a que adotamos.

6.4.2. Angular blocada estendida

Consiste em estender a representação interna acrescentando uma matriz densa esta é a fração do bloco de acoplamento externo associada ao bloco interno. Além disso precisamos de um índice de bloco externo.

Figura 20 - Representação angular blocada estendida

cuja notação em um *cell array* é:

$$NRBAF = \left[\begin{array}{c} \left[\begin{array}{ccc} [Di_1] & [Ci_1] & [Ce_1] \\ [Di_2] & [Ci_1] & [Ce_1] \\ \vdots & \vdots & \vdots \\ [Di_{n-1}] & [Ci_{n-1}] & [Ce_{n-1}] \\ [Di_n] & [Ci_n] & [Ce_n] \end{array} \right]_1 \\ \left[\begin{array}{ccc} [Di_1] & [Ci_1] & [Ce_1] \\ [Di_2] & [Ci_1] & [Ce_1] \\ \vdots & \vdots & \vdots \\ [Di_{n-1}] & [Ci_{n-1}] & [Ce_{n-1}] \\ [Di_n] & [Ci_n] & [Ce_n] \end{array} \right]_2 \\ \vdots \\ \left[\begin{array}{ccc} [Di_1] & [Ci_1] & [Ce_1] \\ [Di_2] & [Ci_1] & [Ce_1] \\ \vdots & \vdots & \vdots \\ [Di_{n-1}] & [Ci_{n-1}] & [Ce_{n-1}] \\ [Di_n] & [Ci_n] & [Ce_n] \end{array} \right]_{m-1} \\ \left[\begin{array}{ccc} [Di_1] & [Ci_1] & [Ce_1] \\ [Di_2] & [Ci_1] & [Ce_1] \\ \vdots & \vdots & \vdots \\ [Di_{n-1}] & [Ci_{n-1}] & [Ce_{n-1}] \\ [Di_n] & [Ci_n] & [Ce_n] \end{array} \right]_m \end{array} \right]$$

Usando esta representação existe a necessidade de estender o código das funções para tratar os blocos densos de acoplamento externo, que são muito parecidos com os blocos de acoplamento interno, mas eliminamos as passagens de ângulos e a necessidade de reparticionar os blocos de acoplamento externo. Na nossa avaliação esta é a forma mais eficiente de tratar o problema.

Os vetores de índices passam a ter três colunas: a primeira é o índice do bloco externo, a segunda do bloco interno e a terceira a da linha.

Cuidados especiais tem que ser tomados com relação aos blocos densos de dimensão nula pois em função deles pode haver, durante a fatoração, migração de um setor para o fim da matriz. Ocorre que no *update* da base, uma linha entra ou sai desse setor em sua posição original e não o final da matriz, logo, a eliminação desse setor vazio na forma fatorada não é possível. Conseqüentemente, todas as funções tem que saber lidar com setores vazios.

Outro aspecto importante a considerar é a possibilidade de um setor se esvaziar na base ou no resíduo e de como tratar as dimensões dos blocos densos nesse caso já que isso influi nas operações com vetores. A solução é manter blocos densos com zero linhas e n colunas.

Problemas onde em todos os blocos diagonais externos não há bloco de acoplamento interno também são resolvidos, ou seja, problemas RBAF podem ser resolvidos por este *solver* NRBAF. O desempenho do *solver* NRBAF e do *solver* específico para problemas RBAF deve diferir por muito pouco, correspondendo apenas a overheads de chamadas de função.

Para facilitar a depuração, são necessárias rotinas de conversão da estrutura NRBAF para uma matriz densa e outras para visualização dos exemplos.

Em alguns casos o tratamento de exceções no MATLAB se mostraram inadequados. Por exemplo quando se faz uma seleção sobre uma matriz vazia seria de se esperar que a operação retornasse uma matriz vazia. O que ocorre de fato é a finalização do *script* com uma mensagem de erro. Para contornar isso são necessários testes para não operar sobre matrizes vazias. Algo que não esperamos no código C.

7. Codificação em Linguagem C

O objetivo nesta etapa é implementar o algoritmo simplex aproveitando a estrutura NRBAF da família de problemas estudados e atualizando as formas fatoradas da base de maneira a minimizar a quantidade de cálculos necessários.

Apresentamos em anexos as listagens dos programas e dos dados obtidos em testes. Segue a teoria envolvida e descrição do código MATLAB.

O desenvolvimento de código em linguagem de nível mais alto – MATLAB – como etapa intermediária do projeto, onde pretendemos chegar ao código em C, se mostra uma estratégia acertada à medida que:

- levanta questões de implementação e facilita seus testes;
- permite avaliação de resultados que possibilitam:
 - correções de rumo a um custo menor que o da correção sobre código C e
 - prévias do desempenho do produto final.
- gera um ambiente para desenvolvimento rápido de melhoria sobre o *solver*;
- isola funções de suporte, como alocação de memória e verificação de dimensões de matrizes, de funções de cálculo, facilitando (ou até induzindo) encapsulamento de funções e
- facilita a detecção de erros de codificação e a depuração do código em C.

Baseados no protótipo em linguagem MATLAB, definimos quais as novas classes (estruturas de dados e métodos) e as adaptações necessárias para a implementação do *solver* para problemas na forma angular blocada aninhada em linguagem C.

Tivemos cuidado especial com a alocação e acesso a memória e integridade do *heap* pois erros simples nesses pontos são difíceis de detectar e provocam desde falhas de proteção de memória até a apresentação de resultados errados. Para isso em todas as funções de acesso verificamos a validade dos índices usados, sempre que alocamos ou liberamos memória um aviso é emitido e existem funções para percorrer o *heap* e checar sua integridade. A verificação, apesar de muito útil, consome tempo, logo optamos por ativá-la por diretivas de compilação, permitindo gerar a aplicação sem este overhead.

Abaixo apresentamos todas as classes e bibliotecas e a extensão das modificações

xrbaf , nrbafe, tripla, vveto	novas classes
iveto	grandes modificações
simplex, QR	pequenas modificações
veto, densa, linear	sem modificações

7.1. densa

Esta é a representação para matrizes densas escolhida para este projeto e correspondem aos blocos densos das formas RBAF e NRBAF. Sua maior vantagem é permitir permutações de linhas (ou rotação por π) dentro de um bloco ou entre blocos em tempo constante. Além disso o acesso a linhas e elementos é muito eficiente. Apenas as operações de inserção e remoção necessitam deslocar os elementos do vetor de cabeças de linhas, mas são pouco usadas se comparadas com as operações de acesso.

Às vezes precisamos redimensionar uma matriz e se sempre que isso ocorrer, precisarmos realocar memória, vamos penalizar muito o desempenho do algoritmo. Por outro lado, é muito custoso (às vezes impossível) computar o tamanho máximo de um problema e alocar toda a memória necessária – se houver memória suficiente para isso. Isto mostra a necessidade de uma boa política de alocação de memória. Nesta e em outras estruturas, os *arrays* têm armazenados o seu comprimento corrente (m , n) e o seu comprimento máximo (m_{max} , n_{max}) – como ilustrado na figura abaixo. Cada vez que os vetores são redimensionados, os comprimentos máximos são estabelecidos com alguma “folga”. Esta folga é controlada por uma heurística que faz o compromisso entre os objetivos conflitantes de uso eficiente de memória (folgas pequenas) e baixa frequência de realocação (folgas grandes).

Figura 21 - Diagrama da representação densa por linhas

```
class densa {
    int m, n,                /* Tamanho corrente. */
        mmax, nmax;        /* Tamanho maximo -
                               que corresponde ao tamanho alocado. */
    double **M;
public:
    ...
}
```

7.2. linear

Esta é uma biblioteca de operações de álgebra linear entre matrizes densas e vetores: produto, produto pela inversa para matrizes triangulares e algumas outras. Nenhuma classe é definida aqui.

7.3. QR

Fatoração QR e rotinas de inserção e remoção de linha e fatoração de matrizes na forma Hessenberg superior. Envolve as representações de matrizes densas e vetores. Cada função opera até três blocos densos simultaneamente pois cada linha na matriz NUFFA tem até três segmentos não nulos.

As funções definidas em linear e QR, sempre que possível, usam os elementos das matrizes linha a linha e em seqüência, logo o acesso a um elemento é feito apenas incrementando ponteiros tornando a “decodificação” dos índices da matriz muito mais eficiente.

7.4. *xrbaf*

Define um objeto RBAF estendido para armazenar os blocos de acoplamento externo. Constituindo um bloco externo NRBAF. Na figura abaixo comparamos as estruturas RBAF e XRBAF, destacando os vetores de ponteiros para blocos densos e os blocos densos dispostos como pede o problema proposto. Nenhuma operação aritmética é definida nesta classe.

Figura 22 - Diagrama da representação XRBAF

```

class xrbaf {
    densa **D,      /* diagonal */
    **C,           /* acoplamento interno */
    **E;           /* acoplamento externo */
    int n,          /* numero de blocos em cada vetor */
        nmax;      /* maximo numero de blocos possivel sem
realocacao. */
public:

```

Código 2 - Estrutura de dados da representação densa por linhas

7.5. *nrbafe*

Esta classe é um *array* de objetos XRBAF. Nela são definidas todas as operações aritméticas necessárias para a implementação do *solver*, por exemplo funções de produto por vetores, solução de sistemas lineares e os *updates* da forma fatorada. A interface desta classe é a mesma para matrizes densas, RBAF ou NRBAF, permitindo reutilizar a maior parte do código escrito para o SIMPLEX para problemas na forma RBAF.

```

class nrbafe {
    xrbaf **X;
    int n,          /* numero de blocos em cada vetor */
        nmax;      /* maximo numero de blocos possivel sem
realocacao. */
public:

```

Código 3 - Estrutura de dados da representação densa por linhas

A representação de uma matriz NRBAF detalhando as estruturas XRBAF e o *array* da estrutura NRBAF é apresentada na figura abaixo. Os objetos da classe densa estão hachurados. Os *arrays* da classe XRBAF (xrbaf.D, xrbaf.C, xrbaf.E) estão agrupados em retângulos tracejados que representam objetos XRBAF. Cada elemento de cada vetor aponta para um objeto da classe densa. Finalmente o *array* X da estrutura NRBAF aponta para as estruturas XRBAF, formando uma matriz NRBAF.

Figura 23 - Diagrama da representação NBAF

7.6. vetor

Figura 24 - Diagrama da representação Vetor

Nesta classe são implementados métodos para criação e destruição de instâncias da classe, algumas operações aritméticas sobre vetores, como soma, produto, máximo e mínimo.

```
class vetor {  
    int n,      /* Tamanho corrente. */  
        nmax; /* Tamanho maximo - que corresponde ao tamanho  
alocado. */  
    double *V;  
public:  
    ...
```

Código 4 Estrutura de dados da representação Vetor

Muitas operações entre matrizes blocadas e vetores podem usar blocos densos e segmentos de vetores de forma muito eficiente. Na figura abaixo, ilustramos o produto entre matriz blocada (quadrada) e vetor. Os blocos densos da matriz e os segmentos de vetor marcados com um asterisco são aqueles que definem o valor do segmento resultante (hachurado). A forma mais segura e elegante de usar esses segmentos é copiá-los em novos vetores, obter os segmentos do resultado e concatená-los – esta solução (1) foi a usada no protótipo escrito em MATLAB. Mas a forma mais eficiente (quando temos apenas um processador) é criar ponteiros (qd.V, qc.V e qe.V) para segmentos de q.V, preencher o resto das estruturas de qd, qc e qe e executar a operação – esta solução (2) foi a usada no protótipo em C. Para isso usamos funções de acesso direto à estrutura da classe vetor.

Figura 25 - Produto entre matriz NRBAF e vetor

Cabe observar que em uma máquina paralela a solução 1 pode ser melhor.

7.7. *ivector*

Esta classe implementa um vetor de índices, usado para manter as listas de variáveis básicas e não básicas. Armazenando de maneira inteligente ela converte índices sequenciais para triplas (bloco externo, bloco interno, índice no bloco interno) e recebe atualizações eficientemente.

```
typedef struct {
    int be,          /* bloco externo */
        bi,          /* bloco interno */
        i;           /* índice */
} tripla;

class ivector {
    int n,            /* comprimento corrente */
        nmax;         /* comprimento alocado */
    vvotor tag,       /* ponteiro para o primeiro elemento do bloco */
        xtag;
    tripla *t;        /* ponteiro para tripla */
public:
```

Código 5 Estrutura da representação ivector

Por exemplo, no simplex o índice da coluna que sai da base é o índice, i , de um certo elemento em um vetor. i também é o índice da tripla no vetor de índices da base, logo sabemos a que blocos (externo e interno) a coluna pertence. Sua posição dentro dos blocos da base é dada pela diferença entre o endereço da i -ésima tripla e o endereço da primeira tripla do bloco interno.

As inserções ocorrem sempre no início ou no final de um bloco interno, como sabemos em que bloco externo e interno a coluna será inserida, e o vetor tag aponta para o início de todo bloco interno, sabemos a posição da inserção em tempo constante.

Figura 26 - Diagrama da representação vetor de índices

8. Testes

Foram realizados testes comparativos entre o protótipo MATLAB do algoritmo para problemas estruturados e um algoritmo simples que não se aproveita de estrutura nem de esparsidade, tratando o problema como uma matriz densa. Os problemas usados para teste têm uma parte gerada aleatoriamente, definindo-se a quantidade de blocos externos (períodos) e a esparsidade, sempre gerando blocos de acoplamento mais esparsos que os blocos diagonais, e outra parte constituída de variáveis artificiais, para termos um vértice inicial e solução ótima. As informações mais importantes são a aceleração – razão entre a quantidade de operações de ponto-flutuante efetuadas pelos algoritmos e o erro entre as soluções encontradas. O erro médio para os exemplos testados é da ordem de 10^{-12} e o número médio de reinversões é treze.

O *solver* para problemas NRBAF escrito em C foi, para os exemplos testados, cerca de 40 vezes mais rápido que o *solver* MATLAB.

Abaixo o padrão de esparsidade de um problema gerado.

Figura 27 – Padrão de esparsidade de um problema testado

Apresentamos a seguir gráficos de dispersão dos problemas de teste. O gráfico 1 mostra, para cada teste, o número de operações de ponto flutuante para o algoritmo simples. O gráfico 2 mostra o número de operações para o algoritmo estruturado e o gráfico 3, a aceleração obtida.

Gráfico 1 – Dispersão dos testes - Solver Denso

Gráfico 2 - Dispersão dos testes - Solver Estruturado

Gráfico 3 - Dispersão dos testes - Aceleração

Os próximos gráficos apresentam quantidades médias em função do número de variáveis básicas. O gráfico 4 mostra o número médio de operações de ponto flutuante para o algoritmo simples. O gráfico 5 mostra o número médio de operações para o algoritmo estruturado e o gráfico 6, a aceleração média obtida. Usamos uma ferramenta de bancos de dados SQL para o tratamento das amostras pois precisamos agregar dados de problemas diferentes de mesmo tamanho e funções agregadas SQL o fazem bem.

Gráfico 4 - Médias dos testes - Solver Denso

Gráfico 5 - Médias dos testes - Solver Estruturado

Gráfico 6 - Médias dos testes - Aceleração

Apresentamos a seguir a listagem completa dos resultados dos testes efetuados. Cada linha da matriz de resultados dos testes corresponde a um teste e cada coluna contém a seguinte informação:

1. número de blocos externos (períodos)
2. número de restrições de acoplamento externo
3. esparsidade esperada dos blocos de acoplamento externo
4. razão esperada entre variáveis do problema e variáveis básicas
5. número de variáveis básicas
6. número de operações efetuadas pelo algoritmo estruturado
7. número de operações efetuadas pelo algoritmo simples
8. norma da diferença entre as soluções ótimas encontradas
9. quantidade de variáveis artificiais remanescente na base ótima
10. número de mudanças de base efetuadas até chegar a solução ótima
11. número de reinversões necessárias até chegar a solução ótima.

Listagem dos resultados dos testes.

1	2	3	4	5	6	7	8	9	10	11
6	10	0.9	2	74	3.550E+06	7.243E+07	2.58549E-11	28	104	4
6	10	0.9	2	79	5.388E+06	1.346E+08	1.72763E-13	30	141	10
6	10	0.9	2	83	4.600E+06	9.479E+07	3.13295E-13	35	107	8
6	10	0.9	2	67	3.925E+06	6.954E+07	1.64805E-11	21	133	3
6	10	0.9	2	71	4.991E+06	1.038E+08	1.31761E-10	23	148	7
6	10	0.9	2	106	6.728E+06	2.313E+08	1.88656E-15	50	126	8
6	10	0.9	2	82	7.169E+06	1.590E+08	4.27543E-15	33	162	20
6	10	0.9	2	114	1.448E+07	6.515E+08	2.55462E-14	45	252	12
6	10	0.9	2	82	5.636E+06	1.605E+08	5.33161E-15	36	138	8
6	10	0.9	2	81	5.700E+06	1.036E+08	6.473E-15	35	122	10
6	10	0.9	2	60	3.046E+06	5.051E+07	2.26969E-14	12	113	5
6	10	0.9	2	73	3.839E+06	6.711E+07	3.24149E-15	31	100	12
6	10	0.9	2	83	7.193E+06	2.030E+08	1.06253E-10	26	180	9
6	10	0.9	2	125	1.383E+07	6.976E+08	3.49576E-13	53	214	10
6	10	0.9	2	103	1.040E+07	4.215E+08	5.47224E-15	43	195	9
6	10	0.9	2	108	1.149E+07	4.887E+08	1.41833E-13	44	217	11
6	10	0.9	2	90	5.589E+06	1.296E+08	1.84129E-14	43	127	7
6	10	0.9	2	131	2.550E+07	1.520E+09	5.55498E-15	56	350	24
6	10	0.9	2	89	4.657E+06	1.039E+08	2.47115E-14	42	112	5
6	10	0.9	2	71	2.949E+06	4.514E+07	4.01821E-15	27	82	5
6	10	0.9	2	120	1.571E+07	7.638E+08	4.32038E-14	45	250	23
6	10	0.9	2	73	7.258E+06	1.525E+08	2.96557E-14	28	187	12
6	10	0.9	2	84	6.177E+06	1.502E+08	6.47331E-14	36	142	13
6	10	0.9	2	81	3.914E+06	7.556E+07	2.82698E-15	32	111	6
6	10	0.9	2	119	1.465E+07	7.386E+08	1.94948E-14	51	235	16
6	10	0.9	2	136	1.677E+07	9.426E+08	1.0878E-13	67	225	16
6	10	0.9	2	111	7.812E+06	2.033E+08	3.58679E-14	52	131	10
6	10	0.9	2	99	1.134E+07	3.596E+08	2.74457E-14	38	227	17
6	10	0.9	2	100	1.216E+07	4.655E+08	2.17076E-13	32	237	18
6	10	0.9	2	76	3.804E+06	8.750E+07	6.32289E-15	35	106	2
6	10	0.9	2	71	4.212E+06	7.417E+07	5.15051E-15	28	121	7
6	10	0.9	2	75	4.395E+06	7.908E+07	3.40359E-13	34	125	11
6	10	0.9	2	73	6.697E+06	1.232E+08	2.93424E-15	26	167	16
6	10	0.9	2	71	6.761E+06	1.340E+08	6.38435E-14	14	184	12
6	10	0.9	2	98	5.483E+06	2.046E+08	4.01151E-13	45	112	4
6	10	0.9	2	127	1.480E+07	7.667E+08	1.64272E-14	62	223	21
6	10	0.9	2	88	6.395E+06	1.522E+08	3.43643E-15	44	130	15
6	10	0.9	2	121	1.825E+07	8.945E+08	1.29025E-13	49	277	21
6	10	0.9	2	97	8.049E+06	2.561E+08	1.98496E-15	49	152	8
6	10	0.9	2	107	6.485E+06	2.036E+08	2.82046E-15	51	118	7
6	10	0.9	2	74	4.770E+06	9.910E+07	1.78148E-12	24	137	6
6	10	0.9	2	97	9.374E+06	2.731E+08	4.42844E-15	34	190	14
6	10	0.9	2	66	2.053E+06	3.625E+07	4.90681E-15	24	69	3
6	10	0.9	2	87	5.068E+06	1.486E+08	1.11489E-14	34	125	4
6	10	0.9	2	113	1.368E+07	5.898E+08	1.09803E-13	48	244	14
6	10	0.9	2	128	1.973E+07	9.018E+08	8.76929E-15	59	272	32
6	10	0.9	2	90	5.060E+06	1.402E+08	1.57851E-14	49	112	4
8	10	0.9	2	92	8.425E+06	2.658E+08	3.60675E-15	36	202	11
8	10	0.9	2	106	1.398E+07	5.838E+08	2.03881E-13	39	260	24
8	10	0.9	2	112	1.132E+07	4.221E+08	2.91868E-15	45	188	9
8	10	0.9	2	150	2.955E+07	2.008E+09	6.04944E-15	70	359	26
8	10	0.9	2	144	2.798E+07	2.111E+09	9.32176E-14	51	409	25

1	2	3	4	5	6	7	8	9	10	11
8	10	0.9	2	133	2.377E+07	1.277E+09	4.11861E-14	55	313	30
8	10	0.9	2	131	1.862E+07	1.080E+09	4.80744E-15	55	280	22
8	10	0.9	2	137	2.362E+07	1.415E+09	4.63376E-11	64	330	21
8	10	0.9	2	117	9.380E+06	3.966E+08	9.01639E-13	57	152	12
8	10	0.9	2	131	1.673E+07	8.770E+08	3.01214E-14	56	239	13
8	10	0.9	2	128	1.507E+07	8.049E+08	4.86991E-14	56	236	17
8	10	0.9	2	176	4.857E+07	4.831E+09	2.54759E-14	77	493	37
8	10	0.9	2	104	1.080E+07	3.621E+08	1.05743E-13	44	194	14
8	10	0.9	2	124	1.371E+07	6.351E+08	1.85865E-14	64	208	14
12	10	0.9	2	206	3.586E+07	4.112E+09	4.99555E-15	112	333	24
12	10	0.9	2	186	3.944E+07	3.701E+09	4.344E-13	87	387	37
12	10	0.9	2	183	5.345E+07	6.286E+09	2.15795E-13	67	564	42
12	10	0.9	2	153	1.939E+07	1.024E+09	5.81284E-14	69	260	18
12	10	0.9	2	192	3.935E+07	3.950E+09	8.42898E-14	82	392	29
12	10	0.9	2	157	2.147E+07	1.558E+09	9.47254E-13	69	285	18
12	10	0.9	2	151	2.117E+07	1.496E+09	8.84944E-13	61	287	19
12	10	0.9	2	124	9.776E+06	4.212E+08	2.07376E-15	56	183	10
12	10	0.9	2	151	1.476E+07	8.073E+08	2.42391E-15	72	204	11
12	10	0.9	2	197	3.984E+07	4.429E+09	1.63051E-13	81	402	33
12	10	0.9	2	183	4.182E+07	4.205E+09	6.89766E-15	75	442	40
12	10	0.9	2	205	4.444E+07	4.611E+09	3.63998E-13	112	411	32
12	10	0.9	2	195	3.847E+07	3.939E+09	1.02005E-12	86	406	20
12	10	0.9	2	165	2.231E+07	2.020E+09	8.15259E-13	71	284	16
12	10	0.9	2	178	2.611E+07	2.117E+09	5.85811E-15	77	304	17
12	10	0.9	2	165	2.993E+07	2.389E+09	1.38453E-14	61	392	19
12	10	0.9	2	192	2.556E+07	2.557E+09	1.41501E-13	101	284	13
12	10	0.9	2	138	1.559E+07	9.345E+08	3.14049E-13	51	252	10
12	10	0.9	2	231	6.908E+07	1.041E+10	1.88261E-14	109	574	38
4	10	0.9	2	79	4.397E+06	9.079E+07	3.66171E-14	35	100	11
4	10	0.9	2	60	2.665E+06	3.381E+07	1.09375E-13	29	86	9
4	10	0.9	2	50	1.461E+06	1.432E+07	5.20182E-15	18	60	3
4	10	0.9	2	72	3.788E+06	6.688E+07	9.56585E-15	33	105	4
4	10	0.9	2	45	9.642E+05	7.336E+06	6.91967E-15	19	43	3
4	10	0.9	2	97	1.289E+07	4.399E+08	4.34059E-15	36	222	22
4	10	0.9	2	78	7.602E+06	1.691E+08	1.0087E-13	31	174	14
4	10	0.9	2	40	9.833E+05	5.914E+06	2.12921E-15	15	46	5
4	10	0.9	2	56	3.156E+06	3.837E+07	1.00035E-14	18	105	9
4	10	0.9	2	91	5.742E+06	1.534E+08	2.5191E-15	47	117	8
4	10	0.9	2	98	1.267E+07	4.702E+08	9.12199E-13	37	217	18
4	10	0.9	2	57	1.531E+06	2.016E+07	7.0625E-15	26	61	1
4	10	0.9	2	74	5.435E+06	1.058E+08	7.81288E-15	29	131	9
4	10	0.9	2	66	4.774E+06	7.350E+07	1.71051E-14	22	129	9
4	10	0.9	2	64	2.743E+06	4.031E+07	9.61342E-14	28	86	6
4	10	0.9	2	79	4.982E+06	1.125E+08	3.11133E-13	34	119	10
4	10	0.9	2	84	6.123E+06	1.339E+08	3.12532E-15	39	136	11
4	10	0.9	2	38	1.007E+06	5.541E+06	4.39912E-15	15	50	4
4	10	0.9	2	68	3.698E+06	6.896E+07	3.36872E-15	26	108	8
3	10	0.9	2	65	5.735E+06	8.707E+07	5.06909E-15	30	133	16
3	10	0.9	2	49	1.875E+06	2.000E+07	1.00206E-13	20	77	4
3	10	0.9	2	67	3.709E+06	5.344E+07	2.96051E-15	28	99	5
3	10	0.9	2	57	1.872E+06	2.076E+07	5.76667E-15	30	59	4
3	10	0.9	2	44	1.491E+06	1.092E+07	1.85293E-13	16	62	6
3	10	0.9	2	51	1.389E+06	1.116E+07	1.05557E-15	28	41	5
3	10	0.9	2	59	3.052E+06	3.301E+07	2.55952E-15	23	84	9

1	2	3	4	5	6	7	8	9	10	11
3	10	0.9	2	32	3.890E+05	1.536E+06	6.33916E-15	15	27	0
3	10	0.9	2	43	9.031E+05	5.837E+06	5.16756E-15	16	47	2
3	10	0.9	2	54	4.983E+06	5.380E+07	1.30067E-14	21	140	22
3	10	0.9	2	34	5.455E+05	2.828E+06	2.72286E-15	14	38	0
3	10	0.9	2	45	1.639E+06	1.375E+07	7.11179E-15	16	69	6
3	10	0.9	2	58	2.723E+06	3.467E+07	7.09866E-15	23	92	10
3	10	0.9	2	49	2.494E+06	2.483E+07	1.32148E-14	19	95	5
3	10	0.9	2	45	1.390E+06	1.284E+07	1.1334E-14	16	71	2
3	10	0.9	2	48	1.627E+06	1.685E+07	1.11142E-15	19	72	1
3	10	0.9	2	56	2.599E+06	2.490E+07	9.71026E-14	24	74	10
3	10	0.9	2	54	1.734E+06	1.851E+07	6.72181E-15	17	66	2
3	10	0.9	2	58	3.122E+06	3.696E+07	9.90216E-15	23	99	10
3	10	0.9	2	56	1.869E+06	2.385E+07	1.73442E-14	25	69	2
3	10	0.9	2	47	2.455E+06	3.680E+07	1.97966E-13	11	124	4
12	10	0.9	2	149	1.624E+07	1.042E+09	2.24836E-14	40	277	9
12	7	0.91	3	193	3.346E+07	4.872E+09	1.55612E-14	42	445	13
12	7	0.91	3	217	3.580E+07	5.723E+09	6.25316E-15	53	419	13
12	7	0.91	3	171	2.355E+07	2.487E+09	4.41718E-14	35	362	9
12	7	0.91	3	175	2.095E+07	2.297E+09	2.59183E-13	54	306	11
12	7	0.91	3	149	1.496E+07	1.110E+09	2.2962E-14	41	269	7
12	7	0.91	3	106	8.475E+06	5.306E+08	1.45051E-13	20	211	8
18	7	0.91	3	288	3.662E+07	1.053E+10	7.406E-15	178	324	7
18	7	0.91	3	261	4.722E+07	1.365E+10	3.6937E-15	156	438	20
18	7	0.91	3	276	3.502E+07	8.387E+09	4.42003E-15	187	315	11
18	7	0.91	3	217	2.444E+07	4.350E+09	3.65502E-15	136	282	13
18	7	0.91	3	271	3.213E+07	8.104E+09	6.70321E-14	177	297	8
18	7	0.91	3	242	3.183E+07	7.593E+09	1.63371E-14	142	323	11
18	7	0.91	3	263	4.633E+07	1.187E+10	5.37825E-15	157	439	18
18	7	0.91	3	257	4.891E+07	1.182E+10	1.82525E-13	158	461	22
18	7	0.91	3	282	4.209E+07	1.071E+10	6.36046E-14	167	368	12
18	7	0.91	3	316	4.237E+07	1.038E+10	3.15131E-15	223	334	13
18	7	0.91	3	277	2.937E+07	6.639E+09	6.04381E-15	191	268	7
18	7	0.91	3	226	2.270E+07	5.020E+09	1.15766E-14	146	258	8
18	7	0.91	3	269	4.561E+07	1.404E+10	7.33204E-14	169	431	18
18	7	0.91	3	324	4.197E+07	1.277E+10	3.88748E-15	212	324	11
18	7	0.91	3	258	3.522E+07	8.196E+09	5.05455E-14	159	345	12
18	7	0.91	3	269	5.343E+07	1.432E+10	1.41656E-14	147	495	17
18	7	0.91	3	245	3.051E+07	7.078E+09	1.82346E-14	157	308	11
18	7	0.91	3	226	3.020E+07	8.211E+09	9.37514E-15	124	337	16
18	7	0.91	3	246	3.414E+07	9.448E+09	2.2262E-10	140	342	15
18	7	0.91	3	257	3.922E+07	9.813E+09	2.53581E-14	162	370	12
18	7	0.91	3	266	5.072E+07	1.431E+10	6.12889E-15	149	474	23
18	7	0.91	3	233	3.156E+07	7.806E+09	1.28464E-14	141	338	14
18	7	0.91	3	221	3.767E+07	7.744E+09	5.90741E-15	113	412	21
18	7	0.91	3	276	3.274E+07	8.022E+09	7.02051E-15	175	296	18
18	7	0.91	3	228	2.603E+07	5.511E+09	2.86485E-15	142	285	16
18	7	0.91	3	247	2.638E+07	7.260E+09	1.30764E-14	174	263	10
18	7	0.91	3	266	3.166E+07	8.261E+09	3.74181E-14	161	300	12
18	7	0.91	3	250	3.683E+07	9.311E+09	4.19835E-15	160	355	15
18	7	0.91	3	180	2.230E+07	3.882E+09	1.73726E-14	98	312	12
18	7	0.91	3	221	2.710E+07	4.858E+09	2.99852E-13	126	311	14
18	7	0.91	3	308	4.021E+07	1.441E+10	3.53673E-15	191	325	13
18	7	0.91	3	251	3.986E+07	1.019E+10	8.88633E-15	151	394	18
18	7	0.91	3	225	2.491E+07	4.567E+09	8.41561E-15	149	269	11

1	2	3	4	5	6	7	8	9	10	11
18	7	0.91	3	244	4.249E+07	1.303E+10	3.66805E-15	142	423	20
18	7	0.91	3	234	2.561E+07	6.112E+09	3.68642E-15	150	270	10
18	7	0.91	3	253	2.659E+07	4.508E+09	1.24888E-13	173	264	10
18	7	0.91	3	235	3.258E+07	7.291E+09	1.37658E-14	135	343	10
18	7	0.91	3	301	7.053E+07	2.441E+10	4.10994E-15	181	584	24
18	7	0.91	3	202	1.955E+07	4.427E+09	7.47223E-15	125	233	8
18	7	0.91	3	246	3.097E+07	6.643E+09	2.5227E-14	162	311	10
18	7	0.91	3	247	3.893E+07	9.333E+09	1.46452E-14	144	394	16
24	7	0.91	3	391	5.059E+07	1.801E+10	1.7645E-14	281	331	8
24	7	0.91	3	304	4.204E+07	1.302E+10	4.19379E-14	191	348	9
24	7	0.91	3	345	7.241E+07	3.039E+10	1.60046E-14	213	520	23
24	7	0.91	3	276	3.258E+07	8.986E+09	3.7094E-14	168	305	7
24	7	0.91	3	355	6.027E+07	2.342E+10	4.11527E-15	222	423	15
24	7	0.91	3	361	5.210E+07	2.050E+10	7.95856E-15	243	361	18
24	7	0.91	3	289	3.670E+07	1.010E+10	1.04098E-14	192	317	9
24	7	0.91	3	280	5.553E+07	1.584E+10	7.33149E-15	150	492	30
24	7	0.91	3	337	3.478E+07	1.231E+10	3.80907E-15	239	257	10
24	7	0.91	3	305	3.428E+07	1.028E+10	7.85801E-15	202	282	7
24	7	0.91	3	446	8.823E+07	4.734E+10	4.15729E-14	314	481	21
24	7	0.91	3	385	4.804E+07	1.436E+10	1.30933E-14	266	317	11
24	7	0.91	3	290	3.808E+07	1.038E+10	3.5512E-14	195	329	11
24	7	0.91	3	312	5.307E+07	1.744E+10	9.10415E-15	192	422	17
24	7	0.91	3	339	5.468E+07	1.854E+10	1.15375E-14	217	414	15
24	7	0.91	3	354	6.764E+07	2.179E+10	2.18083E-12	221	474	17
24	7	0.91	3	328	4.402E+07	1.280E+10	5.96969E-15	205	340	11
24	7	0.91	3	319	4.863E+07	1.661E+10	2.34541E-14	208	381	14
24	7	0.91	3	250	4.130E+07	1.189E+10	3.81653E-14	128	414	15
24	7	0.91	3	337	5.486E+07	1.970E+10	6.37983E-14	217	412	11
24	7	0.91	3	326	5.177E+07	2.014E+10	1.6597E-14	223	411	14
24	7	0.91	3	358	6.139E+07	2.167E+10	1.3977E-14	238	429	21
24	7	0.91	3	348	6.848E+07	2.820E+10	7.56352E-15	229	477	21
24	7	0.91	3	306	3.957E+07	1.363E+10	7.54197E-15	203	314	15
24	7	0.91	3	376	6.993E+07	2.832E+10	2.38016E-14	245	470	16
24	7	0.91	3	392	8.272E+07	3.482E+10	3.80044E-15	261	522	14
24	7	0.91	3	385	4.973E+07	2.248E+10	3.35925E-15	277	318	11
24	7	0.91	3	356	5.582E+07	2.236E+10	4.1919E-15	231	400	13
24	7	0.91	3	357	5.464E+07	2.020E+10	6.79054E-15	235	383	14
24	7	0.91	3	368	5.781E+07	1.809E+10	4.11627E-14	257	395	16
24	7	0.91	3	314	5.477E+07	1.848E+10	8.33538E-15	209	437	17
24	7	0.91	3	317	6.236E+07	1.964E+10	8.39351E-15	177	501	20
24	7	0.91	3	348	4.339E+07	1.625E+10	6.02082E-15	259	314	14
24	7	0.91	3	350	4.605E+07	1.481E+10	9.97784E-14	241	336	6
24	7	0.91	3	301	4.812E+07	1.416E+10	2.74849E-14	198	405	12
24	7	0.91	3	338	4.977E+07	1.565E+10	1.95803E-14	221	369	11
24	7	0.91	3	296	5.183E+07	1.517E+10	5.29122E-15	178	431	20
24	7	0.91	3	341	6.268E+07	2.492E+10	3.79652E-15	209	468	10
24	7	0.91	3	277	4.328E+07	1.539E+10	3.79947E-14	162	396	14
24	7	0.91	3	281	5.628E+07	1.964E+10	5.20532E-14	165	483	24
24	7	0.91	3	341	4.141E+07	1.313E+10	7.02133E-15	217	313	9
24	7	0.91	3	299	3.570E+07	1.016E+10	8.6586E-15	200	306	13
24	7	0.91	3	364	7.270E+07	2.992E+10	1.74465E-14	237	502	16
24	7	0.91	3	375	5.793E+07	2.104E+10	4.40105E-15	245	391	9
24	7	0.91	3	422	6.270E+07	2.667E+10	1.19205E-13	311	362	13
24	7	0.91	3	293	6.095E+07	2.075E+10	2.02117E-12	163	507	19

1	2	3	4	5	6	7	8	9	10	11
24	7	0.91	3	391	8.272E+07	3.367E+10	7.2223E-15	255	519	22
24	7	0.91	3	353	5.168E+07	1.595E+10	1.85435E-14	237	375	9
24	7	0.91	3	343	4.806E+07	1.646E+10	6.61016E-14	227	359	10
24	7	0.91	3	304	4.539E+07	1.377E+10	3.50758E-15	170	391	11
24	7	0.91	3	263	3.852E+07	1.175E+10	3.45565E-15	167	376	18
24	7	0.91	3	332	5.673E+07	1.912E+10	5.37667E-15	210	436	15
24	7	0.91	3	279	5.148E+07	1.496E+10	5.29254E-13	175	468	19
24	7	0.91	3	345	6.016E+07	2.289E+10	9.13954E-15	217	424	26
24	7	0.91	3	340	6.440E+07	2.241E+10	4.00255E-14	201	481	18
24	7	0.91	3	354	5.560E+07	1.822E+10	1.47642E-14	246	389	15
24	7	0.91	3	330	5.211E+07	1.910E+10	1.53566E-14	212	399	10
24	7	0.91	3	327	3.283E+07	9.327E+09	1.3093E-13	236	262	5
24	7	0.91	3	316	5.188E+07	1.577E+10	6.10075E-14	205	416	13
24	7	0.91	3	360	4.432E+07	1.667E+10	1.04383E-14	251	311	5
24	7	0.91	3	416	8.554E+07	4.092E+10	2.20357E-14	289	508	21
24	7	0.91	3	344	8.273E+07	3.039E+10	5.44849E-15	223	591	25
24	7	0.91	3	270	3.827E+07	9.267E+09	2.39761E-13	155	352	12
24	7	0.91	3	320	5.311E+07	1.741E+10	2.65789E-14	194	421	8
24	7	0.91	3	357	5.088E+07	1.677E+10	4.12474E-13	248	355	17
24	7	0.91	3	334	4.423E+07	1.629E+10	5.01187E-15	213	332	9
24	7	0.91	3	295	4.571E+07	1.156E+10	3.29479E-14	178	394	10
24	7	0.91	3	313	6.325E+07	2.134E+10	4.46401E-15	183	504	27
24	7	0.91	3	307	4.614E+07	1.222E+10	4.75341E-15	194	376	18
24	7	0.91	3	348	7.137E+07	2.422E+10	1.89198E-12	199	517	24
24	7	0.91	3	307	4.281E+07	1.242E+10	1.01757E-14	192	354	11
24	7	0.91	3	344	5.268E+07	1.804E+10	3.18568E-14	225	372	16
24	7	0.91	3	330	3.206E+07	8.274E+09	8.13972E-15	242	252	7
24	7	0.91	3	354	4.049E+07	1.544E+10	1.06407E-14	236	292	6
24	7	0.91	3	356	6.301E+07	2.350E+10	3.57103E-15	229	431	19
24	7	0.91	3	247	3.177E+07	7.990E+09	2.95E-13	150	331	13
24	7	0.91	3	387	4.570E+07	1.759E+10	7.54113E-15	276	297	12
24	7	0.91	3	336	5.537E+07	1.993E+10	1.27171E-13	218	422	14
24	7	0.91	3	334	4.062E+07	1.483E+10	3.12817E-14	230	315	9
24	7	0.91	3	375	6.457E+07	2.283E+10	1.0851E-14	258	432	14
24	7	0.91	3	269	4.289E+07	1.104E+10	1.75067E-14	167	394	16
24	7	0.91	3	314	4.385E+07	1.464E+10	6.65585E-14	204	348	12
24	7	0.91	3	343	5.442E+07	2.094E+10	1.43842E-14	236	396	12
24	7	0.91	3	320	5.255E+07	2.101E+10	8.27243E-14	196	412	15
24	7	0.91	3	359	6.011E+07	2.367E+10	8.14142E-13	250	411	16
24	7	0.91	3	332	6.003E+07	1.941E+10	1.61156E-14	207	445	12
24	7	0.91	3	326	3.796E+07	1.148E+10	1.74665E-14	214	297	6
24	7	0.91	3	359	4.697E+07	1.341E+10	3.84336E-14	259	328	13
24	7	0.91	3	359	5.466E+07	1.942E+10	4.40256E-15	242	374	16
24	7	0.91	3	348	5.301E+07	1.799E+10	1.17261E-14	230	376	18
24	7	0.91	3	375	7.415E+07	2.854E+10	6.70132E-15	240	489	25
24	7	0.91	3	393	5.715E+07	2.168E+10	7.16557E-14	278	360	13
24	7	0.91	3	313	5.327E+07	1.879E+10	6.52868E-15	191	415	22
24	7	0.91	3	357	4.101E+07	1.417E+10	1.1792E-13	249	290	9
24	7	0.91	3	317	3.084E+07	1.083E+10	1.01709E-14	242	243	12
24	7	0.91	3	348	4.450E+07	1.581E+10	1.45153E-14	245	325	7
24	7	0.91	3	405	6.557E+07	2.863E+10	9.02879E-14	270	413	12
24	7	0.91	3	369	5.262E+07	2.179E+10	4.40844E-15	251	357	12
24	7	0.91	3	368	4.328E+07	1.519E+10	1.46822E-13	254	301	7
24	7	0.91	3	340	4.433E+07	1.522E+10	1.85271E-14	228	326	14

1	2	3	4	5	6	7	8	9	10	11
24	7	0.91	3	351	4.529E+07	1.671E+10	8.17615E-15	244	325	11
24	7	0.91	3	361	4.987E+07	1.520E+10	4.58294E-15	270	336	15
24	7	0.91	3	367	7.060E+07	2.877E+10	1.28438E-14	237	474	16
24	7	0.91	3	318	3.331E+07	8.992E+09	3.05063E-15	232	263	8
24	7	0.91	3	305	6.283E+07	2.229E+10	3.5673E-15	170	522	20
24	7	0.91	3	297	4.389E+07	1.249E+10	2.28883E-14	183	375	12
24	7	0.91	3	334	5.682E+07	2.062E+10	3.49429E-14	209	431	13
24	7	0.91	3	274	2.758E+07	6.821E+09	5.48159E-13	183	257	4
24	7	0.91	3	347	6.389E+07	2.708E+10	8.11559E-14	228	457	14
24	7	0.91	3	322	6.179E+07	2.229E+10	3.13522E-13	198	480	17
24	7	0.91	3	307	3.123E+07	7.835E+09	2.72329E-14	214	261	9
24	7	0.91	3	351	4.365E+07	1.691E+10	1.14176E-14	255	319	9
24	7	0.91	3	332	5.278E+07	2.009E+10	1.06125E-13	200	398	8
24	7	0.91	3	331	5.508E+07	1.923E+10	9.58541E-15	217	413	16
24	7	0.91	3	291	3.660E+07	1.127E+10	1.07655E-13	190	326	12
24	7	0.91	3	351	6.173E+07	2.569E+10	1.30043E-13	215	437	16
24	7	0.91	3	331	6.212E+07	2.150E+10	1.80474E-14	208	471	14
24	7	0.91	3	371	5.231E+07	1.844E+10	4.98069E-15	262	340	15
24	7	0.91	3	279	4.504E+07	1.148E+10	4.38194E-14	163	414	21
24	7	0.91	3	314	5.264E+07	2.073E+10	1.7103E-14	208	419	11
24	7	0.91	3	313	3.344E+07	9.932E+09	3.68269E-15	229	273	11
24	7	0.91	3	345	9.492E+07	4.348E+10	3.48892E-14	201	656	33
24	7	0.91	3	312	4.484E+07	1.382E+10	4.48567E-15	177	367	15
24	7	0.91	3	339	4.580E+07	1.462E+10	1.42075E-13	230	347	9
24	7	0.91	3	367	5.979E+07	2.521E+10	6.65984E-15	244	407	12
24	7	0.91	3	307	5.715E+07	2.326E+10	1.36979E-14	187	467	21
24	7	0.91	3	338	4.777E+07	1.503E+10	9.6498E-15	210	360	10
24	7	0.91	3	313	5.125E+07	1.638E+10	4.56759E-15	187	410	18
24	7	0.91	3	246	3.156E+07	7.267E+09	6.08872E-15	144	323	9
24	7	0.91	3	397	9.618E+07	3.980E+10	3.23252E-14	252	584	16
24	7	0.91	3	389	8.020E+07	3.184E+10	2.43372E-14	248	522	22
24	7	0.91	3	325	4.127E+07	1.073E+10	2.15995E-14	236	316	16
24	7	0.91	3	338	3.972E+07	1.447E+10	4.56919E-14	239	300	6
24	7	0.91	3	296	3.717E+07	9.689E+09	3.41219E-13	207	313	12
24	7	0.91	3	317	6.550E+07	1.953E+10	7.54029E-15	195	521	26
24	7	0.91	3	351	6.297E+07	2.172E+10	1.2315E-14	210	454	16
24	7	0.91	3	331	3.413E+07	1.121E+10	4.6323E-15	242	264	6
24	7	0.91	3	402	8.578E+07	3.588E+10	2.43324E-14	263	534	15
24	7	0.91	3	350	5.914E+07	2.360E+10	2.67128E-14	220	421	16
30	7	0.91	3	387	6.628E+07	2.550E+10	3.72947E-15	247	439	15
30	7	0.91	3	418	6.034E+07	2.488E+10	8.67605E-13	299	365	17
30	7	0.91	3	472	8.868E+07	5.138E+10	1.47242E-14	327	456	16
30	7	0.91	3	444	8.359E+07	4.601E+10	6.99641E-14	299	483	12
30	7	0.91	3	426	5.832E+07	1.985E+10	7.87686E-15	307	342	14
30	7	0.91	3	503	8.723E+07	5.136E+10	4.27235E-13	357	450	11
30	7	0.91	3	287	2.931E+07	8.594E+09	2.81398E-14	205	263	9
30	7	0.91	3	447	9.134E+07	4.038E+10	3.84003E-15	302	516	17
30	7	0.91	3	412	4.613E+07	1.736E+10	9.78968E-15	317	281	8
30	7	0.91	3	349	4.362E+07	1.449E+10	6.29957E-15	252	309	14
30	7	0.91	3	453	6.162E+07	2.570E+10	6.23903E-15	347	335	16
30	7	0.91	3	399	6.303E+07	2.120E+10	8.92085E-15	281	391	14
30	7	0.91	3	412	4.905E+07	2.328E+10	3.87549E-14	300	310	5
30	7	0.91	3	411	8.442E+07	3.655E+10	1.69978E-14	284	523	19
30	7	0.91	3	400	7.203E+07	2.404E+10	7.93928E-13	266	451	11

1	2	3	4	5	6	7	8	9	10	11
30	7	0.91	3	454	6.902E+07	3.229E+10	3.4675E-15	324	390	10
30	7	0.91	3	393	7.592E+07	3.438E+10	9.24954E-15	255	486	17
30	7	0.91	3	417	7.011E+07	3.586E+10	4.83463E-15	282	424	17
30	7	0.91	3	399	5.686E+07	2.445E+10	1.63755E-14	276	365	12
30	7	0.91	3	481	9.315E+07	4.867E+10	3.3232E-14	338	475	17
30	7	0.91	3	362	5.794E+07	2.404E+10	1.71295E-14	237	413	10
30	7	0.91	3	358	4.326E+07	1.563E+10	2.74588E-14	255	309	9
30	7	0.91	3	413	6.899E+07	2.843E+10	4.42361E-15	279	416	13
30	7	0.91	3	447	1.060E+08	5.868E+10	6.884E-15	281	597	15
30	7	0.91	3	422	5.138E+07	1.895E+10	4.65741E-15	306	318	11
30	7	0.91	3	415	8.014E+07	4.133E+10	1.76109E-13	272	490	19
30	7	0.91	3	376	6.527E+07	2.863E+10	1.56535E-14	236	431	19
30	7	0.91	3	397	7.222E+07	3.256E+10	1.00977E-14	252	472	16
30	7	0.91	3	420	7.955E+07	3.558E+10	1.00301E-14	289	467	14
30	7	0.91	3	399	5.736E+07	2.070E+10	4.15982E-15	299	360	11
30	7	0.91	3	442	1.116E+08	6.474E+10	2.77896E-14	284	638	18
30	7	0.91	3	348	7.657E+07	2.995E+10	2.06336E-13	214	550	19
30	7	0.91	3	426	3.249E+07	1.171E+10	5.77265E-15	346	193	5
30	7	0.91	3	431	7.389E+07	3.908E+10	3.58607E-15	313	426	19
30	7	0.91	3	413	5.606E+07	2.505E+10	5.64731E-15	284	343	9
30	7	0.91	3	420	8.558E+07	4.521E+10	1.81607E-13	285	514	12
30	7	0.91	3	366	8.551E+07	3.300E+10	7.97147E-15	224	583	29
30	7	0.91	3	429	6.453E+07	2.388E+10	1.33488E-14	313	383	12
30	7	0.91	3	463	8.398E+07	4.530E+10	3.79638E-15	336	449	23
30	7	0.91	3	374	4.507E+07	1.384E+10	3.56364E-15	261	304	9
30	7	0.91	3	346	5.309E+07	1.900E+10	7.49797E-14	238	399	13
30	7	0.91	3	510	9.914E+07	5.643E+10	1.20375E-13	367	493	13
30	7	0.91	3	470	4.477E+07	2.083E+10	1.316E-14	373	243	4
30	7	0.91	3	430	5.706E+07	2.412E+10	1.93753E-13	310	340	9
30	7	0.91	3	454	8.233E+07	3.928E+10	2.25881E-13	297	459	10
30	7	0.91	3	376	5.344E+07	1.983E+10	6.5001E-15	249	360	10
30	7	0.91	3	453	6.753E+07	2.747E+10	1.82441E-14	329	374	14
30	7	0.91	3	424	8.194E+07	3.449E+10	5.02472E-15	286	483	17
30	7	0.91	3	399	5.600E+07	1.850E+10	1.12383E-13	274	367	10
30	7	0.91	3	391	4.394E+07	1.734E+10	2.84656E-14	290	292	8
30	7	0.91	3	405	5.681E+07	2.485E+10	1.25521E-14	292	349	11
30	7	0.91	3	407	8.599E+07	4.587E+10	1.37644E-11	250	525	19
30	7	0.91	3	428	8.908E+07	4.111E+10	6.65112E-14	274	531	17
30	7	0.91	3	395	5.912E+07	2.425E+10	2.51448E-13	273	383	9
30	7	0.91	3	384	5.245E+07	2.025E+10	7.44684E-14	266	348	10
30	7	0.91	3	380	5.807E+07	2.429E+10	4.09861E-15	248	390	15
30	7	0.91	3	374	5.169E+07	1.920E+10	1.18335E-14	258	352	11
30	7	0.91	3	400	6.072E+07	2.371E+10	2.41433E-14	279	372	13
30	7	0.91	3	390	7.752E+07	3.546E+10	5.99494E-14	255	488	19
30	7	0.91	3	380	7.643E+07	3.213E+10	1.14182E-14	239	509	22
30	7	0.91	3	438	9.158E+07	4.247E+10	4.91268E-15	283	536	18
30	7	0.91	3	407	1.102E+08	5.224E+10	1.00533E-14	261	666	39
30	7	0.91	3	423	6.839E+07	3.339E+10	4.56921E-14	301	407	7
30	7	0.91	3	449	8.424E+07	3.727E+10	3.92186E-14	285	470	15
30	7	0.91	3	402	8.381E+07	3.471E+10	6.16E-14	271	523	22
30	7	0.91	3	385	8.021E+07	3.447E+10	1.06359E-14	252	518	21
30	7	0.91	3	427	7.578E+07	3.582E+10	5.04316E-15	292	459	12
30	7	0.91	3	429	7.095E+07	2.991E+10	5.63518E-14	305	418	14
30	7	0.91	3	393	8.615E+07	3.877E+10	5.5453E-15	264	534	23

1	2	3	4	5	6	7	8	9	10	11
30	7	0.91	3	493	1.432E+08	9.394E+10	4.37095E-14	333	721	24
30	7	0.91	3	505	1.087E+08	7.443E+10	1.00143E-14	346	546	13
30	7	0.91	3	431	8.447E+07	4.189E+10	4.02859E-13	283	496	14
30	7	0.91	3	439	9.086E+07	4.261E+10	1.67267E-12	290	518	15
30	7	0.91	3	434	6.394E+07	2.856E+10	1.71748E-14	315	359	16
30	7	0.91	3	397	5.399E+07	2.605E+10	4.33593E-15	287	336	7
30	7	0.91	3	346	6.525E+07	2.537E+10	1.7348E-14	206	470	28
30	7	0.91	3	442	6.855E+07	4.234E+10	1.27762E-13	308	389	11
30	7	0.91	3	442	9.481E+07	4.689E+10	1.93845E-13	302	535	16
30	7	0.91	3	369	4.981E+07	1.807E+10	1.23198E-13	264	338	8
30	7	0.91	3	389	5.072E+07	2.420E+10	4.70553E-14	282	339	6
30	7	0.91	3	459	6.927E+07	3.107E+10	6.24953E-15	330	383	12
30	7	0.91	3	393	9.901E+07	4.464E+10	2.65066E-13	254	624	25
30	7	0.91	3	343	4.245E+07	1.239E+10	5.82342E-14	239	327	7
30	7	0.91	3	451	1.110E+08	5.159E+10	5.32464E-15	308	620	25
30	7	0.91	3	477	9.658E+07	4.482E+10	6.44481E-14	334	500	16
30	7	0.91	3	431	9.973E+07	4.508E+10	6.11725E-15	293	576	21
30	7	0.91	3	445	1.004E+08	5.353E+10	7.11575E-15	300	569	20
30	7	0.91	3	422	9.513E+07	4.667E+10	3.61155E-13	274	555	22
30	7	0.91	3	419	9.043E+07	4.618E+10	9.10177E-15	269	539	20
30	7	0.91	3	434	6.996E+07	3.090E+10	4.9121E-15	287	412	13
30	7	0.91	3	394	6.050E+07	2.403E+10	7.88595E-15	276	396	12
30	7	0.91	3	426	1.030E+08	4.451E+10	6.90514E-14	271	602	23
30	7	0.91	3	396	7.374E+07	3.442E+10	1.58916E-14	272	465	16
30	7	0.91	3	469	7.673E+07	4.551E+10	5.78193E-15	328	408	12
30	7	0.91	3	402	5.963E+07	2.151E+10	1.03988E-14	273	381	11
30	7	0.91	3	400	7.364E+07	3.815E+10	1.7712E-14	274	456	21
30	7	0.91	3	427	5.604E+07	2.263E+10	7.4213E-14	306	334	6
30	7	0.91	3	457	9.406E+07	4.836E+10	5.01176E-14	319	506	18
30	7	0.91	3	430	7.191E+07	4.274E+10	5.05649E-14	309	415	13
30	7	0.91	3	384	5.577E+07	2.452E+10	7.88195E-15	277	360	14
30	7	0.91	3	451	8.978E+07	4.672E+10	8.37788E-13	313	492	22
30	7	0.91	3	448	8.231E+07	3.574E+10	5.13917E-14	323	460	15
30	7	0.91	3	353	5.623E+07	2.103E+10	3.94591E-15	231	400	18
30	7	0.91	3	419	8.596E+07	4.143E+10	1.62442E-13	275	512	22
30	7	0.91	3	446	7.530E+07	3.953E+10	4.02662E-14	308	426	13
30	7	0.91	3	442	7.795E+07	3.622E+10	2.00951E-14	306	440	12
30	7	0.91	3	440	6.404E+07	3.789E+10	1.67723E-13	314	362	12
30	7	0.91	3	460	6.088E+07	2.956E+10	6.05013E-15	348	332	9
30	7	0.91	3	444	5.516E+07	2.527E+10	5.25847E-14	330	313	6
30	7	0.91	3	466	6.684E+07	3.114E+10	1.51377E-14	360	365	10
30	7	0.91	3	447	5.042E+07	1.835E+10	1.85344E-13	364	280	7
30	7	0.91	3	369	5.447E+07	2.502E+10	2.72555E-13	238	382	12
30	7	0.91	3	389	5.288E+07	2.498E+10	7.09056E-15	272	345	7
30	7	0.91	3	415	7.949E+07	3.212E+10	4.56011E-15	274	476	17
30	7	0.91	3	326	4.121E+07	1.520E+10	1.49221E-14	217	326	9
30	7	0.91	3	428	4.697E+07	2.040E+10	2.84422E-14	330	279	7
30	7	0.91	3	448	8.948E+07	3.985E+10	5.70869E-15	321	507	18
30	7	0.91	3	457	6.012E+07	2.806E+10	1.71493E-14	355	334	11
30	7	0.91	3	470	1.025E+08	5.794E+10	8.79557E-15	311	548	20
30	7	0.91	3	434	5.722E+07	2.839E+10	7.60867E-14	309	342	7
30	7	0.91	3	445	5.547E+07	2.674E+10	2.05051E-14	342	313	11
30	7	0.91	3	424	7.170E+07	3.326E+10	1.48101E-13	297	415	11
30	7	0.91	3	358	3.421E+07	1.303E+10	6.5312E-15	279	238	10

1	2	3	4	5	6	7	8	9	10	11
30	7	0.91	3	403	4.830E+07	1.924E+10	2.90655E-13	291	309	5
30	7	0.91	3	415	5.517E+07	2.254E+10	3.2208E-14	290	337	7
30	7	0.91	3	393	7.118E+07	2.945E+10	4.38578E-13	269	466	15
30	7	0.91	3	361	5.728E+07	2.598E+10	6.88434E-15	242	403	17
30	7	0.91	3	441	5.504E+07	2.613E+10	7.50162E-15	330	315	8
30	7	0.91	3	439	5.279E+07	1.881E+10	4.75983E-15	339	301	11
30	7	0.91	3	415	7.003E+07	2.901E+10	1.01976E-14	281	430	11
30	7	0.91	3	377	6.179E+07	2.221E+10	7.14548E-15	249	427	15
30	7	0.91	3	346	5.684E+07	2.227E+10	2.61581E-13	232	408	15
30	7	0.91	3	370	3.669E+07	1.172E+10	2.16813E-14	276	250	8
30	7	0.91	3	462	7.622E+07	2.909E+10	6.327E-15	310	418	14
30	7	0.91	3	427	6.951E+07	2.157E+10	6.63305E-13	304	415	12
30	7	0.91	3	405	6.931E+07	3.142E+10	8.69498E-15	268	433	11
30	7	0.91	3	434	1.040E+08	5.110E+10	4.4269E-13	267	608	25
30	7	0.91	3	433	5.579E+07	2.133E+10	9.28111E-15	318	325	9
30	7	0.91	3	392	5.095E+07	1.929E+10	4.73896E-15	275	325	12
30	7	0.91	3	445	8.287E+07	4.185E+10	1.11752E-14	301	466	23
30	7	0.91	3	337	3.086E+07	9.805E+09	3.61653E-14	243	235	6
30	7	0.91	3	404	6.378E+07	2.141E+10	3.52254E-14	264	398	15
30	7	0.91	3	397	6.619E+07	2.661E+10	3.53648E-14	252	416	12
30	7	0.91	3	413	7.793E+07	4.110E+10	9.0184E-14	268	488	11
30	7	0.91	3	444	1.046E+08	5.131E+10	8.65966E-15	277	592	22
30	7	0.91	3	385	7.138E+07	3.636E+10	1.53759E-14	246	469	15
30	7	0.91	3	475	5.383E+07	2.223E+10	2.19531E-14	365	287	8
30	7	0.91	3	380	4.361E+07	1.424E+10	3.22978E-15	289	290	10
30	7	0.91	3	396	4.419E+07	1.704E+10	9.64869E-15	281	281	9
30	7	0.91	3	461	9.062E+07	4.853E+10	1.03479E-14	318	489	13
30	7	0.91	3	459	4.898E+07	2.111E+10	9.86151E-15	355	270	6
30	7	0.91	3	381	6.902E+07	2.569E+10	2.14177E-14	262	464	17
30	7	0.91	3	455	7.414E+07	4.171E+10	9.28893E-15	307	417	8
30	7	0.91	3	437	7.059E+07	3.033E+10	9.04647E-15	303	404	11
30	7	0.91	3	402	7.946E+07	3.027E+10	2.34535E-14	255	500	14
30	7	0.91	3	351	5.347E+07	1.726E+10	4.48983E-14	236	386	16
30	7	0.91	3	485	1.075E+08	5.547E+10	6.84826E-15	331	552	30
30	7	0.91	3	430	7.205E+07	3.484E+10	8.49432E-15	294	422	16
30	7	0.91	3	372	5.636E+07	1.807E+10	7.44806E-15	248	379	13
30	7	0.91	3	490	1.277E+08	7.889E+10	1.73126E-14	328	647	26
30	7	0.91	3	462	1.019E+08	5.556E+10	7.84402E-15	308	558	16
30	7	0.91	3	478	1.073E+08	5.397E+10	2.20454E-14	326	560	18
30	7	0.91	3	413	1.081E+08	5.440E+10	2.41154E-14	244	663	27
30	7	0.91	3	454	8.606E+07	4.311E+10	1.47578E-13	305	483	15
30	7	0.8	3	441	5.519E+07	2.586E+10	5.34752E-15	339	316	16
30	7	0.95	3	403	7.495E+07	3.543E+10	2.16633E-14	260	467	12
30	7	0.95	3	587	2.595E+08	1.590E+11	6.16706E-15	418	779	69
30	7	0.97	3	649	2.107E+08	1.080E+11	5.45764E-15	513	545	50
10	7	0.9	4	231	8.233E+07	1.459E+10	2.2619E-15	0	511	51
24	7	0.9	4	519	3.042E+08	1.331E+11	2.70076E-15	0	959	65
24	7	0.9	4	472	2.993E+08	1.407E+11	3.10282E-15	0	1045	81
24	7	0.9	4	487	2.195E+08	9.317E+10	2.84346E-15	0	787	38
24	7	0.9	4	441	3.124E+08	1.543E+11	4.02991E-15	0	1126	91
24	7	0.9	4	469	2.615E+08	1.402E+11	2.60793E-15	0	886	73
24	7	0.9	4	496	2.955E+08	1.381E+11	3.29514E-15	0	998	66
24	7	0.9	4	553	3.472E+08	1.494E+11	2.80833E-15	0	969	58
24	7	0.9	4	571	3.720E+08	2.071E+11	2.87765E-15	0	1061	56

1	2	3	4	5	6	7	8	9	10	11
24	7	0.9	4	467	2.121E+08	7.528E+10	3.33324E-15	0	798	33
48	7	0.9	4	1008	9.775E+08	8.391E+11	4.60907E-15	0	1650	73
48	7	0.9	4	926	9.476E+08	7.535E+11	4.95425E-15	0	1737	89
24	7	0.9	4	547	3.996E+08	2.751E+11	3.28443E-15	0	1151	96
24	7	0.9	4	228	2.417E+07	3.262E+09	1.87748E-15	0	287	5
24	7	0.9	4	184	1.753E+07	2.375E+09	2.16922E-15	0	247	3
24	7	0.9	4	244	3.095E+07	5.943E+09	1.92293E-14	0	336	6
24	7	0.9	4	265	3.461E+07	6.482E+09	2.97662E-15	0	342	4
24	7	0.9	4	233	2.649E+07	7.107E+09	1.52212E-15	0	301	4
24	7	0.9	4	229	2.700E+07	3.359E+09	3.16988E-15	0	314	5
24	7	0.9	4	209	2.206E+07	4.228E+09	1.78159E-15	0	286	3
24	7	0.9	4	222	2.321E+07	4.975E+09	1.81294E-15	0	273	5
24	7	0.9	4	158	1.132E+07	1.278E+09	1.76337E-15	0	194	3
24	7	0.9	4	181	1.317E+07	3.254E+09	8.67492E-16	0	211	2
24	7	0.9	4	181	1.661E+07	1.782E+09	3.75731E-15	0	244	5
24	7	0.9	4	264	3.560E+07	6.136E+09	1.3123E-15	0	353	6
24	7	0.9	4	216	2.466E+07	3.019E+09	1.69004E-15	0	296	7
24	7	0.9	4	247	3.303E+07	6.946E+09	1.78168E-15	0	341	9
24	7	0.9	4	234	2.865E+07	9.932E+09	1.3831E-15	0	337	5
24	7	0.9	4	249	3.033E+07	5.938E+09	1.58837E-15	0	324	5
24	7	0.9	4	192	2.032E+07	4.077E+09	1.30066E-15	0	284	8
24	7	0.9	4	240	3.121E+07	4.564E+09	1.16471E-15	0	339	8
24	7	0.9	4	241	3.011E+07	4.286E+09	1.18177E-15	0	330	5
24	7	0.9	4	276	3.705E+07	6.589E+09	1.47429E-15	0	346	7
24	7	0.9	4	237	3.228E+07	5.551E+09	4.40165E-15	0	353	7
24	7	0.9	4	212	2.344E+07	3.305E+09	3.77842E-15	0	295	3
24	7	0.9	4	229	2.660E+07	5.104E+09	6.2182E-15	0	300	5
24	7	0.9	4	273	3.610E+07	1.127E+10	3.26456E-15	0	348	5
24	7	0.9	4	232	2.567E+07	5.827E+09	1.30514E-15	0	295	4
24	7	0.9	4	240	2.854E+07	3.995E+09	1.35219E-15	0	316	5
24	7	0.9	4	209	2.099E+07	5.792E+09	2.33705E-15	0	265	5
24	7	0.9	4	228	2.730E+07	5.198E+09	2.76249E-15	0	311	8
24	7	0.9	4	220	2.316E+07	3.505E+09	2.75733E-15	0	277	3
24	7	0.9	4	195	1.953E+07	3.222E+09	1.92786E-15	0	268	5
24	7	0.9	4	220	2.450E+07	3.515E+09	1.35529E-15	0	292	5
24	7	0.9	4	230	2.912E+07	5.301E+09	3.17228E-15	0	331	8
24	7	0.9	4	236	3.000E+07	4.190E+09	2.35427E-15	0	331	8
24	7	0.9	4	215	2.388E+07	2.767E+09	1.31089E-15	0	288	7
24	7	0.9	4	210	2.251E+07	5.007E+09	3.26282E-15	0	283	6
24	7	0.9	4	243	3.162E+07	4.040E+09	1.40645E-15	0	334	7
24	7	0.9	4	271	4.113E+07	9.989E+09	1.72628E-15	0	397	8
24	7	0.9	4	202	2.087E+07	3.324E+09	3.02763E-15	0	271	5
24	7	0.9	4	187	1.696E+07	3.484E+09	2.42205E-15	0	248	3
24	7	0.9	4	258	3.601E+07	1.076E+10	1.24223E-15	0	365	7
24	7	0.9	4	237	2.870E+07	4.906E+09	8.36637E-15	0	319	4
24	7	0.9	4	201	2.029E+07	2.498E+09	3.63987E-15	0	267	2
24	7	0.9	4	220	2.881E+07	4.636E+09	1.87407E-15	0	341	16
24	7	0.9	4	255	3.345E+07	5.332E+09	2.17301E-15	0	348	5
24	7	0.9	4	216	2.240E+07	5.246E+09	2.14552E-15	0	288	5
24	7	0.9	4	210	2.290E+07	8.218E+09	2.53377E-15	0	298	4
24	7	0.9	4	210	2.349E+07	4.599E+09	2.75523E-15	0	293	9
24	7	0.9	4	202	2.012E+07	5.447E+09	1.85026E-15	0	270	5
24	7	0.9	4	241	3.232E+07	5.037E+09	2.00002E-15	0	331	9
24	7	0.9	4	227	3.294E+07	6.140E+09	1.88841E-15	0	358	15

1	2	3	4	5	6	7	8	9	10	11
24	7	0.9	4	225	2.559E+07	3.676E+09	4.29016E-15	0	299	6
24	7	0.9	4	221	2.432E+07	4.047E+09	3.03779E-15	0	287	5
24	7	0.9	4	198	2.304E+07	3.314E+09	1.46306E-15	0	311	6
24	7	0.9	4	219	2.425E+07	4.500E+09	3.66051E-15	0	298	4
24	7	0.9	4	244	3.233E+07	8.061E+09	1.76544E-15	0	337	6
24	7	0.9	4	260	3.675E+07	1.609E+10	1.32667E-15	0	357	7
24	7	0.9	4	217	2.278E+07	3.393E+09	1.6077E-15	0	286	6
24	7	0.9	4	293	5.654E+07	1.586E+10	1.97283E-15	0	462	12
24	7	0.9	4	236	3.123E+07	4.825E+09	4.4026E-15	0	334	9
24	7	0.9	4	229	3.109E+07	4.438E+09	1.40756E-15	0	350	9
24	7	0.9	4	223	2.494E+07	3.609E+09	1.47571E-15	0	293	6
24	7	0.9	4	254	3.077E+07	1.926E+10	1.67553E-15	0	313	5
24	7	0.9	4	265	3.746E+07	6.743E+09	1.95058E-15	0	359	6
24	7	0.9	4	233	2.543E+07	1.159E+10	1.90028E-15	0	286	4
24	7	0.9	4	247	3.098E+07	7.062E+09	1.02E-15	0	330	4
24	7	0.9	4	177	1.614E+07	2.763E+09	2.64782E-15	0	246	5
24	7	0.9	4	217	2.261E+07	3.105E+09	1.61843E-15	0	275	3
24	7	0.9	4	230	2.814E+07	5.232E+09	2.45338E-15	0	327	5
24	7	0.9	4	215	2.395E+07	2.185E+09	1.78223E-15	0	286	7
24	7	0.9	4	214	2.214E+07	5.332E+09	1.37903E-15	0	275	4
24	7	0.9	4	215	2.586E+07	4.585E+09	3.65254E-15	0	315	5
36	7	0.9	4	261	3.586E+07	6.359E+09	1.53675E-15	0	364	6
36	7	0.9	4	368	7.136E+07	1.866E+10	2.07383E-15	0	493	10
36	7	0.9	4	337	5.798E+07	2.891E+10	1.41639E-15	0	444	6
36	7	0.9	4	343	5.560E+07	1.777E+10	2.41204E-15	0	425	4
36	7	0.9	4	348	6.820E+07	1.966E+10	1.99596E-15	0	516	14
36	7	0.9	4	316	4.544E+07	1.159E+10	1.7149E-15	0	383	4
36	7	0.9	4	365	6.950E+07	3.645E+10	2.43673E-15	0	485	6
36	7	0.9	4	338	5.854E+07	1.593E+10	2.45374E-15	0	445	7
36	7	0.9	4	372	6.534E+07	5.282E+10	2.77323E-15	0	449	5
36	7	0.9	4	291	3.937E+07	6.993E+09	2.14904E-15	0	362	6
36	7	0.9	4	288	4.274E+07	7.763E+09	1.84963E-15	0	395	9
36	7	0.9	4	359	6.687E+07	1.568E+10	1.62776E-15	0	492	8
36	7	0.9	4	368	6.568E+07	2.139E+10	7.49545E-15	0	470	8
36	7	0.9	4	371	6.814E+07	1.739E+10	2.50786E-15	0	480	4
36	7	0.9	4	340	5.590E+07	1.343E+10	3.5495E-15	0	430	4
36	7	0.9	4	352	5.954E+07	1.652E+10	2.89013E-15	0	446	7
36	7	0.9	4	306	4.607E+07	1.069E+10	3.53648E-15	0	398	4
36	7	0.9	4	323	5.369E+07	1.010E+10	2.16577E-15	0	430	10
36	7	0.9	4	383	7.115E+07	2.419E+10	7.06531E-15	0	482	7
36	7	0.9	4	343	5.312E+07	3.321E+10	2.75476E-15	0	422	4
36	7	0.9	4	333	5.645E+07	1.186E+10	1.94288E-15	0	444	11
36	7	0.9	4	315	5.507E+07	2.859E+10	1.90364E-15	0	444	11
36	7	0.9	4	339	5.903E+07	1.069E+10	2.33273E-15	0	465	7
36	7	0.9	4	314	4.988E+07	1.314E+10	2.79147E-15	0	419	4
36	7	0.9	4	332	5.838E+07	1.528E+10	2.74463E-15	0	474	7
36	7	0.9	4	297	4.983E+07	9.067E+09	1.11891E-14	0	450	10
36	7	0.9	4	391	8.612E+07	2.195E+10	2.21793E-15	0	577	14
36	7	0.9	4	369	6.243E+07	3.258E+10	2.35321E-15	0	455	3
36	7	0.9	4	368	6.832E+07	1.871E+10	3.04904E-15	0	485	5
36	7	0.9	4	359	7.319E+07	1.548E+10	1.54078E-15	0	523	10
36	7	0.9	4	320	4.909E+07	1.039E+10	2.70384E-15	0	403	6
36	7	0.9	4	351	5.816E+07	1.043E+10	1.499E-15	0	445	5
36	7	0.9	4	329	5.215E+07	2.148E+10	4.64987E-15	0	423	4

1	2	3	4	5	6	7	8	9	10	11
36	7	0.9	4	339	5.731E+07	1.637E+10	1.81788E-15	0	446	4
36	7	0.9	4	340	6.013E+07	1.554E+10	2.28495E-15	0	472	9
36	7	0.9	4	282	3.827E+07	6.646E+09	3.7141E-15	0	364	6
36	7	0.9	4	330	5.048E+07	1.258E+10	2.93635E-15	0	409	5
36	7	0.9	4	321	5.094E+07	9.995E+09	4.50722E-15	0	418	8
12	7	0.9	4	89	5.363E+06	2.882E+08	1.97511E-15	0	156	4
12	7	0.9	4	106	5.985E+06	3.786E+08	1.55107E-15	0	147	4
12	7	0.9	4	111	6.726E+06	5.382E+08	2.37384E-15	0	156	5
12	7	0.9	4	121	7.673E+06	7.550E+08	1.95516E-15	0	162	2
12	7	0.9	4	115	6.675E+06	4.743E+08	2.48407E-15	0	154	2
12	7	0.9	4	127	9.368E+06	7.159E+08	2.28619E-15	0	189	3
12	7	0.9	4	101	4.675E+06	1.956E+08	1.19728E-15	0	122	2
12	7	0.9	4	149	1.615E+07	1.584E+09	2.78531E-15	0	251	7
12	7	0.9	4	144	1.199E+07	1.163E+09	1.98646E-15	0	208	7
12	7	0.9	4	122	8.849E+06	7.737E+08	1.2547E-15	0	187	6
12	7	0.9	4	130	1.085E+07	8.839E+08	2.50276E-15	0	204	6
12	7	0.9	4	111	6.389E+06	3.983E+08	1.71312E-15	0	159	3
12	7	0.9	4	83	3.183E+06	1.581E+08	1.47826E-15	0	109	2
12	7	0.9	4	80	3.125E+06	1.168E+08	9.7534E-16	0	105	3
12	7	0.9	4	119	8.701E+06	6.432E+08	2.30039E-15	0	183	6
12	7	0.9	4	125	8.333E+06	5.665E+08	3.62101E-15	0	166	4
12	7	0.9	4	130	9.963E+06	7.898E+08	2.03253E-15	0	198	6
12	7	0.9	4	158	1.399E+07	1.475E+09	1.18294E-15	0	215	7
12	7	0.9	4	99	3.864E+06	2.708E+08	8.81308E-16	0	109	3
12	7	0.9	4	127	7.616E+06	4.643E+08	2.30258E-15	0	165	2
12	7	0.9	4	120	9.015E+06	5.880E+08	2.30506E-15	0	194	6
12	7	0.9	4	133	9.167E+06	6.317E+08	2.39533E-15	0	179	4
12	7	0.9	4	148	1.294E+07	1.130E+09	1.35224E-15	0	218	4
12	7	0.9	4	122	8.786E+06	7.062E+08	2.44062E-15	0	181	6
12	7	0.9	4	118	7.769E+06	5.571E+08	2.55971E-15	0	168	2
12	7	0.9	4	99	6.277E+06	3.495E+08	9.40751E-15	0	159	4
12	7	0.9	4	96	5.103E+06	2.824E+08	2.90738E-15	0	148	3
12	7	0.9	4	88	3.847E+06	1.981E+08	9.75527E-16	0	120	1
12	7	0.9	4	165	1.825E+07	1.960E+09	1.69214E-15	0	270	13
12	7	0.9	4	101	5.036E+06	2.960E+08	1.12352E-15	0	135	3
12	7	0.9	4	73	2.485E+06	8.206E+07	8.96426E-16	0	94	2
12	7	0.9	4	101	4.491E+06	2.757E+08	1.81583E-15	0	123	1
12	7	0.9	4	141	1.074E+07	8.108E+08	2.14692E-15	0	195	6
12	7	0.9	4	108	8.164E+06	5.083E+08	1.07598E-14	0	195	5
12	7	0.9	4	102	5.644E+06	2.987E+08	2.12197E-15	0	149	3
12	7	0.9	4	123	1.102E+07	9.104E+08	1.12346E-15	0	213	10
12	7	0.9	4	133	8.616E+06	8.734E+08	1.68644E-15	0	166	4
12	7	0.9	4	153	1.364E+07	1.371E+09	1.17072E-15	0	218	4
12	7	0.9	4	109	5.796E+06	4.156E+08	1.39486E-15	0	144	2
12	7	0.9	4	121	7.499E+06	4.484E+08	1.19176E-15	0	159	5
12	7	0.9	4	103	5.462E+06	3.630E+08	1.3172E-15	0	149	3
12	7	0.9	4	85	4.184E+06	1.657E+08	8.22639E-16	0	133	5
12	7	0.9	4	124	8.402E+06	5.795E+08	3.49635E-15	0	182	3
12	7	0.9	4	96	5.553E+06	3.277E+08	9.4523E-16	0	150	4
12	7	0.9	4	107	6.833E+06	5.171E+08	1.25495E-15	0	161	4
12	7	0.9	4	110	7.196E+06	4.731E+08	1.68759E-15	0	173	5
12	7	0.9	4	156	1.413E+07	1.496E+09	1.63388E-15	0	226	6
12	7	0.9	4	133	8.700E+06	7.913E+08	1.563E-15	0	176	4
12	7	0.9	4	100	6.197E+06	4.019E+08	1.25329E-15	0	159	4

1	2	3	4	5	6	7	8	9	10	11
12	7	0.9	4	111	6.496E+06	3.888E+08	1.37063E-15	0	150	5
12	7	0.9	4	143	1.197E+07	1.172E+09	4.16365E-15	0	209	4
12	7	0.9	4	112	6.468E+06	5.079E+08	9.58485E-16	0	151	3
12	7	0.9	4	123	7.931E+06	6.189E+08	1.2691E-15	0	171	5
12	7	0.9	4	104	6.474E+06	4.106E+08	2.4383E-15	0	162	2
12	7	0.9	4	91	4.065E+06	3.151E+08	1.33117E-15	0	119	2
12	7	0.9	4	133	1.147E+07	1.141E+09	1.24081E-15	0	219	6
12	7	0.9	4	101	6.962E+06	3.275E+08	2.09333E-15	0	172	9
12	7	0.9	4	153	1.247E+07	1.095E+09	1.97011E-15	0	206	4
12	7	0.9	4	99	5.473E+06	2.437E+08	1.44655E-15	0	141	3
12	7	0.9	4	103	5.179E+06	5.168E+08	1.33651E-15	0	138	3
12	7	0.9	4	145	1.265E+07	2.151E+09	1.92986E-15	0	221	6
12	7	0.9	4	96	5.087E+06	2.882E+08	1.31345E-15	0	139	4
12	7	0.9	4	139	1.188E+07	1.407E+09	1.04697E-15	0	228	5
12	7	0.9	4	104	5.969E+06	3.828E+08	1.62431E-15	0	149	3
12	7	0.9	4	137	1.079E+07	9.353E+08	3.55532E-15	0	196	6
12	7	0.9	4	136	1.017E+07	7.861E+08	2.74656E-15	0	194	4
12	7	0.9	4	111	6.340E+06	5.379E+08	1.23935E-15	0	156	1
12	7	0.9	4	94	3.928E+06	3.091E+08	1.11896E-15	0	112	2
12	7	0.9	4	125	9.752E+06	8.779E+08	2.0271E-15	0	193	4
12	7	0.9	4	115	7.168E+06	4.899E+08	2.43028E-15	0	162	2
12	7	0.9	4	109	6.316E+06	3.555E+08	1.56298E-15	0	150	5
12	7	0.9	4	121	8.533E+06	6.429E+08	1.50055E-15	0	184	4
12	7	0.9	4	107	5.658E+06	3.504E+08	1.02936E-15	0	141	3
12	7	0.9	4	133	9.138E+06	6.489E+08	2.05003E-15	0	180	4
12	7	0.9	4	125	8.267E+06	5.150E+08	1.90377E-15	0	168	5
12	7	0.9	4	97	4.496E+06	2.821E+08	1.73222E-15	0	125	1
12	7	0.9	4	118	8.697E+06	8.484E+08	1.47345E-15	0	183	5
12	7	0.9	4	137	1.127E+07	1.214E+09	1.83969E-15	0	205	4
12	7	0.9	4	160	1.528E+07	1.520E+09	1.63051E-15	0	227	6
12	7	0.9	4	110	6.645E+06	4.389E+08	3.80671E-14	0	161	2
12	7	0.9	4	99	4.432E+06	1.806E+08	1.11181E-15	0	119	2
12	7	0.9	4	95	4.644E+06	2.713E+08	1.55829E-15	0	131	2
12	7	0.9	4	134	8.635E+06	5.913E+08	2.98804E-15	0	164	3
12	7	0.9	4	140	1.230E+07	1.111E+09	2.15893E-15	0	224	9
24	7	0.9	4	566	3.793E+08	2.329E+11	3.15956E-15	0	1000	66
21	7	0.9	4	707	9.600E+08	8.082E+11	3.86588E-15	0	1766	168

9. Extensões ao Trabalho Apresentado

Apresentamos as nossas propostas para o que seria a evolução natural deste trabalho.

- 1- Incluir limites superiores e inferiores nos critérios de entrada(otimização em uma “caixa”), a fim de melhorar o desempenho do *solver*;
- 2- Estudar empiricamente o comportamento de várias heurísticas para critério de entrada.

3- Desenvolver um *solver* comercial baseado no trabalho apresentado. Este produto inclui a construção de um banco de dados para armazenar os dados primários do problema, implementação de rotinas de pré-processamento desta informação para que então o problema seja resolvido e interfaces para visualização dos dados e dos resultados. Segue uma descrição mais detalhada do projeto:

- Definição do esquema relacional do BD
- Implementação do BD e adaptação das atuais funções de entrada de dados.
- Funções de interface com o *solver* por arquivos texto.
- Funções para escalamento automático.
- Funções para detecção de *outliers*.
- Funções de interface de alto desempenho com o *solver*.
- Funções de visualização de dados.
- Sistema de auxílio a modelagem.
- Sistema de *help* do *software*.

4- Experimentar outras aplicações (problemas com estrutura semelhante nas áreas industrial e financeira), por exemplo em finanças, onde na análise de cenários econômicos para alocação de recursos ocorre o problema de programação estocástica descrito na página 18, e

5- Paralelizar este *solver* É importante notar, que na implementamos este *solver* seqüencial conservando a independência existente entre os blocos estendidos, já prevendo sua paralelização. Acreditamos que isso permita a reutilização de grande parte código escrito em diferentes esquemas de distribuição de memória e tarefas.

Os itens 1, 2 e 3 deverão ser desenvolvidos para incorporar o protótipo do *solver* desenvolvido projeto aos produtos comerciais da UniSoma. Os itens 1, 2, 4 e 5 serão objeto de pesquisa futura, como parte do programa de Doutorado do Engenheiro Fábio Nakano.

10. Bibliografia Básica

[Stern1] - Stern, J. M., Vavasis, S. A. - Active set methods for problems in column block angular form Comp. Appl. Math., V. 12 nº3, pp. 199-226, 1993

[Stern2] - Stern, J. M. et. al – Métodos Matemáticos em Otimização em Finanças XIX Congresso Nacional de Matemática Aplicada e Computacional, 1996

[Stern3] - Stern, J. M. - Esparsidade, Estrutura, Estabilidade e Escalonamento em Álgebra Linear Computacional. IX Escola de Computação - 1994

[Golub1] - Golub, G.H., Van Loan, Charles F. - Matrix Computations - pp. 592-597 Johns Hopkins, 1984

[Nazareth1] - Nazareth, J. L. - Computer Solutions of Linear Programming - pp. 20-26 Oxford Science, 1987

[Lasdon1] - Lasdon, Leon S. - Optimization Theory for Large Systems - pp. 115-121, MacMillan, 1970

[Rosen1] - J.B.Rosen, Supercomputers in Large-Scale Optimization, Editor, Balzer AG, Basel, 1990

[Amaral1] - L.R. de Souza Amaral, J. M. Martínez Pérez. Um Algoritmo Estável para Resolução do Problema de Otimização de Rações. IMECC-UNICAMP, 1987.

[Gomes1] - F.A.M.Gomes, A.Fridlander, O Método Simplex com Decomposição LU Aplicado a Problemas Bloco-Angulares. Pesquisa Operacional, V-12 pp.63-74, 1994.

11. Bibliografia de Referência

[Aho74] - A.Aho, J.Hopcroft, J.Ullman, - The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading Massachusetts, 1974.

[Bartels69] - R.H.Bartels, G.H.Golub, - The Simplex Method of Linear Programming Using LU Decomposition.
Comm. ACM, 12, 266-268, 1969.

[Bastian81] - M.Bastian - Aspects of Basis Factorization for Block Angular Systems with Coupling Rows. - In [Dantzig81]

[Bazaraa79] - M.S.Bazaraa, C.M.Shetty - Nonlinear Programming.
John Wiley, Chichester, 1979..

[Bennett65] - J.M.Bennett - Triangular Factors of Modified Matrices.
Numerische Mathematik, 7, 217-221, 1965.

[Bennett66] - J.M.Bennett - An Approach to Some Structured Linear Programming Problems.
Operations Research, 14(4) 636-645, 1966.

[Bertsekas89] - D.P.Bertsekas, J.N.Tsitsiklis - Parallel and Distributed Computation: Numerical Methods
Prentice Hall, Englewood Cliffs, 1989.

[Bisschop77] J.Bisschop, A.Meeraus - Matrix Augmentation and Partitioning in the Updating of the Basis Inverse.
Mathematical Programming, 13, 241-254, 1977.

[Bodewig56] E.Bodewig - Matrix Calculus.
North-Holland, Amsterdam., 1956.

[Buckley82] A.G.Buckley J.L.Goffin - Algorithms for Constrained Minimization of Smooth Nonlinear Functions.
Mathematical Programming Study 16. North Holland, Amsterdam, 1982.

- [Bunch76] - J.R.Bunch, D.J.Rose - Sparse Matrix Computations.
Academic Press, New York, 1976.
- [Carey89] - G.F.Carey - Parallel Supercomputing.
John Wiley, Chichester, 1989.
- [Carpenter90] - T.J.Carpenter, I.J.Lustig, J.M.Mulvey, D.F.Shanno - A primal-Dual Interior Point Method for Convex Separable Nonlinear Programs.
Technical Report SOR 90-2, Department of Civil Engineering and Operations Research, Princeton University, Princeton, 1990.
- [Coleman87] - T.F.Coleman, A.Pothen - The Null Space Problem II. Algorithms.
SIAM J. Alg. Disc. Meth. 8, 544-563, 1987.
- [Coleman88] - T.F.Coleman, C.van Loan - Handbook for Matrix Computations.
SIAM, Philadelphia, 1988.
- [Coleman90] - T.F.Coleman, Y.Li - Large Scale Numerical Optimization.
SIAM, Philadelphia, 1990.
- [Dantzig63] G.B.Dantzig - Linear Programming and Extensions.
Princeton University Press, Princeton, 1963.
- [Dantzig81] - B.Dantzig, M.A.H.Dempster, M.J.Kallio, Editors - Large--Scale Linear Programming.
IIASA Collaborative Proceedings Series CP-81-S1, Laxenburg, Austria, 1981.
- [Dantzig90] - B.Dantzig, W.Glynn - Parallel Processors for Planning Under Uncertainty.
In [Rosen90].
- [Dongarra91] - J.J.Dongarra, L.S.Duff, D.C.Sorensen, H.A.'van der Vorst - Solving Linear Systems on Vector and Shared Memory Computers}.
SIAM, Philadelphia, 1991.
- [Duff78] - I.S.Duff, G.W.Stewart - Sparse Matrix Proceedings.
SIAM, Philadelphia, 1978.
- [Duff86] I.S.Duff - Direct methods for sparse matrices.
Clarendon press, Oxford 1986.
- [Erisman85] - A.M.Erisman, R.G.Grimes, J.G.Lewis, W.G.Poole - A Structurally Stable Modification of Hellerman--Rarick's P4 Algorithm for Reordering Unsymmetric Sparse Matrices.
SIAM J. Numer. Anal. 22(2), 369-385, 1985.
- [Ermoliev87] - E.Ermoliev, R.J.B.Wets - Numerical Procedures for Stochastic Optimization.
Springer Verlag, Berlin, 1987.
- [Fiduccia82] - C.M.Fiduccia, R.M.Mattheyses - A Linear Time Heuristic for Improving Network Partitions.
IEEE Design Automation Conferences 19, 175-181, 1982.

- [Forrest92] - J.Forrest, J.Tomlin - Implementing the Simplex Method for the Optimization Subroutine Library
IBM Systems Journal, vol 31 pp 11-25, 1992.
- [Forrest92] - J.Forrest, J.Tomlin - Implementing Interior Point Methods in the Optimization Subroutine Library
IBM Systems Journal, vol 31 pp 26-38, 1992.
- [Forsythe67] - G.E.Forsythe, C.M.Moler - Computer Solution of Linear Algebraic Systems.
Prentice Hall, Englewood Cliffs, 1967.
- [George90] - A.George, M.Heath, J.Liu, E.Ng - Solution of Sparse Positive Definite Systems on a Hypercube.
In [Vorst90].
- [Gilbert87] - J.Gilbert, M.Heath - Computing a Sparse Basis for the Null Space.
SIAM J. Alg. Disc. Meth. 8, 446-459, 1987.
- [Gill74] - P.E.Gill, G.H.Golub, W.Murray, M.Saunders - Methods for Modifying Matrix Factorizations.
Mathematics of Computations, 28 (126), 505-535, 1974.
- [Gill75] - P.E.Gill, G.H.Golub, W.Murray, M.Saunders - Methods for Computing and Modifying LDV Factorizations of a Matrix.
Math. of Comp. 29, 1051-1077, 1975.
- [Glover89] - F.Glover - Candidate List Strategies and Tabu Search.
Preprint, Graduate School of Business, University of Colorado, Boulder, 1989.
- [Glover90] - F.Glover - Simple Tabu Thresholding.
Preprint, Graduate School of Business, University of Colorado, Boulder, 1990.
- [Golub83] - G.E.Golub, C.F.van Loan - Matrix Computations.
Jons Hopkins University Press, Baltimore, 1983.
- [Gustavson76] - F.Gustavson - Finding the Block Lower Triangular Form of a Sparse Matrix.
In [Bunch76].
- [Harary72] - F.Harary - Graph Theory.
Addison-Wesley, 1972.
- [Hellerman71a] - E.Hellerman and D.C.Rarick - Reinversion with the Preassigned Pivot Procedure.
Mathematical Programming 1, 195-216 1971.
- [Hellerman71b] - E.Hellerman and D.C.Rarick - The Partitioned Preassigned Pivot Procedure}.
In [Rose71].
- [Howell76] - T.D.Howell - Partitioning Using PAQ.
In [Bunch76].

- [Kernighan70] - B.W.Kernighan, S.Lin - An Efficient Heuristic Procedure for Partitioning Graphs.
The Bell System Technical Journal 49, 291-307, 1970.
- [Lasdon70] - L.S.Lasdon - Optimization Theory for Large Systems}.
The Macmillan Company, New York, 1970.
- [Lawson74] - C.L.Lawson, R.J.Hanson - Solving Least Square Problems.
Prentice Hall, Englewood Cliffs, 1974.
- [Liu88] - J.W.H.Liu - Equivalent Sparse Matrix Reordering by Elimination Tree Rotations.
Siam J. Sci.Statist.Comput. 9, 424-444, 1988.
- [Luenberger84] - D.G.Luenberger - Linear and Nonlinear Programming.
Addison-Wesley, Reading Massachusetts, 1984.
- [Lustig87] - I.J.Lustig - An analysis of an available set of Linear Programming test problems.
Technical Report SOL-87-11, Department of Operations Research, Stanford University, 1987.
- [McCormick83] - G.P.McCormick - Nonlinear Programming: Theory, algorithms and applications.
John Wiley, Chichester, 1983.
- [Mulvey89] - J.M.Mulvey, H.Vladimirou - Stochastic Network Programming for Financial Planning Problems.
Report SOR-89-7, Department of Civil Engineering and Operations Research, Princeton University, Princeton, 1989.
- [Murray82] - W.Murray, M.H.Wright - Computation of Search Directions in Constrained Optimization Algorithms.
In [Buckley82].
- [Murtagh81] - B.A.Murtagh - Advanced Linear Programming.
McGraw-Hill, New York, 1981.
- [Murtagh82] - B.A.Murtagh, M.A.Saunders - A Projected Lagrangean Algorithm and its Implementations for Sparse Nonlinear Constraints.
In [Buckley82].
- [Nemhauser89] - G.L.Nemhauser, A.H.G.Rinnooy Kan, M.J.Todd – Optimization Handbooks in Operations Research and Management Science
North Holland, Amsterdam, 1989.
- [Orchard68] - W.Orchard-Hays - Advanced Linear-Programming Computing Techniques.
McGraw-Hill, New York, 1968.
- [Perold78] - A.F.Perold, G.B.Dantzig - A Basis Factorization Method for Block Triangular Linear Programs.
In [Duff78].
- [Pissanetzky84] - S.Pissanetzky - Sparse Matrix Technology.
Academic Press, London, 1984.

- [Plemmons90] - R.J.Plemmons, R.E.White - Substructuring Methods for Computing the Nullspace of Equilibrium Matrices.
SIAM Journal on Matrix Analysis and Applications 11, 1-22, 1990.
- [Pothen84] - A.Pothen - Sparse Null Bases and Marriage Problems.
Technical Report 85-676, Department of Computer Science, Cornell University, Ithaca, 1984.
- [Pothen84] - A.Pothen - Sparse Null Basis Computations in Structural Optimization.
Technical Report CS-88-27, Department of Computer Science, Pennsylvania State University, University Park, 1988.
- [Pothen90] - A.Pothen, C.Sun - Compact Clique Tree Data Structures in Sparse Matrix Factorizations.
In [Coleman90].
- [Prekopa80] - A.Prekopa - Studies on Mathematical Programming.
Akademiai Kiado, Budapest, 1980.
- [Rose71] - D.J.Rose, R.A.Willoughby, Editors - Sparse Matrices and Applications.
Plenum Press, New York, 1971.
- [Rosen90] - J.B.Rosen, Editor - Supercomputers and Large--Scale Optimization.
Baltzer AG, Basel, 1990.
- [Rosen90] - J.B.Rosen, R.S.Maier - Parallel Solution of Large--Scale Block Angular Linear Programs.
In [Rosen90].
- [Saunders72] - M.A.Saunders - Large Scale Linear Programming Using the Cholesky Factorization.
PhD thesis, Computer Science Department, Stanford University, 1972.
- [Saunders76] - M.A.Saunders - A Fast, Stable Implementation of the Simplex Method Using Bartels and Golub Updating.
In [Bunch76].
- [Sechen87]{sechen87}C. Sechen, K. Lee - An Improved Simulated Annealing Algorithm for Row-Based Placement.
Proceedings IEEE international conference on computer-aided design, 478-481, 1987.
- [Stern87] - J.M.Stern - Algoritmos Eficientes de Programação Linear.
Instituto de Matemática e Estatística da Universidade de São Paulo, São Paulo, 1987.
- [Stern90a] - J.M.Stern - Simulated Annealing with a Temperature Dependent Cost Function.
Technical Report CCOP-90-1, Cornell Computational Optimization Project. Cornell University, Ithaca, 1990.
- [Stern90b] - J.M. Stern, S.A. Vavasis - Nested Dissection for Sparse Nullspace Bases.
Technical Report TR-90-1173, Department of Computer Science, Cornell University, Ithaca, 1990.
- [Stern91a] - J.M.Stern - Simulated Annealing with a Temperature Dependent Penalty Function.

In [Boros91].

[Stern91b] - J.M. Stern, S.A. Vavasis - Active Set Algorithms for Problems in Block Angular Form.
Department of Computer Science, Cornell University, Ithaca, 1991.

[Stewart74] - G.W. Stewart - Modifying Pivot Elements in Gaussian Elimination}.
Mathematics of Computation, 28(126), 537-542, 1974.

[Strang88] - G. Strang - A Framework for Equilibrium Equations.
SIAM Review 30, 283-297, 1988.

[Tewarson73] - R.P. Tewarson - Sparse Matrices.
Academic Press, New York, 1973.

[Ullman84] - J. Ullman - Computational Aspects of VLSI.
Computer Science Press, Rockville, Maryland, 1984.

[Vorst90] - H.A. van der Vorst, P. van Dooren - Parallel Algorithms for Numerical Linear Algebra.
North Holland, Amsterdam, 1990.

[Weil71] - R.L. Weil, P.C. Kettler - Rearranging Matrices to Block Angular Form for Decomposition Algorithms.
Management Science, 18 (1), 98-108, 1971.

[Welsh76] - D. Welsh - Matroid Theory.
London Mathematical Society Monograph 8. Academic Press, New York, 1976.

[Wilkinson65] - J.H. Wilkinson - The Algebraic Eigenvalue Problem.
Clarendon Press, Oxford, 1965.

[Winkler80] - C. Winkler - Basis Factorization for Block Angular Linear Programs.
In [Prekopa80]

[Zoutendijk60] - G. Zoutendijk - Methods of Feasible Directions.
Elsevier, Amsterdam, 1960.

[Wirth89] - N. Wirth - Algoritmos e Estruturas de Dados.
Prentice Hall, 1989.

[Yourdon89] - E. Yourdon - Modern Structured Analysis.
Yourdon Press, 1989.

[Stroustrup91] - B. Stroustrup - The C++ Programming Language
Addison Wesley, 1991.

[Ellis95] - M. A. Ellis, B. Stroustrup - The Annotated C++ Manual
Addison Wesley, 1995.

[Spuler94] - D. A. Spuler - C++ and C Debugging, Testing, and Reliability
Prentice Hall, 1994.

- [Winston94] – P. H. Winston – On To C++
Addison Wesley, 1994.
- [Baker89] – L. Baker – C Tools for Scientists and Engineers
McGraw-Hill, 1989.
- [Baker91] – L. Baker – More C Tools for Scientists and Engineers
McGraw-Hill, 1991.
- [Ferraris93] – G. Buzzi-Ferraris – Scientific C++
Addison Wesley, 1993.
- [Dubois97] – P. F. Dubois – Object Technology for Scientific Computing
Prentice Hall, 1997.
- [Engeln96] – G. Engeln-Müllges, F. Uhlig - Numerical Algorithms with C
Springer, 1996.
- [Reverchon93] – A. Reverchon, M. Ducamp – Mathematical Software Tools in C++
Wiley, 1993.
- [Press92] – W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery– Numerical Recipes in C
Cambridge, 1992.
- [Palmer84] – J. F. Palmer, S. P. Morse - The 8087 Primer
Wiley, 1984.

Anexo I

Código Matlab4.0 do *solver* para
problemas na forma angular bloqueada

Anexo II

Código Matlab5.0 do *solver* para
problemas na forma angular bloqueada

Anexo III

Código Matlab5.0 do *solver* para
problemas na forma angular blocada
aninhada

Anexo IV

Código C do *solver* para problemas na
forma angular bloqueada

Anexo V

Código C do *solver* para problemas na
forma angular bloqueada aninhada

Parte I

Código Matlab4.0 do *solver* para
problemas na forma angular bloqueada

Código MATLAB

DRIVE.M

```
clear all
global V1 V2 V3 W1 W2 W3 S ...
      B1 B2 B3 C1 C2 C3 ...
      RD1 RD2 RD3 RC1 RC2 RC3 ...
      AD1 AD2 AD3 AC1 AC2 AC3 ...
      c1 c2 c3 cr cb ...
      bb bi rb ri c d ...
      sm sn sd kb pm pn;

teste
montabr (bb, bi, rb, ri);
sn=0; sm=0; pm=0; pn=0;
for k=1:kb
    kstr= int2str(k);
    eval (['[mk,nk]=size(B' kstr ');'])
    sm=sm+mk; sn=sn+nk;
    eval (['[mk,nk]=size(AD' kstr ');'])
    pm=pm+mk;
    pn=pn+nk;
end

pm=pm+size(AC1, 1);

sd=sm-sn;
sqrbafe
seu
```

SIMPLEX.M

```
%function [Vo,b,x,r] = simplex(criterio,trace)

global AD1 AD2 AD3 AC1 AC2 AC3;
global B1 B2 B3 C1 C2 C3;
global V1 V2 V3 W1 W2 W3 S;
global RD1 RD2 RD3 RC1 RC2 RC3;
global bb bi rb ri c d;
global cb cr;
global sm sn sd kb pm pn;

criterio=1;
trace=0;

    acabou = 0;
    while ~acabou
%       montabr (bb, bi, rb, ri);
%       sqrbafe;

        x=cb';
        etmx;
        x=y;
```

```

utix;
x=y;
uix;
cbtil=emx ('RD', 'RC', kb, y)';
z = cbtil - cr

x=d;
utix;
x=y;
uix;
dttil=emx ('B', 'C', kb, y);

if ( any(z > 0) )
    indices = find(z > 0);
    if criterio == 1
        J = indices(1);    % preguica
    else
        [aux,J] = max(z);    %maior declive
    end %if criterio

    jb=rb(1,J);
    ji=ri(1,J);
    [J, jb, ji]

    rj=zeros (sm, 1);          % montando coluna de R
    rn=0; rm=0;
    for k=1:jb-1
        kstr= int2str(k);
        eval (['mk,nk']=size(AD' kstr '));
        rm=rm+mk; rn=rn+nk;
    end
    kstr=int2str (jb);
    eval (['mk,nk']=size(AD' kstr '));
    eval (['rj(rn+1:rm+mk, 1)=AD' int2str(jb) '(:,ji);']);
    eval (['rj(sn+1:sm, 1)=AC' int2str(jb) '(:,ji);']);

    % coluna da matriz R que entra na base
    eval (['qe=AD' int2str(jb) '(:,ji)'''])
    eval (['qc=AC' int2str(jb) '(:,ji)'''])

    x=rj;
    utix;
    x=y;
    uix;
    x=emx ('B', 'C', kb, y);
    rtilj= x;

    k = find( rtilj > 1e-10 );

    epsilon = dttil(k) ./ rtilj(k)
    [aux,i] = min( epsilon );
    i = k(i);
    ib=bb(1, i);
    ii=bi(1, i);
    klinha=find (abs(rtilj)>1e-10);
    [dttil(klinha)./rtilj(klinha) bb(klinha)' bi(klinha)']
    [aux, i, ib, ii]

    k=find (bb==ib);
    ki=min(k);          % indice de bb onde o bloco ib comeca
    kf=max(k);          % indice de bb onde o bloco ib termina

    h=find (bi(1,ki:kf)==ii); % indice da linha que sai da base na base

    k=ib; cpmk2m; [B,C,V,W,S,n2q]=upo (B, C, V, W, S, h); cpm2mk; n2q
    k=jb; cpmk2m; [B,C,V,W,S]=upi (B, C, V, W, S, qe, qc); cpm2mk;

% sai de R

k=find (rb==jb);
ki=min(k);          % indice de bb onde o bloco 'ib' comeca
kf=max(k);          % indice de bb onde o bloco ib termina
h=find (ri(1,ki:kf)==ji) % indice da linha que entra na base, em R
kstr=int2str(jb);
eval (['RD' kstr '(h,:)=[];']);
eval (['RC' kstr '(h,:)=[];']);

```

```

% entra em R

kstr=int2str(ib);
eval (['RD' kstr '=[RD' kstr '; AD' kstr '(:,ii)'' ];']);
eval (['RC' kstr '=[RC' kstr '; AC' kstr '(:,ii)'' ];']);

% nao da para garantir que o j seja o indice em bb, bi da coluna
% que sai pois ha muitos scripts no meio. Mas se a variavel for J
% nao ha problema.

bb(i)={};
bi(i)={};
k=find (bb==jb);
k=min(k);
bm=size(cb, 2);

% auxbb=bb(i);
% bb(i)=bb(k);
% bb(k)=rb(J);
bb=[bb(1:k-1) rb(J) bb(k:bm-1)];

% auxbi=bi(i);
% bi(i)=bi(k);
% bi(k)=rb(J);
bi=[bi(1:k-1) ri(J) bi(k:bm-1)];

auxcb=cb(i);
% cb(i)=cb(k);
% cb(k)=cr(J);
cb(i)={};
cb=[cb(1:k-1) cr(J) cb(k:bm-1)];

rb(J) = [];
ri(J) = [];
crj=cr(1, J);
cr(J)={};

k=find (rb==ib);
k=max(k);
bkrb=rb;
bkri=ri;
bkcr=cr;
rm=size (rb, 2);
rb=[bkrb(1,1:k) ib bkrb(1,(k+1):rm)]
ri=[bkri(1,1:k) ii bkri(1,(k+1):rm)]
cr=[bkcr(1,1:k) auxcb bkcr(1,(k+1):rm)]

else % acabou
    acabou = 1;
x=zeros (pn, 1);
offset=0;
estado=1;
for i=1:size(bb, 2)
    if (estado~=bb(1, i))
        estr=int2str(estado);
        eval (['offset=offset+size(B' estr ',1)']);
        estado=estado+1;
    end
    ii=bi(1, i)+offset;
    x(ii, 1)=dti(i);
end
'valores otimos'
cb*dti
x
end %if indices

% na ultima base, que é a ótima, dti=B\ d=x(b)

if (trace)
    'Paused'
    pause
end
end %while

```

end

%*****

MONTABR.M

```
function dummy=montabr(bb, bi, rb, ri)
% function dummy=montabr(bb, bi, rb, ri)
```

```
global B1 B2 B3 C1 C2 C3 ...
        RD1 RD2 RD3 RC1 RC2 RC3 ...
        AD1 AD2 AD3 AC1 AC2 AC3 ...
        c1 c2 c3 cr cb;
```

```
B1=[]; B2=[]; B3=[];
C1=[]; C2=[]; C3=[];
cb=[];
```

```
n=size(bb,2);
for i=1:n
    i
    bstr= int2str(bb(1,i));
    istr= int2str(bi(1,i));
    eval(['B' bstr '= [B' bstr '; AD' bstr '(:, ' istr ')'];']);
    eval(['C' bstr '= [C' bstr '; AC' bstr '(:, ' istr ')'];']);
    eval(['cb(1,i)= c' bstr '(1, ' istr ')'];'])
end
```

```
RD1=[]; RD2=[]; RD3=[];
RC1=[]; RC2=[]; RC3=[];
cr=[];
```

```
n=size(rb,2);
for i=1:n
    bstr= int2str(rb(1,i));
    istr= int2str(ri(1,i));
    eval(['RD' bstr '= [RD' bstr '; AD' bstr '(:, ' istr ')'];']);
    eval(['RC' bstr '= [RC' bstr '; AC' bstr '(:, ' istr ')'];']);
    eval(['cr(1,i)= c' bstr '(1, ' istr ')'];'])
end
%*****
```

EMX.M

```
function y=emx (dstr, cstr, kn, x)
% x=emx (dstr, cstr, kn, x)
% produto de matriz blocada por vetor coluna
%
%      |dstr1 0 0 cstr1|
% y=   | 0 dstr2 0 cstr2|*x
%      | 0 0 dstr3 cstr3|
% dstr: nome dos blocos diagonais
% cstr: nome dos blocos de conexão
% kn:   quantidade de blocos (diagonais==conexão)
% x:    vetor coluna.
```

```
global B1 B2 B3 C1 C2 C3 sm sn sd ...
        RD1 RD2 RD3 RC1 RC2 RC3;
xbk=x;
xs(1:sd,1)=x(sn+1:sm,1);
x=x(1:sn,1);
```

```
y=[];
for k=1:kn
    %inpar
    kstr= int2str(k);
    eval(['B= ' dstr kstr ';']);
    eval(['C= ' cstr kstr ';']);
    [mk,nk]=size(B);
    nx=size(x,1);
    xk=x(1:nk,1);
    x=x(nk+1:nx,1);
    yk= B*xk +C*xs;
    y=[y;yk];
    %seqver
    %seqver
    %seqver
```

```
end
x=xbk;
```

```
%*****
```

ETMX.M

```
% etmx.m
```

```
global B1 B2 B3 C1 C2 C3
xbk=x;
```

```
y=[];
ys=zeros(sd,1);
for k=1:kb %inpar
    cpmk2m;
    [mk,nk]=size(B);
    nx=size(x,1); %seqver
    xk=x(1:mk,1); %seqver
    x=x(mk+1:nx,1); %seqver
    yk= B'*xk;
    ctxk= C'*xk;
    ys= ys+ctxk;
    y=[y;yk]; %seqver
end
y=[y;ys];
x=xbk;
```

```
%*****
```

UIX.M

```
% uix.m
```

```
xbk=x;
sd=sm-sn;
xs(1:sd,1)=x(sn+1:sm,1);
x=x(1:sn,1);
ys= zeros(sd,1);
ys(sd,1)= xs(sd,1)/S(sd,sd);
for i=sd-1:-1:1
    ys(i,1)=( xs(i,1) -S(i,i+1:sd)*ys(i+1:sd,1) )/S(i,i);
end
y=ys; %fanout-ys

for k=kb:-1:1 %in par
    cpmk2m;
    [nk,sd]=size(W);
    wys= W*ys;
    nx=size(x,1); %seqver
    xk=x(nx-nk+1:nx,1); %seqver
    x=x(1:nx-nk,1); %seqver
    yk=zeros(nk,1);
    yk(nk,1)=( xk(nk,1) -wys(nk,1) )/V(nk,nk);
    for i=nk-1:-1:1;
        yk(i,1)=( xk(i,1) -wys(i,1) -V(i,i+1:nk)*yk(i+1:nk,1) )/V(i,i);
    end
    y=[yk;y]; %seqver
end
x=xbk;
```

```
%*****
```

UTIX.M

```
% utix.m
```

```
xbk=x;
sd= sm-sn;
wty= zeros(sd,1);
```



```

y=[];
for k=1:kb %in par
    cpmk2m;
    [nk,sd]=size(W);
    nx=size(x,1); %sever
    xk=x(1:nk,1); %sever
    x=x(nk+1:nx,1); %sever
    yk=zeros(nk,1);
    yk(1,1)=xk(1,1)/V(1,1);
    for i=2:nk;
        yk(i,1)=( xk(i,1) -V(1:i-1,i)'*yk(1:i-1,1) )/V(i,i);
    end
    wtyk= W'*yk;
    y=[y;yk]; %sever
    wty= wty+wtyk; %sever
end

xs=x(1:sd,1);
ys= zeros(sd,1); %terminal
ys(1,1)= ( xs(1,1) -wty(1,1) )/S(1,1);
for i=2:sd
    ys(i,1)=( xs(i,1) -wty(i,1) -S(1:i-1,i)'*ys(1:i-1,1) )/S(i,i);
end
y=[y;ys];

x=xbk;

```

```

%*****

```

SQRBAF.M

```

%script sqrbaf.m

```

```

sm=0; sn=0;
for k=1:kb
    cpmk2m;
    [m,n]=size(B);
    sm=sm+m; sn=sn+n;
    nc=size(C,2);
    [V,W]=qrb(B,C);
    Zk=W(n+1:m,1:nc);
    W=W(1:n,1:nc);
    if(k==1) Z=Zk;
    else Z=[Z;Zk];
end%if
cpm2mk;
end
S= qrb(Z);

```

```

%*****

```

UPI.M

```

function [B,C,V,W,S]= upi(B,C,V,W,S,eb,ec)

```

```

% new column enters as the 1-th row of
% the k-th block of the basis transpose E'
[mk,nk]= size(B); sd= size(C,2);

```

```

% S is (sd-1)xsd
% Q'*E=R => diag(1,Q')*[e;E]=[r;R]

```

```

[mk,nk]= size(B);
sd= size(C,2);
B=[eb;B]; C=[ec;C];
mk=mk+1;
V=[eb;V]; W=[ec;W];

```

```

% UU= [V, W; zeros(sd-1,nk), S],
% UU=qrh(UU),

```

```

[V,W]= qrh(V,W);

nkp=nk+1;
V(nkp,:)=[];
wnkp= W(nkp,:);
W(nkp,:)=[];
S= [wnkp;S];
S= qrh(S);

```

```

%*****

```

UPO.M

```

function [B,C,V,W,S,n2q]= upo(B,C,V,W,S,h)

%permuting h-th and 1-st rows of k-th block
[mk,nk]= size(B); sd= size(C,2);
B= [B(h,:);B(1:h-1,:);B(h+1:mk,:)];
C= [C(h,:);C(1:h-1,:);C(h+1:mk,:)];

%B= [B(h,:);B(2:h-1,:);B(1,:);B(h+1:mk,:)];
%C= [C(h,:);C(2:h-1,:);C(1,:);C(h+1:mk,:)];

% 1-st column of k-th block leaves the basis E'
% Q*U=E => Q=E*inv(U) => Q'=inv(U')*E'
% inv(E)= inv(U)*Q'= inv(U)*inv(U')*E'
% inv(E')= E*inv(U)*inv(U')
% we have U=Q'*E
%      V1 0 0 W1      B1 0 0 C1
% U= 0 Vk 0 Wk = Q'* 0 Bk 0 Ck = Q'*E
%      0 0 Vb Wb      0 0 Bb Cb
%      0 0 0 S
% Bk is m(k) x n(k) , Vk is m(k) x m(k)
% sm= sum(m(k)) sn= sum(n(k)) sd=sm-sn
% Ck is m(k)xsd Wk is n(k)xsd S is sdxsd
% diag(1,Q')*[e;E]=[r;R] => Q'*E=R
% q= Q'(:,1) , Gsm'..G1'*q = P'*q = [(+-)1;0;...0]
% U=Q'E => NU=P'U=P'Q'E = diag(1,NQ)*E
% q= inv(U')*E'(:,1)=
% q= Q'(:,ei)= [0;...0;qb;0;...0;qc]
% UU= [V, W; zeros(sd,nk), S];
% ebc= [eb;ec]; qbc= vtim(UU,ebc);
% or [qb,qc]= uitet(V,W,S,eb,ec);
% reducing q to I(:,ei)
% [nqbc,NUU]= qrx(qbc,UU);

eb=B(1,:); ec=C(1,:);
B= B(2:mk,1:nk); C= C(2:mk,1:sd);

[qb,qc]= uitet(V,W,S,eb,ec);
[qc,S]= qrx(qc,S);
s1= S(1,:);
S= S(2:sd,1:sd);
x= [qb;qc(1,1)];
V= [V;zeros(1,nk)];
W= [W;s1];
[x,V,W]= qrx(x,V,W);
n2q= norm(eb'-x(1,1)*V(1,:)) ...
+norm(ec'-x(1,1)*W(1,:));
V=V(2:nk+1,1:nk);
W=W(2:nk+1,1:sd);

```

```

%*****

```

QRB.M

```

function [V,C] = qrb(B,C)

% QR factorization of B , V = Q'*B , W = Q'*C

```

```

if( nargin==1 ) C=[]; end%if
[m,n]= size(B);
nc= size(C,2);

for j=1:n
    for i=j+1:m
        [gc,gs]= givcs(B(j,j),B(i,j));
        for k=j:n
            [B(j,k),B(i,k)]=givrt(gc,gs,B(j,k),B(i,k));
        end%k
        for k=1:nc %for1:0=nothing
            [C(j,k),C(i,k)]=givrt(gc,gs,C(j,k),C(i,k));
        end%k
    end%i
end%j
V=B(1:n,1:n);

%*****

```

QRH.M

```

function [V,W] = qrh(H,W)

% QR factorization of H , Hessemberg
% V = Q'*H , NW = Q'*W

if( nargin==1 ) W=[]; end%if
[m,n]= size(H);
nw= size(W,2);

for j=1:n
    i=j+1;
    if( i<=m)
        [gc,gs]= givcs(H(j,j),H(i,j));
        for k=j:n
            [H(j,k),H(i,k)]= givrt(gc,gs,H(j,k),H(i,k));
        end%k
        for k=1:nw %for1:0=nothing
            [W(j,k),W(i,k)]= givrt(gc,gs,W(j,k),W(i,k));
        end%k
    end%ifi
end%j
V= triu(H(1:n,1:n));

%*****

```

QRX.M

```

function [x,H,W] = qrx(x,V,W)

% reduction of x to ||x||*I(:,1)
% V upper triang => Hess.

if( nargin==2 ) W=[]; end%if
[m,n]= size(V);
nw= size(W,2);
H=V;
for i=m-1:-1:1
    ip=i+1;
    [gc,gs]= givcs(x(i,1),x(ip,1));

% gc=-gc; gs=-gs;

    [x(i,1),x(ip,1)]= givrt(gc,gs,x(i,1),x(ip,1));
    for j=i:n
        [H(i,j),H(ip,j)]= givrt(gc,gs,H(i,j),H(ip,j));
    end%j
    for j=1:nw
        [W(i,j),W(ip,j)]= givrt(gc,gs,W(i,j),W(ip,j));
    end%j
end%j

```

```
end%i
```

```
% *****
```

VIMR.M

```
function y = vimr(V,x)
```

```
% y=inv(V)*x  V upper-triangular
```

```
[m,n]=size(V);
```

```
y=zeros(m,1);
```

```
y(m,1) = x(m,1)/V(m,m);
```

```
for i=m-1:-1:1
```

```
    y(i,1)=( x(i,1) - V(i,i+1:m)*y(i+1:m,1) )/V(i,i);
```

```
end
```

```
% *****
```

VTIMR.M

```
function y = vtimr(V,x)
```

```
% y=inv(V')*x  V upper-triangular
```

```
[m,n]=size(V);
```

```
y=x;
```

```
for i=1:m-1
```

```
    y(i,1)= y(i,1)/V(i,i);
```

```
    y(i+1:m,1)= y(i+1:m,1) -y(i,1)*(V(i,i+1:m)');
```

```
end
```

```
y(m,1)= y(m,1)/V(m,m);
```

```
% *****
```

Parte II

Código Matlab5.0 do *solver* para
problemas na forma angular bloqueada

```

function y = vimr(V,x)

% y=inv(V)*x  V upper-triangular

[m,n]=size(V);

y=zeros(m,1);
y(m,1) = x(m,1)/V(m,m);
for i=m-1:-1:1
    y(i,1)=( x(i,1) - V(i,i+1:m)*y(i+1:m,1) )/V(i,i);
end
%*****

% uixd.m

sgena;

yy=rand(sm,1);
x=U*yy;
uix;
norm(y-yy),

% *****

function y=utix(U, x)
% inv(U')*x, U upper triangular
% U is square and its last diagonal block is empty.
kb=size(U,1);
sd= size(U(kb,2),2);

wty= zeros(sd,1);
y=[];
for k=1:kb-1          %in par
    nk=size(U(k,2),1);
    nx=size(x,1);      %seqver
    xk=x(1:nk,1);      %seqver
    x=x(nk+1:nx,1);    %seqver
    yk=zeros(nk,1);
    yk(1,1)=xk(1,1)/U(k,1)(1,1);
    for i=2:nk;
        yk(i,1)=( xk(i,1) -U(k,1)(1:i-1,i)'*yk(1:i-1,1) )/U(k,1)(i,i);
    end
    wtyk= U(k,2)'*yk;
    y=[y;yk];          %seqver
    wty= wty+wtyk;     %seqver
end

xs=x(1:sd,1);
ys= zeros(sd,1);      %terminal
ys(1,1)= ( xs(1,1) -wty(1,1) )/U(kb,2)(1,1);
for i=2:sd
    ys(i,1)=( xs(i,1) -wty(i,1) -U(kb,2)(1:i-1,i)'*ys(1:i-1,1) )/U(kb,2)(i,i);
end
y=[y;ys];

%*****

function y=emx (E, x)
% x=emx (E, x)
% produto de matriz blocada por vetor coluna
% y=E*x
% E: matriz rbaf
kn=size(E,1);
sd=size(E(kn,2),2);
sm=size(x,1);
sn=sm-sd;
xs(1:sd,1)=x(sn+1:sm,1);
x=x(1:sn,1);

y=[];
for k=1:kn          %inpar

```

```

[mk,nk]=size(E{k,1});
nx=size(x,1); %seqver
xk=x(1:nk,1); %seqver
x=x(nk+1:nx,1); %seqver
yk= E{k,1}*xk +E{k,2}*xs;
y=[y;yk]; %seqver
end

%*****

function [x,H,W] = qrx(x,V,W)

% reduction of x to ||x||*I(:,1)
% V upper triang => Hess.

if( nargin==2 ) W=[]; end%if
[m,n]= size(V);
nw= size(W,2);
H=V;
for i=m-1:-1:1
    ip=i+1;
    [gc,gs]= gives(x(i,1),x(ip,1));

% gc=-gc; gs=-gs;

    [x(i,1),x(ip,1)]= givrt(gc,gs,x(i,1),x(ip,1));
    for j=i:n
        [H(i,j),H(ip,j)]= givrt(gc,gs,H(i,j),H(ip,j));
    end%j
    for j=1:nw
        [W(i,j),W(ip,j)]= givrt(gc,gs,W(i,j),W(ip,j));
    end%j
end%i

%*****

function [w1,w2]= givrt(v1,v2,gc,gs)

w1= gc*v1 -gs*v2;
w2= gs*v1 +gc*v2;

%*****

function [D, C, E] = qrb(D, C, E)

% QR factorization of B , V = Q'*B , W = Q'*C

if (nargin == 1) C=[]; end%if
if (nargin ~= 3) E=[]; end
[m,n]= size(D);
nc= size(C,2);
ne= size(E,2);

for j=1:n
    for i=j+1:m
        [gc,gs]= gives(D(j,j),D(i,j));
        for k=j:n
            [D(j,k),D(i,k)]=givrt(gc,gs,D(j,k),D(i,k));
        end%k
        for k=1:nc %for1:0=nothing
            [C(j,k),C(i,k)]=givrt(gc,gs,C(j,k),C(i,k));
        end%k
        for k=1:ne %for1:0=nothing
            [E(j,k),E(i,k)]=givrt(gc,gs,E(j,k),E(i,k));
        end%k
    end%i
end%j
D=D(1:n,1:n);

```

```
%*****
```

```
function [V,W] = qrh(H,W)
```

```
% QR factorization of H , Hessemberg
```

```
% V = Q'*H , NW = Q'*W
```

```
if( nargin==1 ) W=[]; end%if
```

```
[m,n]= size(H);
```

```
nw= size(W,2);
```

```
for j=1:n
```

```
    i=j+1;
```

```
    if( i<=m)
```

```
        [gc,gs]= givcs(H(j,j),H(i,j));
```

```
        for k=j:n
```

```
            [H(j,k),H(i,k)]= givrt(gc,gs,H(j,k),H(i,k));
```

```
        end%k
```

```
        for k=1:nw %for1:0=nothing
```

```
            [W(j,k),W(i,k)]= givrt(gc,gs,W(j,k),W(i,k));
```

```
        end%k
```

```
    end%ifi
```

```
end%j
```

```
V= triu(H(1:n,1:n));
```

```
%*****
```

```
"
```

```
"
```

```
function [gc,gs] = givcs(v1,v2)
```

```
% computes G | G'*[v1,v2]'=[norm(2,v),0]'
```

```
% G = [gcos, gsen; -gsen, gcos]
```

```
EPSLN= 1E-10;
```

```
if( abs(v2)<EPSLN )
```

```
    gcos=1; gsen=0;
```

```
else
```

```
    if( abs(v1)>=abs(v2) )
```

```
        tau= -v2/v1;
```

```
        gcos= (1+tau*tau)^(-1/2);
```

```
        gsen= gcos*tau;
```

```
    else % abs(v2)>abs(v1)
```

```
        tau= -v1/v2;
```

```
        gsen= (1+tau*tau)^(-1/2);
```

```
        gcos= gsen*tau;
```

```
    end %else_abs
```

```
end %else_v20
```

```
%G = [gcos, gsen; -gsen, gcos];
```

```
gc=gcos; gs=gsen;
```

```
%*****
```

```
function sparspy(E)
```

```
EPS=1E-9;
```

```
[m,n]= size(E);
```

```
for i=1:m
```

```
    for j=1:n
```

```
        if( abs(E(i,j))<EPS )
```

```
            E(i,j)=0;
```

```
        end%if
```

```
    end%j
```

```
end%i
```

```
spy(E)
```

```
%*****
```

```
%sqrbd.m
```



```

m=7; n=4;
B=rand(m,n); B=triu(B,-1);
C=eye(m);
[V,W]=qrh(B,C), %qrh
norm(V-W*B),

%*****

function y = vimc(V,x)

% y=inv(V)*x  V upper-triangular

[m,n]=size(V);

y=x;
for i=m:-1:2
    y(i,1)= y(i,1)/V(i,i);
    y(1:i-1,1)= y(1:i-1,1) -y(i,1)*V(1:i-1,i);
end
y(1,1)= y(1,1)/V(1,1);

%*****

function y = vimr(V,x)

% y=inv(V)*x  V upper-triangular

[m,n]=size(V);

y=zeros(m,1);
y(m,1) = x(m,1)/V(m,m);
for i=m-1:-1:1
    y(i,1)=( x(i,1) - V(i,i+1:m)*y(i+1:m,1) )/V(i,i);
end

%*****

function y = vtimc(V,x)

% y=inv(V')*x  V upper-triangular

[m,n]=size(V);

y=zeros(m,1);
y(1,1) = x(1,1)/V(1,1);
for i=2:m
    y(i,1)=( x(i,1) - V(1:i-1,i)'*y(1:i-1,1) )/V(i,i);
end

%*****

function y = vtimr(V,x)

% y=inv(V')*x  V upper-triangular

[m,n]=size(V);

y=x;
for i=1:m-1
    y(i,1)= y(i,1)/V(i,i);
    y(i+1:m,1)= y(i+1:m,1) -y(i,1)*(V(i,i+1:m)');
end
y(m,1)= y(m,1)/V(m,m);

%*****

function y = vtimr(V,x)

% y=inv(V')*x  V upper-triangular

```

```

[m,n]=size(V);

y=x;
for i=1:m-1
    y(i,1)= y(i,1)/V(i,i);
    y(i+1:m,1)= y(i+1:m,1) -y(i,1)*(V(i,i+1:m)');
end
y(m,1)= y(m,1)/V(m,m);

%*****

function y=uix (U, x)
% y=inv(U)*x; U is rbafe, upper triangular

kb=size(U,1);
sm=size(x,1);
sd=size(U(kb,2),2);
sn=sm-sd;
xs(1:sd,1)=x(sn+1:sm,1);
x=x(1:sn,1);
ys= zeros(sd,1);
ys(sd,1)= xs(sd,1)/U(kb,2)(sd,sd);
for i=sd-1:-1:1
    ys(i,1)=( xs(i,1) -U(kb,2)(i,i+1:sd)*ys(i+1:sd,1) )/U(kb,2)(i,i);
end
y=ys;          %fanout-ys

for k=kb-1:-1:1 %in par
    [nk,sd]=size(U(k,2));
    wys= U(k,2)*ys;
    nx=size(x,1);          %seqlver
    xk=x(nx-nk+1:nx,1);    %seqlver
    x=x(1:nx-nk,1);        %seqlver
    yk=zeros(nk,1);
    yk(nk,1)=( xk(nk,1) -wys(nk,1) )/U(k,1)(nk,nk);
    for i=nk-1:-1:1;
        yk(i,1)=( xk(i,1) -wys(i,1) -U(k,1)(i,i+1:nk)*yk(i+1:nk,1) )/U(k,1)(i,i);
    end
    y=[yk;y];              %seqlver
end

%*****
% utixd.m

sgena;

yy=rand(sm,1);
x=U'*yy;
utix;
norm(y-yy),

% *****

function [qb,qc]=uitet(V,W,S,eb,ec);

[nk,sd]= size(W);

% Q'=inv(U')*E'  q=inv(U')*[eb;ec]
% UU=[V, W; zeros(sd,nk), S]; ebc=[eb;ec];
% qbc= vtim(UU,ebc); %qbc',

qb=zeros(nk,1);
qb(1,1)= eb(1,1)/V(1,1);
for i=2:nk
    qb(i,1)=( eb(i,1) - V(1:i-1,i)'*qb(1:i-1,1) )/V(i,i);
end
wtqb(1:sd,1)= W(1:nk,1:sd)'*qb(1:nk,1);

qc=zeros(sd,1);
qc(1,1)=( ec(1,1) -wtqb(1,1) )/S(1,1);
for i=2:sd

```

```

        qc(i,1)=( ec(i,1) -wtqb(i,1) ...
                  -S(1:i-1,i)'*qc(1:i-1,1) )/S(i,i);
    end

%*****

function y=etmx(E, x)
% y=etmx(E, x)
% y=E'*x,
% E: matriz rbaf
kb=size(E,1);
sd=size(E(kb,2),2);

y=[];
ys=zeros(sd,1);
for k=1:kb %inpar
    [mk,nk]=size(E{k,1});
    nx=size(x,1); %seqver
    xk=x(1:mk,1); %seqver
    x=x(mk+1:nx,1); %seqver
    yk= E{k,1}'*xk;
    ctxk= E{k,2}'*xk;
    ys= ys+ctxk;
    y=[y;yk]; %seqver
end
y=[y;ys];

function U=sqrbafe (B)
% U=sqrbafe (B)
% fatoracao QR da matriz RBAF B
% U é triangular superior.

siz=size(B);
kb=siz(1,1);
U=cell(kb+1,2);
sm=0; sn=0;
for k=1:kb
    [m,n]=size(B{k,1});
    sm=sm+m; sn=sn+n;
    nc=size(B{k,2},2);
    [U{k,1},W]=qrb(B{k,1},B{k,2});
    Zk=W(n+1:m,1:nc);
    U{k,2}=W(1:n,1:nc);
    if(k==1) Z=Zk;
    else Z=[Z;Zk];
    end%if
end
U{kb+1,2}= qrb(Z);

%*****

function MDensa = r2d (Mrbaf)
% converte uma matriz na forma angular blocada em
% matriz densa.
[mb, nb]= size (Mrbaf);
mm=0; mn=0;
for i=1:mb
    [m1, n1]= size (Mrbaf{i,1});
    [m2, n2]= size (Mrbaf{i,2});
    mm=mm+m2; % o bloco diagonal pode ser vazio.
    mn=m1+n1;
end
cn=mn; % ultima coluna dos blocos diagonais como todo.
mn=mn+n2;
MDensa=zeros (mm, mn);
tempm=0; tempn=0;
for i=1:mb
    [m1, n1]= size (Mrbaf{i,1});
    [m2, n2]= size (Mrbaf{i,2});
    % m2 pois o bloco diagonal pode ser vazio.

```

```

%               vv
MDensa(1+tempm:tempm+m2,1+tempn:tempn+n1)=Mrbaf(i,1);
MDensa(1+tempm:tempm+m2,1+cn:mn)=Mrbaf(i,2);
tempm=tempm+m1;           % o bloco diagonal pode ser vazio.
tempn=tempn+n1;
end
spy (N);function y=par(x)
% teste para passagem de argumentos:
% verifica se a passagem e' feita por valor ou por referencia.
x(1,1)=x(1,1)/2;
y=1;

function [B, cb]=monta(A, c, b)
% function [B,cb]=monta(A, c, b)
% monta a partir do vetor de indices, b,
% a matriz rbaf, B e o vetor cb.
% note que montar R e cr segue o mesmo processo.

kb=size(A,1);
B=cell(kb,2);
offset=zeros(kb+1,1);
for i=2:kb+1
    offset(i, 1)=offset(i-1, 1)+size(A{i-1,2},2);
end

n=size(b,1);
cb=zeros (1,n);
for i=1:n
    B(b(i, 1),1)=[B(b(i, 1),1);A{b(i, 1),1}(:,b(i, 2))'];
    B(b(i, 1),2)=[B(b(i, 1),2);A{b(i, 1),2}(:,b(i, 2))'];
    cb(1,i)=c(1,offset(b(i, 1))+b(i, 2));
end
%*****

function U= upi(U, k, qa, qc)

kb=size(U,1);
U{k,1}=[qa; U{k,1}];
U{k,2}=[qc; U{k,2}];
[U{k,1}, U{k,2}]=qrh(U{k,1}, U{k,2});
m=size(U{k,1},1);
U{kb,2}=[U{k,2}(m+1,:);U{kb,2}];
U{k,2}=U{k,2}(1:m,:);
U{kb,2}=qrh(U{kb,2});

function [U,n2q]= upo(U, h, ba, bc)
% 1-st column of k-th block leaves the basis E'
% Q*U=E => Q=E*inv(U) => Q'=inv(U')*E'
% inv(E)= inv(U)*Q'= inv(U)*inv(U')*E'
% inv(E')= E*inv(U)*inv(U')
% we have U=Q'*E
%      V1 0 0 W1          B1 0 0 C1
% U=  0 Vk 0 Wk  =  Q'*  0 Bk 0 Ck  = Q'*E
%      0 0 Vb Wb          0 0 Bb Cb
%      0 0 0  S
% Bk is m(k) x n(k) , Vk is m(k) x m(k)
% sm= sum(m(k)) sn= sum(n(k)) sd=sm-sn
% Ck is m(k)xsd Wk is n(k)xsd S is sdxs
% diag(1,Q')*[e;E]=[r;R] => Q'*E=R
% q= Q'(:,1) , Gsm'..G1'*q = P'*q = [(+-)1;0;...0]
% U=Q'E => NU=P'U=P'Q'E = diag(1,NQ)*E
% q= inv(U')*E'(:,1)=
% q= Q'(:,ei)= [0;...0;q;0;...0;q]
% UU= [V, W; zeros(sd,nk), S];
% ebc= [eb;ec]; qbc= vtim(UU,ebc);
% or [qb,qc]= uitet(V,W,S,eb,ec);
% reducing q to I(:,ei)
% [nqbc,NUU]= qrx(qbc,UU);

kb=size(U,1);
nk=size(U{h,1},2);

```

```

sd=size(U{kb,2}, 1);
[qb,qc]= uitet(U{h,1},U{h,2},U{kb,2},ba',bc');
[qc,U{kb,2}]= qrx(qc,U{kb,2});
s1= U{kb,2}(1,:);
U{kb,2}= U{kb,2}(2:sd,1:sd);
x= [qb;qc(1,1)];
U{h,1}= [U{h,1};zeros(1,nk)];
U{h,2}= [U{h,2};s1];
[x,U{h,1},U{h,2}]= qrx(x,U{h,1},U{h,2});
n2q= norm(ba-x(1,1)*U{h,1}(1,:)) ...
      +norm(bc-x(1,1)*U{h,2}(1,:));
U{h,1}=U{h,1}(2:nk+1,1:nk);
U{h,2}=U{h,2}(2:nk+1,1:sd);

%*****

function B=bpo (B, k, i)
% bpo (B, k, i)
% removes the i-th row from the k-th block of B.
% B is a RBAF matrix.

B(k,1)(i,:)=[];
B(k,2)(i,:)=[];
function B=bpi (B, k, ba, bc)
% inserts [ba..bc] as the first row of the k-th
% block of B.
B(k,1)=[ba; B(k,1)];
B(k,2)=[bc; B(k,2)];

function err=ver (B, U)
% calculates norm(inv(U')*B'*B*inv(U)-I)
% that must be VERY small.

BD=r2d(B);
UD=r2d(U);
err=norm(inv(UD')*BD'*BD*inv(UD)-eye(size(BD, 1)));

function indice=blocpos (b, i)
% Given a row b(i) of b, index vector,
% returns the position of the element inside
% its block for instance,
% blockpos (b, 7) with
% b'=[ 1 1 1 1 2 2 2 2 3 3 3 3      <- blocks
%      5 2 1 4 2 6 4 3 9 8 4 7 ];   <- indices in A
%      ^ this is the 7-th row in b,
%      3-rd element in the 2-nd block.
% returns 3

k=min(find(b(:,1)==b(i, 1)));
indice=i-k+1;function [indice, b]=updatel(b, bb, bi, j)
% Removes the j-th row of b (index vector) and inserts
% [bb, bi] as the first row of the bb-th block of b.
% returns the position of the inserted row in b.

b(j,:)=[];
n=size(b, 1);
indice=min(find(b(:,1)==bb));
b=[b(1:indice-1,:); [bb, bi]; b(indice:n,:)];

function [indice, b]=updaten(b, bb, bi, j)
% Removes the j-th row of b (index vector) and inserts
% [bb, bi] as the last row of the bb-th block of b.
% returns the position of the inserted row in b.

b(j,:)=[];
n=size(b, 1);
indice=max(find(b(:,1)==bb));
indice=indice+1;
b=[b(1:indice-1,:); [bb, bi]; b(indice:n,:)];

```

```

function c=update(c, vc, indice, j)
% Removes the j-th column of c (ordinary vector) and
% inserts vc in indice row of c.

    c(:,j)=[];
    n=size (c, 2);
    c=[c(1, 1:indice-1), vc, c(1, indice:n)];

function [jd, jc, id, ic]=swap (B, ib, ii, R, jb, ji)
% stores a copy of B{ib,:}(ii,:) in [id, ic] and
% removes them from B.
% stores a copy of R{jb,:}(ji,:) in [jd, jc] and
% removes them from R.
% inserts [jd, jc] as the 1-st row of B{jb,:}
% inserts [id, ic] as the 1-st row of R{ib,:}

    id=B{ib, 1}(ii,:); B{ib, 1}(ii,:)=[];
    ic=B{ib, 2}(ii,:); B{ib, 2}(ii,:)=[];
    jd=R{jb, 1}(ji,:); R{jb, 1}(ji,:)=[];
    jc=R{jb, 2}(ji,:); R{jb, 2}(ji,:)=[];
    B{jb, 1}= [B{jb, 1}; jd];
    B{jb, 2}= [B{jb, 2}; jc];
    R{ib, 1}= [id; R{ib, 1}];
    R{ib, 2}= [ic; R{ib, 2}];

function [bo, Vo, o]=simplex (A, d, c, b, r)
% function [bo, Vo, o]=simplex (A, d, c, b, r)
% Calculates o=max (cx) | Ax=d
% stops if it finds an optimal solution or
% the the polyhedral set is unbounded or
% took more than npassos and did not find
% the optimal solution.
% b0 is the initial basis
% bo is the optimal basis
% Vo is the optimal basic variables values and
% o is the optimal objective function value.
% A is ROW angular blocked, but we factorize
% the basis transpose. To do so, B=A(:,b) and so does R. , , ,
% monta transposes the basis.

    npassos=10;
    ipassos=0;
    [B, cb]=monta (A, c, b);
    [R, cr]=monta (A, c, r);
    U=sqrbafe(B);
    while (npassos>=ipassos)
        ipassos=ipassos+1;
        % [B, cb]=monta (A, c, b);
        % [R, cr]=monta (A, c, r);
        % U=sqrbafe(B);
        cbtil=emx(R, uix(U, utix(U, etmx(B, cb'))));
        z=cbtil-cr
        dtil=emx(B, uix(U, utix(U, d)));
        [aux, j]=max(z);
        if (aux<1e-12)
            'optimal solution'
            Vo=dtil;
            o= cb*dtil;
            bo=b;
            return
        end
        jb=r(j,1);
        ji=r(j,2);
        vcr=cr(1,j);
        rj=coluna (A, jb, ji);
        rtilj=emx(B, uix(U, utix(U, rj)));
        k=find (rtilj > 1e-12);
        epsilon=dtil(k)./rtilj(k);
        [epsilon, b(k,:)]

```

```

% testado OK para primeira rodada.
if (isempty (k))
    'unbounded problem'
    Vo=[];
    bo=[];
    o=[];
    return
end
[aux, i]= min (epsilon);
i=k(i);
ib=b(i, 1);
ii=b(i, 2);
vcb=cb(i);
% this is swap
sii=blockpos (b, i);
sji=blockpos (r, j);
id=B{ib, 1}(sii,:); B{ib, 1}(sii,:)=[];
ic=B{ib, 2}(sii,:); B{ib, 2}(sii,:)=[];
jd=R{jb, 1}(sji,:); R{jb, 1}(sji,:)=[];
jc=R{jb, 2}(sji,:); R{jb, 2}(sji,:)=[];
B{jb, 1}= [jd; B{jb, 1}];
B{jb, 2}= [jc; B{jb, 2}];
R{ib, 1}= [R{ib, 1}; id];
R{ib, 2}= [R{ib, 2}; ic];
%%%%%%%%%%
[indice, b]= update1(b, jb, ji, i);
cb= update (cb, vcr, indice, i);
[indice, r]= updaten (r, ib, ii, j);
cr= update (cr, vcb, indice, j);
[U, n2q]=upo (U, ib, id, ic);
U=upi (U, jb, jd, jc);
%     n2q, [b, cb'], [r, cr']
%     BD=r2d(B);
%     UD=r2d(U);
%     (inv(UD)*inv(UD')*BD'*BD)-eye(15)
end
'npassos reached'
Vo=[];
bo=[];
o=[];

function At=transp (A)
% function At=transpose (A)
% At=A', A and At are RBAF

[m, n]=size(A);
for i=1:m
    for j=1:n
        At(i,j)=A(i,j)';
    end
end

function v=coluna(A, jb, ji)
% function v=coluna(A, jb, ji)
% assembles v, dense array from A, rbafe matrix.
% v=A{jb,1}(:,ji); zeros (...) ;A{jb,2}(:,ji)

kb=size(A,1);

offset=zeros(kb+2,1);
for i=2:kb+1
    offset(i, 1)=offset(i-1, 1)+size(A{i-1,1},1);
end
offset (kb+2, 1)=offset(kb+1, 1)+size (A{1,2}, 1);
v=zeros (offset (kb+1,1), 1);
v(offset(jb)+1:offset(jb+1),1)=A{jb,1}(:,ji);
v(offset(kb+1)+1:offset(kb+2),1)=A{jb,2}(:,ji);
%*****

```

Parte III

Código Matlab5.0 do *solver* para
problemas na forma angular bloqueada
aninhada


```

function [w1,w2]= givrt(v1,v2,gc,gs)

w1= gc*v1 -gs*v2;
w2= gs*v1 +gc*v2;

%*****

function [gc,gs] = givcs(v1,v2)

% computes G | G'*[v1,v2]'=[norm(2,v),0]'
% G = [gcos, gsen; -gsen, gcos]

EPSLN= 1E-10;
if( abs(v2)<EPSLN )
    gcos=1; gsen=0;
else
    if( abs(v1)>=abs(v2) )
        tau= -v2/v1;
        gcos= (1+tau*tau)^(-1/2);
        gsen= gcos*tau;
    else % abs(v2)>abs(v1)
        tau= -v1/v2;
        gsen= (1+tau*tau)^(-1/2);
        gcos= gsen*tau;
    end %else_abs
end %else_v20
%G = [gcos, gsen; -gsen, gcos];
gc=gcos; gs=gsen;

%*****

function D=rdensa (m, n, s)
% function D=rdensa (m, n, s)
% builds D, dense, (m x n) non-negative elements, with sparsity s
D=rand(m,n);
pat=(D-s)>=0;
D=pat.*rand(m,n);

%*****

function y = vimr(V,x)

% y=inv(V)*x  V upper-triangular

[m,n]=size(V);
if m==0 y=[]; return;
end
y=zeros(m,1);
y(m,1) = x(m,1)/V(m,m);
for i=m-1:-1:1
    y(i,1)=( x(i,1) - V(i,i+1:m)*y(i+1:m,1) )/V(i,i);
end

%*****

function [D, C, E] = qrb(D, C, E)

% QR factorization of B , V = Q'*B , W = Q'*C , X=Q'*E

if(nargin == 1 ) C=[]; end%if
if(nargin ~= 3) E=[]; end
[m,n]= size(D);
if ((m==0)|(n==0))
    % a concatenacao de uma matriz vazia (0 x 0) e uma matriz vazia (0 x n)
    % e' uma matriz vazia (0 x n), o que gera um erro quando fazemos D(1:n,1:n);
    return
end

```

```

nc= size(C,2);
ne= size(E,2);

for j=1:n
    for i=j+1:m
        [gc,gs]= givcs(D(j,j),D(i,j));
        for k=j:n
            [D(j,k),D(i,k)]=givrt(gc,gs,D(j,k),D(i,k));
        end%k
        for k=1:nc %for1:0=nothing
            [C(j,k),C(i,k)]=givrt(gc,gs,C(j,k),C(i,k));
        end%k
        for k=1:ne %for1:0=nothing
            [E(j,k),E(i,k)]=givrt(gc,gs,E(j,k),E(i,k));
        end%k
    end%i
end%j
D=D(1:n,1:n);
D=triu(D); % FN set.98
%*****

function U=sqrnrhaf (B)
% U=sqrnrhaf (B)
% fatoracao QR da matriz NRBAF B
% U é NRBAF triangular superior.

nexternal=size (B, 1);
U=cells (nexternal+1, 1);
U(nexternal+1, 1)=cells (1, 3);

for i=1:nexternal
    ninternal=size (B(i,1), 1);
    U(i, 1)=cell (ninternal+1, 3);
    for k=1:ninternal
        n= size (B(i,1){k,1}, 2);
        nc= size (B(i,1){k,2}, 2);
        [m, ne]= size (B(i,1){k,3});
        [U(i,1){k,1}, U(i,1){k,2}, U(i,1){k,3}]=qrb(B(i,1){k,1}, B(i,1){k,2}, B(i,1){k,3});
        if (~isempty (U(i,1){k,2}))
            % this will not be necessary in C
            % because concatenate and select rows/columns of
            % empty matrices will not cause an error and stop
            % the program.
            U(i,1){ninternal+1, 2}=[U(i,1){ninternal+1, 2}; U(i,1){k, 2}(n+1:m, :)];
            U(i,1){k, 2}= U(i,1){k,2}(1:n, :);
        end

        if (~isempty (U(i,1){k,3}))
            % this will not be necessary in C
            % because concatenate and select rows/columns of
            % empty matrices will not cause an error and stop
            % the program.
            U(i,1){ninternal+1, 3}=[U(i,1){ninternal+1, 3}; U(i,1){k, 3}(n+1:m, :)];
            U(i,1){k, 3}= U(i,1){k,3}(1:n, :);
        end
    end
    nc=size (U(i,1){ninternal+1, 2}, 2);
    [m, ne]=size (U(i, 1){ninternal+1, 3});
    [U(i,1){ninternal+1,2}, U(i,1){ninternal+1,3}]=qrb(U(i,1){ninternal+1,2}, U(i,1){ninternal+1,3});

    if (~isempty (U(i,1){ninternal+1,3}))
        % this will not be necessary in C
        % because concatenate and select rows/columns of
        % empty matrices will not cause an error and stop
        % the program.
        U(nexternal+1,1){1, 3}=[U(nexternal+1,1){1, 3}; U(i,1){ninternal+1, 3}(nc+1:m, :)];
        U(i,1){ninternal+1, 3}= U(i,1){ninternal+1,3}(1:nc, :);
    end
end
end
U(nexternal+1, 1){1,3}=qrb(U(nexternal+1, 1){1, 3});

```

```

%*****
function NR=rnrbaf (m, s)
% function NR=rnrbaf (m, s)
% builds NR, square, nrbaif with m diagonal external blocks
% each diagonal external block has a random positive number
% of diagonal internal blocks. Internal diagonal (mid x nid)
% blocks have 0<=(mid-nid)<=3. Internal coupling (mic x, nic)
% blocks have 0<mic<=3. External diagonal blocks (med x ned)
% are more likely to have (med > ned) - not equal, never
% smaller. External coupling (mec x nec) blocks have nec
% adjusted to present a square matrix.

minternal=floor(4*rand(m, 1)+1);
nid=floor(3*rand(sum(minternal), 1)+1);
d=floor(3*rand(sum(minternal), 1));
mid=nid+d;
k=1; dnic=0; nic=zeros(m, 1);
for i=1:m
    dmde=sum(d(k:k+minternal(i,1)-1, 1));
    r=0;
    while r<0.7
        % at least 70% of the difference must be internal coupling
        r=rand(1,1);
    end
    nic(i, 1)=floor (r*dmde);
    dnic= dnic+dmde-nic(i,1);
    k=k+minternal(i,1);
end
NR=cells(m, 1); k=1;
for i=1:m
    NR(i,1)=cells(minternal(i,1), 3);
    for j=k: k+minternal(i,1)-1
        NR(i,1){j-k+1, 1}=rdensa (mid(j), nid(j), s);
        NR(i,1){j-k+1, 2}=rdensa (mid(j), nic(i), s);
        NR(i,1){j-k+1, 3}=rdensa (mid(j), dnic, s);
    end
    k=k+minternal(i,1);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function y=uiux (U, x)
% y=inv(U)*x; U is nrbaif, upper triangular

nexternal= size (U, 1);
mx=size(x,1); % U and x MUST have compatible sizes.
[m,n]=size(U{nexternal, 1}{1, 3});
xce=x(mx-m+1:mx,1); % pass sub-array pointer to the function
xce=vim(U{nexternal, 1}{1, 3}, xce);
x(mx-m+1:mx, 1)=xce;
mx=mx-m;
for i=nexternal-1:-1:1
    ninternal=size(U{i,1},1);
    [m,n]=size(U{i,1}{ninternal, 2});
    xci=x(mx-m+1:mx, 1);
    if (size(xce,1)>0) xci=xci-U{i,1}{ninternal,3}*xce;
    end
    xci=vim(U{i,1}{ninternal,2}, xci);
    x(mx-m+1:mx,1)=xci;
    mx=mx-m;
    for j=ninternal-1:-1:1
        [m,n]=size(U{i,1}{j,1});
        xdi=x(mx-m+1:mx,1);
        if (size(xci,1)>0) xdi=xdi-U{i,1}{j,2}*xci;
        end
        if (size(xce,1)>0) xdi=xdi-U{i,1}{j,3}*xce;
        end
        xdi=vim(U{i,1}{j,1}, xdi);
        x(mx-m+1:mx,1)=xdi;
        mx=mx-m;
    end
end
y=x;

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function D=nr2d (NR)
% Converts NR, nrbafe in D dense.
mNR=size (NR, 1);
mdi=[];
ndi=[];
nci=[];
for i=1:mNR
    b=size(NR{i,1}, 1);
    for j=1:b
        mdi=[mdi;size(NR{i,1}{j,3}, 1)];
        ndi=[ndi;size(NR{i,1}{j,1}, 2)];
    end
    nci=[nci; size(NR{i,1}{1,2}, 2)];
end
nce=size (NR{1,1}{1,3}, 2);
m=sum(mdi);
n=sum(ndi)+sum(nci)+nce;
D=zeros(m,n);
m0=1;
n0=1;
k=1;
for i=1:mNR
    b=size(NR{i,1}, 1);
    pnc=n0+sum(ndi(k:k+b-1,1));
    for j=1:b
        if (~isempty(NR{i,1}{j,1}))
%           'di'
            D(m0:m0+mdi(k, 1)-1, n0:n0+ndi(k,1)-1)=NR{i,1}{j,1};
        end
        if (~isempty(NR{i,1}{j,2}))
%           'ci'
%           [i, j]
%           [mdi(k,1), size(NR{i,1}{j,2},1)]
%           [nci(i,1), size(NR{i,1}{j,2},2)]
            D(m0:m0+mdi(k, 1)-1, pnc:pnc+nci(i,1)-1)=NR{i,1}{j,2};
        end
        if (~isempty(NR{i,1}{j,3}))
%           'ce'
            D(m0:m0+mdi(k, 1)-1, n-nce+1:n)=NR{i,1}{j,3};
        end
        m0=m0+mdi(k, 1);
        n0=n0+ndi(k, 1);
        k=k+1;
    end
    n0=n0+nci(i,1);
end
% script to generate and spy NRBAF matrices

NR=rnrbafe(3, 0);
D=nr2d(NR);
spy (D);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function sparspy(E)
% function sparspy(E)
% plots the fill pattern of E, elements <EPS=1E-9
% are considered zeros.
EPS=1E-9;
[m,n]= size(E);
for i=1:m
    for j=1:n
        if( abs(E(i,j))<EPS )
            E(i,j)=0;
        end%if
    end%j
end%i
spy(E)

%*****

```

```

function y=utix (U, x)
% U'y=x

nexternal=size (U, 1); mx0=1; de=zeros(size(U{nexternal, 1}{1,3}, 2),1);
for i=1:nexternal-1
    ninternal=size (U{i,1}, 1); di=zeros(size(U{i, 1}{ninternal, 2}, 2),1);
    for j=1:ninternal-1
        [m,n]=size (U{i,1}{j,1});
        xdi=x(mx0:mx0+n-1, 1);
        xdi=vtim(U{i,1}{j,1}, xdi);
        x(mx0:mx0+n-1,1)=xdi;
        if (~isempty(U{i,1}{j,2}))
            di=di+U{i,1}{j,2}'*xdi;
        end
        if (~isempty(U{i,1}{j,3}))
            de=de+U{i,1}{j,3}'*xdi;
        end

        mx0=mx0+n; % o tamanho amarra o processamento
    end
    [m,n]=size (U{i,1}{ninternal,2});
    xci=x(mx0:mx0+n-1,1);
    xci=xci-di;
    xci=vtim(U{i,1}{ninternal,2}, xci);
    if (~isempty(U{i,1}{ninternal,3}))
        de=de+U{i,1}{ninternal,3}'*xci;
        x(mx0:mx0+n-1,1)=xci;
    end
    mx0=mx0+n;
end
[m,n]=size (U{nexternal,1}{1,3});
xce=x(mx0:mx0+n-1, 1);
xce=xce-de;
xce=vtim(U{nexternal,1}{1,3}, xce);
x(mx0:mx0+n-1,1)=xce;
y=x;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function y = vtim(V,x)

% y=inv(V')*x  V upper-triangular

[m,n]=size(V);
if (m==0) y=zeros(size(x, 1),1); return; % this is
%necessary due to the error handling of MATLAB.
%if no error handling were done, everything would
%be OK... this indicates good ways of doing it in C.
end
y=x;
for i=1:m-1
    y(i,1)= y(i,1)/V(i,i);
    y(i+1:m,1)= y(i+1:m,1) -y(i,1)*(V(i,i+1:m)');
end
y(m,1)= y(m,1)/V(m,m);

```

```

%*****

function y=emx(B, x)
% function y=emx(B, x)
% y=B*x, B nrbaf.
nexternal=size(B,1);
n=size(x,1);
nce=size(B(nexternal, 1){1,3},2);
nci=n-nce;
ndi=n-nce;
xce=x(n-nce+1:n,1);
y=[];
for i=nexternal:-1:1
    ninternal=size(B{i,1},1);
    nc=size(B{i,1}(ninternal,2),2);
    xci=x(nci-nc+1:nci,1);
    ndi=ndi-nc;
    for j=ninternal:-1:1
        [md, nd]=size (B{i,1}(j,1));
        yi=zeros(md, 1);
        xdi=x(ndi-nd+1:ndi,1);
        if (~isempty(B{i,1}(j,1))) yi=yi+B{i,1}(j,1)*xdi; end;
        if (~isempty(B{i,1}(j,2))) yi=yi+B{i,1}(j,2)*xc; end;
        if (~isempty(B{i,1}(j,3))) yi=yi+B{i,1}(j,3)*xce; end;
        y=[yi;y];
        ndi=ndi-nd;
    end
    nci=ndi;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [D, C, E] = qrh(D, C, E)
% function [D, C, E] = qrh(D, C, E)
% QR factorization of B Hessenberg, V = Q'*B , W = Q'*C , X=Q'*E

if(nargin == 1 ) C=[]; end%if
if(nargin ~= 3) E=[]; end
[m,n]= size(D);
nc= size(C,2);
ne= size(E,2);
for i=1:m-1
    [gc,gs]= givcs(D(i,i),D(i+1,i));
    for k=i:n
        [D(i,k),D(i+1,k)]=givrt(gc,gs,D(i,k),D(i+1,k));
    end%k
    for k=1:nc %for1:0=nothing
        [C(i,k),C(i+1,k)]=givrt(gc,gs,C(i,k),C(i+1,k));
    end%k
    for k=1:ne %for1:0=nothing
        [E(i,k),E(i+1,k)]=givrt(gc,gs,E(i,k),E(i+1,k));
    end%k
end%i
D=D(1:n,1:n);
D=triu (D); % FN set.98
%*****

```

```

function [U,n2q]= upo(U, he, hi, bdi, bci, bce)
% function [U,n2q]= upo(U, he, hi, bdi, bci, bce)

nexternal=size(U,1);
ninternal=size(U{he,1},1);
m=size(U{nexternal,1}{1,3}, 1);
nce=size(U{he,1}{hi,3},2);
nci=size(U{he,1}{hi,2},2);
ndi=size(U{he,1}{hi,1},2);

[qdi,qci,qce]= uitet(U, he, hi, bdi, bci, bce);
[qce,U{nexternal,1}{1,3}]= qrx(qce,U{nexternal,1}{1,3});
sce= U{nexternal,1}{1,3}(1,:);
U{nexternal,1}{1,3}= U{nexternal,1}{1,3}(2:m,:);
qci= [qci;qce(1,1)];
U{he,1}{ninternal,3}= [U{he,1}{ninternal,3};sce];
U{he,1}{ninternal,2}= [U{he,1}{ninternal,2};zeros(1,nci)];
[qci,U{he,1}{ninternal,2},U{he,1}{ninternal,3}]= ...
    qrx(qci,U{he,1}{ninternal,2},U{he,1}{ninternal,3});

m=size(U{he,1}{ninternal,3},1);
sce= U{he,1}{ninternal,3}(1,:);
U{he,1}{ninternal,3}= U{he,1}{ninternal,3}(2:m,:);

m=size(U{he,1}{ninternal,2},1);
sci= U{he,1}{ninternal,2}(1,:);
U{he,1}{ninternal,2}= U{he,1}{ninternal,2}(2:m,:);

U{he,1}{hi,3}= [U{he,1}{hi,3};sce];
U{he,1}{hi,2}= [U{he,1}{hi,2};sci];
U{he,1}{hi,1}= [U{he,1}{hi,1};zeros(1,ndi)];
qdi= [qdi;qci(1,1)];
[qdi,U{he,1}{hi,1}, U{he,1}{hi,2},U{he,1}{hi,3}]= ...
    qrx(qdi,U{he,1}{hi,1}, U{he,1}{hi,2},U{he,1}{hi,3});

n2q=0;
if (~isempty (bdi))
    n2q= n2q+norm(bdi-qdi(1,1)*U{he,1}{hi,1}(1,:));
end
if (~isempty (bci))
    n2q= n2q+norm(bci-qdi(1,1)*U{he,1}{hi,2}(1,:));
end
if (~isempty (bce))
    n2q= n2q+norm(bce-qdi(1,1)*U{he,1}{hi,3}(1,:));
end

if (~isempty(U{he,1}{hi,1}))
    U{he,1}{hi,1}=U{he,1}{hi,1}(2:ndi+1,:);
end
if (~isempty(U{he,1}{hi,2}))
    U{he,1}{hi,2}=U{he,1}{hi,2}(2:ndi+1,:);
end
if (~isempty(U{he,1}{hi,3}))
    U{he,1}{hi,3}=U{he,1}{hi,3}(2:ndi+1,:);
end
%*****

```

```

function U= upi(U, ke, ki, qdi, qci, qce)
% function U= upi(U, ke, ki, qdi, qci, qce)
% Inserts [qdi,...,qci,...,qce] as the first line in
% the ki-th internal block of the ke-th external block
% of U (it's now upper Hessenberg) then restores its form
% (upper triangular) by QR-factorization.

nexternal=size(U,1);
ninternal=size(U{ke,1},1);
U{ke,1}{ki,1}=[qdi; U{ke,1}{ki,1}];
U{ke,1}{ki,2}=[qci; U{ke,1}{ki,2}];
U{ke,1}{ki,3}=[qce; U{ke,1}{ki,3}];

[U{ke,1}{ki,1}, U{ke,1}{ki,2}, U{ke,1}{ki,3}]=...
qrh(U{ke,1}{ki,1}, U{ke,1}{ki,2}, U{ke,1}{ki,3});

m=size(U{ke,1}{ki,3},1);

if (~isempty (U{ke,1}{ki,2}))
    U{ke,1}{ninternal,2}=[U{ke,1}{ki,2}(m,:);U{ke,1}{ninternal,2}];
    U{ke,1}{ki,2}=U{ke,1}{ki,2}(1:m-1,:);
end

if (~isempty (U{ke,1}{ki,3}))
    U{ke,1}{ninternal,3}=[U{ke,1}{ki,3}(m,:);U{ke,1}{ninternal,3}];
    U{ke,1}{ki,3}=U{ke,1}{ki,3}(1:m-1,:);
end

[U{ke,1}{ninternal,2},U{ke,1}{ninternal,3}]=...
qrh(U{ke,1}{ninternal,2},U{ke,1}{ninternal,3});

m=size(U{ke,1}{ninternal,3},1);

if (~isempty (U{ke,1}{ninternal,3}))
    U{nexternal,1}{1,3}=[U{ke,1}{ninternal,3}(m,:);U{nexternal,1}{1,3}];
    U{ke,1}{ninternal,3}=U{ke,1}{ninternal,3}(1:m-1,:);
end

U{nexternal,1}{1,3}=qrh(U{nexternal,1}{1,3});

```



```

function nrspy(S,EPS);
%NRSPY Visualize sparsity pattern of NRBAF matrices.
% NRSPY(S) plots the sparsity pattern of the matrix S.
%
% NRSPY(S,EPS) plots the sparsity pattern of the matrix S.
% If the absolute value of an element in S is less than EPS it's
% not plotted.

hold off;
clf;
marker = '.'; color = 'blue'; linestyle = 'none'; markersize = 0;
if isempty(marker), marker = '.'; end

[m,n]=nrsize(S);

if marker~='.' & markersize==0,
    markersize = get(gcf,'defaultlinemarkersize');
end
if markersize == 0
    units = get(gca,'units');
    set(gca,'units','points');
    pos = get(gca,'position');
    markersize = max(4,min(14,round(6*min(pos(3:4))/max(m+1,n+1)))));
    set(gca,'units',units);
end

set(gca,'xlim',[0 n+1],'ylim',[0 m+1],'ydir','reverse', ...
    'grid','none','plotboxaspectratio',[n+1 m+1 1]);
set(gcf,'renderer','painters')

Xh=[1;n+1];
Yv=[1;m+1];

hold on
m0=0; n0=0; m00=m0;
nexternal = size(S,1);
for k=1:nexternal
    marker = '.';
    ninternal= size(S{k,1},1);
    m0=m00;
    for l=1:ninternal
        if (nargin==2) [i,j] = find(abs(S{k,1}{l,1})>EPS);
        else
            [i,j] = find(S{k,1}{l,1});
        end

        if isempty(i)
            i = NaN; j = NaN;
        end
        if isempty(S{k,1}{l,1})
            marker = 'none';
        end

        plot(n0+j,m0+i,'marker',marker,'markersize',markersize, ...
            'linestyle',linestyle,'color',color);
        d=max([size(S{k,1}{l,1});size(S{k,1}{l,2});size(S{k,1}{l,3})]);
        d(1,2)=size(S{k,1}{l,1},2);
        m0=m0+d(1,1);
        n0=n0+d(1,2);
        Xv=[n0+1;n0+1;];
        Yh=[m0+1;m0+1;];
        plot (Xh,Yh, ':g');
        plot (Xv,Yv, ':g');
    end
    marker = '.';
    m0=m00;
    for l=1:ninternal
        if (nargin==2) [i,j] = find(abs(S{k,1}{l,2})>EPS);
        else
            [i,j] = find(S{k,1}{l,2});
        end
        if isempty(i)

```

```

        i = NaN; j = NaN;
    end
    if isempty(S{k,1}{1,2})
        marker = 'none';
    end
        plot(n0+j,m0+i,'marker',marker,'markersize',markersize, ...
            'linestyle',linestyle,'color',color);
    d=max([size(S{k,1}{1,1});size(S{k,1}{1,2});size(S{k,1}{1,3})]);
    d(1,2)=size(S{k,1}{1,2},2);
    m0=m0+d(1,1);
end
m00=m0;
n0=n0+d(1,2);
Xv=[n0+1;n0+1;];
plot (Xv,Yv, '-m');
Yh=[m00+1;m00+1];
plot (Xh,Yh, '-m');
end

m0=0;
for k=1:nexternal
    marker = '.';
    ninternal= size(S{k,1},1);
    for l=1:ninternal
        if (nargin==2)
            [i,j] = find(abs(S{k,1}{1,3})>EPS);
        else
            [i,j] = find(S{k,1}{1,3});
        end
        if isempty(i)
            i = NaN; j = NaN;
        end
        if isempty(S{k,1}{1,3})
            marker = 'none';
        end
        plot(n0+j,m0+i,'marker',marker,'markersize',markersize, ...
            'linestyle',linestyle,'color',color);
    d=max([size(S{k,1}{1,1});size(S{k,1}{1,2});size(S{k,1}{1,3})]);
    m0=m0+d(1,1);
end
end

hold off
function [m,n]=nrsize (NR)
% Returns m, n, size of NR
mNR=size(NR,1);
mdi=[];

ndi=[];
nci=[];

for i=1:mNR
    b=size(NR{i,1}, 1);
    for j=1:b
        mdi=[mdi;max([size(NR{i,1}{j,1}, 1);size(NR{i,1}{j,2}, 1);size(NR{i,1}{j,3}, 1)]]);
        ndi=[ndi;size(NR{i,1}{j,1}, 2)];
    end
    nci=[nci; size(NR{i,1}{1,2}, 2)];
end
nce=size (NR{1,1}{1,3}, 2);
m=sum(mdi);
n=sum(ndi)+sum(nci)+nce;
%*****
function [qdi, qci, qce]=uitet(U, ke, ki, bdi, bci, bce);
nexternal=size(U,1);
ninternal=size(U{ke, 1},1);
bdi=bdi'; bci=bci'; bce=bce';
qdi=vtim(U{ke,1}{ki,1}, bdi);
if (~isempty(U{ke,1}{ki,2}))
    bci=bci-U{ke,1}{ki,2}.*qdi;
end

```

```

qci=vtim(U{ke,1}{ninternal,2},bci);
if (~isempty (U{ke,1}{ki,3}))
    bce=bce-(U{ke,1}{ki,3}'*qdi);
end
if (~isempty (U{ke,1}{ninternal,3}))
    bce=bce-(U{ke,1}{ninternal,3}'*qci);
end
qce=vtim(U{nexternal,1}{1,3},bce);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [x, D, C, E] = qrx(x, D, C, E)
% function [x, V, W, X] = qrx(x, D, C, E)
% QR factorization of x, vector, V = Q'*B , W = Q'*C , X=Q'*E
if(nargin == 2 ) C=[]; end%if
if(nargin ~= 4) E=[]; end
m= size(x,1);
nd= size(D,2);
nc= size(C,2);
ne= size(E,2);
for i=m-1:-1:1
    [gc,gs]= givcs(x(i,1),x(i+1,1));
    [x(i,1),x(i+1,1)]=givrt(gc,gs,x(i,1),x(i+1,1));
    for k=1:nd
        [D(i,k),D(i+1,k)]=givrt(gc,gs,D(i,k),D(i+1,k));
    end%k
    for k=1:nc %for1:0=nothing
        [C(i,k),C(i+1,k)]=givrt(gc,gs,C(i,k),C(i+1,k));
    end%k
    for k=1:ne %for1:0=nothing
        [E(i,k),E(i+1,k)]=givrt(gc,gs,E(i,k),E(i+1,k));
    end%k
end%i

%*****

function B=bpo (B, ke, ki, i)
% bpo (B, ke, ki, i)
% removes the i-th row from the ki-th internal block
% of the ke-th externalblock of B.
% B is a NRBAF matrix.

B{ke,1}{ki,1}(i,:)=[];
B{ke,1}{ki,2}(i,:)=[];
B{ke,1}{ki,3}(i,:)=[];
function B=bpi (B, ke, ki, bd, bc, be)
% function B=bpi (B, ke, ki, bd, bc, be)
% inserts [bd..bc..be] as the first row of the ki-th
% internal block of the ke-th external block of B, nrbafe.
B{ke,1}{ki,1}=[bd; B{ke,1}{ki,1}];
B{ke,1}{ki,2}=[bc; B{ke,1}{ki,2}];
B{ke,1}{ki,3}=[be; B{ke,1}{ki,3}];

```

```

function indice=blockpos (b, i)
% Given a row b(i) of b, index vector,
% returns the position of the element inside
% its internal block for instance,
% blockpos (b, 7) with
% b'=[ 1 1 2 2 2 2 2 2 2 2 2 2      <- external blocks
%      1 1 1 1 2 2 2 2 3 3 3 3      <- internal blocks
%      5 2 1 4 2 6 4 3 9 8 4 7 ];    <- indices in A
%                                     ^ this is the 7-th row in b,
%                                     3-rd element in the 2-nd internal block.
% returns 3

n=size(b,1);
ke=min(find(b(:,1)==b(i, 1)));
ki=min(find(b(ke:n,2)==b(i, 2)));
indice=i-(ke+ki-1)+1;function [indice, b]=updatel(b, be, bi, br, j)
% function [indice, b]=updatel(b, be, bi, br, j)
% Removes the j-th row of b (index vector) and inserts
% [be, bi, br] as the first row of the bi-th internal block
% of the be-th internal block of b.
% returns the position of the inserted row in b.

b(j,:)=[];
n=size(b, 1);
iei=size(find(b(:,1)<be))+1;
ief=n-size(find(b(:,1)>be));
iii=size(find(b(iei:ief,2)<bi))+1;
indice=iei+iii-1;
b=[b(1:indice-1,:); [be, bi, br]; b(indice:n,:)];

function [indice, b]=updaten(b, be, bi, br, j)
% function [indice, b]=updaten(b, be, bi, br, j)
% Removes the j-th row of b (index vector) and inserts
% [be, bi, br] as the last row of the bi-th internal block of
% the be-th external block b.
% returns the position of the inserted row in b.

b(j,:)=[];
n=size(b, 1);
iei=size(find(b(:,1)<be))+1;
ief=n-size(find(b(:,1)>be));
iif=size(find(b(iei:ief,2)>bi));
indice=ief-iif+1;
b=[b(1:indice-1,:); [be, bi, br]; b(indice:n,:)];

```

```

function [B, cb]=monta(A, c, b)
% function [B,cb]=monta(A, c, b)
% monta a partir do vetor de indices, b,
% a matriz NRBAF, B e o vetor cb.
% note que montar R e cr segue o mesmo processo.

ne=size(A,1);
B=cell(ne,1);
ccb=cell(ne,1);
offset=cell(ne,1);
coff=0;
for i=1:ne
    ni=size(A{i,1},1);
    offset{i,1}=zeros(ni+1,1);
    offset{i,1}(1,1)=coff;
    B{i,1}=cell(ni,3);
    ccb{i,1}=cells(ni,1);
    for j=2:ni+1
        % warning: B is the tranpose of the selection of columns in A
        % swap dimensions.
        % Initializing matrices with a (0 x n) empty one is necessary
        % to keep the NRBAF matrix size correct.
        ndi=size(A{i,1}{j-1,1},1);
        B{i,1}{j-1,1}=zeros(0,ndi);
        nci=size(A{i,1}{j-1,2},1);
        B{i,1}{j-1,2}=zeros(0,nci);
        nce=size(A{i,1}{j-1,3},1);
        B{i,1}{j-1,3}=zeros(0,nce);
        offset{i,1}(j,1)=offset{i,1}(j-1,1)+size(A{i,1}{j-1,3},2);
    end
    coff=offset{i,1}(ni+1,1);
end
nb=size(b, 1);
for k=1:nb
    if (~isempty(A{b(k,1),1}{b(k,2),1}))
        B{b(k,1),1}{b(k,2),1}=[B{b(k,1),1}{b(k,2),1};A{b(k,1),1}{b(k,2),1}(:,b(k,3))'];
    end
    if (~isempty(A{b(k,1),1}{b(k,2),2}))
        B{b(k,1),1}{b(k,2),2}=[B{b(k,1),1}{b(k,2),2};A{b(k,1),1}{b(k,2),2}(:,b(k,3))'];
    end
    if (~isempty(A{b(k,1),1}{b(k,2),3}))
        B{b(k,1),1}{b(k,2),3}=[B{b(k,1),1}{b(k,2),3};A{b(k,1),1}{b(k,2),3}(:,b(k,3))'];
    end
    ccb{b(k,1),1}{b(k,2),1}=[ccb{b(k,1),1}{b(k,2),1},c(1,offset{b(k,1),1}{b(k,2),1}+b(k,3))];
end
cb=[];
for i=1:ne
    ni=size(A{i,1},1);
    for j=1:ni
        cb=[cb, ccb{i,1}{j,1}];
    end
end
end
%*****

```

```

function v=coluna(A, je, ji, jr)
% function v=coluna(A, je, ji, jr)
% assembles v, dense array from A, NRBAF matrix.
% v'=A{je,1}{ji,1}(:,jr); zeros (...) ;A{je,1}{ji,2}(:,jr); zeros (...) ;A{je,1}{ji,3}(:,jr)

z=0;
nexternal=size(A,1);
for i=1:je-1
    ninternal=size(A{i,1},1);
    for j=1:ninternal
        z=z+size(A{i,1}{j,1},1);
    end
    z=z+size(A{i,1}{ninternal,2},1);
end
v=zeros (z,1);
ninternal=size(A{je,1},1);
z=0;
for j=1:ji-1
    z=z+size(A{je,1}{j,1},1);
end
v=[v; zeros(z,1)];
if (~isempty(A{je,1}{ji,1}))
    v=[v; A{je,1}{ji,1}(:,jr)];
end
z=0;
for j=ji+1:ninternal
    z=z+size(A{je,1}{j,1},1);
end
v=[v; zeros(z,1)];
if (~isempty(A{je,1}{ji,2}))
    v=[v; A{je,1}{ji,2}(:,jr)];
end
z=0;
for i=je+1:nexternal
    ninternal=size(A{i,1},1);
    for j=1:ninternal
        z=z+size(A{i,1}{j,1},1);
    end
    z=z+size(A{i,1}{ninternal,2},1);
end
v=[v; zeros(z,1)];
if (~isempty(A{je,1}{ji,3}))
    v=[v; A{je,1}{ji,3}(:,jr)];
end
end

```

```

function At=transp (A)
% function At=transpose (A)
% At=A', A and At are NRBAF
nexternal=size(A,1);
for i=1:nexternal
    ninternal=size(A{i,1},1);
    for j=1:ninternal
        At{i,1}{j,1}=A{i,1}{j,1}';
        At{i,1}{j,2}=A{i,1}{j,2}';
        At{i,1}{j,3}=A{i,1}{j,3}';
    end
end
function [Aaux, caux, baux, raux]=pplaux(A,d);
% function [Aaux, caux, baux, raux]=pplaux(A,d);
% generates the auxiliary linear problem to get the
% initial basis of the main linear problem.
% A and Aaux are NRBAF.
caux=[]; baux=[]; raux=[];
ds=sign(d);
nexternal=size(A,1);
for i=1:nexternal
    ninternal=size(A{i,1},1);
    for j=1:ninternal
        [m,n]=size(A{i,1}{j,1});
        mci=size(A{i,1}{j,2},1);
        mce=size(A{i,1}{j,3},1);
        mds=size(ds,1);
        dsi=ds(1:m);
        ds=ds(m+1:mds);
        Aaux{i,1}{j,1}=[A{i,1}{j,1}, diag(dsi)];
        Aaux{i,1}{j,2}=[A{i,1}{j,2}, zeros(mci, m)];
        Aaux{i,1}{j,3}=[A{i,1}{j,3}, zeros(mce, m)];
        caux=[caux,zeros(1,n),ones(1,m)];
        baux=[baux;[i*ones(m, 1), j*ones(m, 1), (n+1:n+m)']]];
        raux=[raux;[i*ones(n, 1), j*ones(n, 1), (1:n)']]];
    end
    mdi=size(Aaux{i,1}{ninternal,1}, 1);
    [m,n]=size(Aaux{i,1}{ninternal,2});
    mce=size(Aaux{i,1}{ninternal,3}, 1);
    mds=size(ds,1);
    dsi=ds(1:m);
    ds=ds(m+1:mds);
    Aaux{i,1}{ninternal,1}=[Aaux{i,1}{ninternal,1},zeros(mdi, m)];
    Aaux{i,1}{ninternal,2}=[Aaux{i,1}{ninternal,2},diag(dsi)];
    Aaux{i,1}{ninternal,3}=[Aaux{i,1}{ninternal,3}, zeros(mce, m)];
    caux=[caux,ones(1,m)];
    baux=[baux;[i*ones(m, 1), ninternal*ones(m, 1), (n+1:n+m)']]];
end
mdi=size(Aaux{nexternal,1}{ninternal,1}, 1);
mci=size(Aaux{nexternal,1}{ninternal,2}, 1);
[m,n]=size(Aaux{nexternal,1}{ninternal,3});
Aaux{nexternal,1}{ninternal,1}=[Aaux{nexternal,1}{ninternal,1},zeros(mdi, m)];
Aaux{nexternal,1}{ninternal,2}=[Aaux{nexternal,1}{ninternal,2},zeros(mci, m)];
Aaux{nexternal,1}{ninternal,3}=[Aaux{nexternal,1}{ninternal,3}, diag(ds)];
caux=[caux,ones(1,m)];
baux=[baux;[nexternal*ones(m, 1), ninternal*ones(m, 1), (n+1:n+m)']]];

```

```

function [bo, Vo, o, ipassos, nreinv]=simplex (A, d, c, b, r)
% function [bo, Vo, o, ipassos, nreinv]=simplex (A, d, c, b, r)
% Calculates o=max (cx) | Ax=d
% stops if it finds an optimal solution or
% the the polyhedral set is unbounded or
% took more than npassos and did not find
% the optimal solution.
% b0 is the initial basis
% bo is the optimal basis
% Vo is the optimal basic variables values and
% o is the optimal objective function value.
% A is ROW angular blocked, but we factorize
% the basis transpose. To do so, B=A(:,b)' and so does R.
% monta tranposes the basis.
% nreinv is the number of reinversions made.
% ipassos is the number of simplex loops.

EPSILON=1E-15;
%EPSILON=1E-10 RESULTS IN BAD QUALITY IN THE RESULTS: max(Vo) ~ 1.5
%and error (relative to dense solver and every step re-factorization)
% of 1e-2.

nreinv=-1;
npassos=2000;
ipassos=0;
[B, cb]=monta (A, c, b);
[R, cr]=monta (A, c, r);
% U=sqrnrhaf(B);
n2q=1;
while (npassos>=ipassos)
    ipassos=ipassos+1;

    % [B, cb]=monta (A, c, b);
    % [R, cr]=monta (A, c, r);
    % U=sqrnrhaf(B);
    n2q
    if (n2q>EPSILON)
        U=sqrnrhaf(B);
        nreinv=nreinv+1;
    end

    cbtil=emx(R, uix(U, utix(U, etmx(B, cb'))));
    z=cbtil-cr;
    dtil=emx(B, uix(U, utix(U, d)));
    [aux, j]=max(z);
    if (aux<1e-12)
        'optimal solution'
        Vo=dtil;
        o= cb*dtil;
        bo=b;
        return
    end
    je=r(j,1); ji=r(j,2); jr=r(j,3);
    vcr=cr(1,j);
    rj=coluna (A, je, ji, jr);
    rtilj=emx(B, uix(U, utix(U, rj)));
    k=find (rtilj > 1e-12);
    epsilon=dtil(k)./rtilj(k);
    [epsilon, b(k,:)]
%
    if (isempty (k))
        'unbounded problem'
        Vo=[];
        bo=[];
        o=[];
        return
    end
    [aux, i]= min (epsilon);
    i=k(i);
    ie=b(i, 1); ii=b(i, 2); ir=b(i, 3);
    vcb=cb(i);

```



```

% this is swap
sir=blockpos (b, i);
sjr=blockpos (r, j);

idi=zeros (0, size(B{ie, 1}{ii, 1},2));
if (~isempty(B{ie, 1}{ii, 1}))
    idi=B{ie, 1}{ii, 1}(sir,:); B{ie, 1}{ii, 1}(sir,:)=[];
end
ici=zeros (0, size(B{ie, 1}{ii, 2},2));
if (~isempty(B{ie, 1}{ii, 2}))
    ici=B{ie, 1}{ii, 2}(sir,:); B{ie, 1}{ii, 2}(sir,:)=[];
end

ice=B{ie, 1}{ii, 3}(sir,:); B{ie, 1}{ii, 3}(sir,:)=[];

jdi=zeros (0, size(R{je, 1}{ji, 1},2));
if (~isempty(R{je, 1}{ji, 1}))
    jdi=R{je, 1}{ji, 1}(sjr,:); R{je, 1}{ji, 1}(sjr,:)=[];
end

jci=zeros (0, size(R{je, 1}{ji, 2},2));
if (~isempty(R{je, 1}{ji, 2}))
    jci=R{je, 1}{ji, 2}(sjr,:); R{je, 1}{ji, 2}(sjr,:)=[];
end
jce=R{je, 1}{ji, 3}(sjr,:); R{je, 1}{ji, 3}(sjr,:)=[];
B{je, 1}{ji, 1}= [jdi; B{je, 1}{ji, 1}];
B{je, 1}{ji, 2}= [jci; B{je, 1}{ji, 2}];
B{je, 1}{ji, 3}= [jce; B{je, 1}{ji, 3}];
R{ie, 1}{ii, 1}= [R{ie, 1}{ii, 1}; idi];
R{ie, 1}{ii, 2}= [R{ie, 1}{ii, 2}; ici];
R{ie, 1}{ii, 3}= [R{ie, 1}{ii, 3}; ice];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[indice, b]= updatel(b, je, ji, jr, i);
cb= update (cb, vcr, indice, i);
[indice, r]= updaten(r, ie, ii, ir, j);
cr= update (cr, vcb, indice, j);
[U, n2q]=upo (U, ie, ii, idi, ici, ice);
U=upi (U, je, ji, jdi, jci, jce);
end
'npassos reached'
Vo=[];
bo=[];
o=[];

function c=update(c, vc, indice, j)
% Removes the j-th column of c (ordinary vector) and
% inserts vc in indice row of c.

c(:,j)=[];
n=size (c, 2);
c=[c(1, 1:indice-1), vc, c(1, indice:n)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% macro to test the NRBAF simplex method.
clear all;

load samples

% generates a square matrix - part of the polihedral set
nexternal=12;
sparsity=0.90;
formfactor=2;
nec=10;

P=fnrbafe (nexternal, nec, sparsity, formfactor); % this is NCBAF - transpose it

% the following lines convert NRBAF in RBAF eliminating the internal coupling blocks.
mexternal=size(P,1);
for ki=1:mexternal

```

```

        minternal=size(P{ki,1},1);
        for kj=1:minternal
            P{ki,1}{kj,2}=[];
        end
    end
end
% end of conversion.

Pt=transp(P);
[m,n]=nrsize (P)      % nrsize reports the size for NCBAF ' '
d=rand(n, 1);

% the entire polihedral set
[A, c, b, r]=pplaux (Pt, d);

% the simplex
flops(0);
[bo, Vo, o, ipassos, nreinv]=simplex (A, d, c, b, r);
if (isempty (bo)) return;
end;
v=org (b, r, bo, Vo);
fNR=flops;

% compare with dense matrix solver
AD=nr2d(transp(A))';

b=find(c~=0);
r=find(c==0);

flops(0);
[Vo,b,xb,r] = simpsimp(AD,d,c,b,r);
fd=flops;

e= norm (v-xb);
if (e>1e-8) save erro
end

[nexternal, nec, sparsity, formfactor, n, fNR, fd, e, sum(c(b)), ipassos, nreinv]
samples = [samples; [nexternal, nec, sparsity, formfactor, n, fNR, fd, e, sum(c(b)), ipassos, nreinv]];

% saves statistics
save samples samples

spy (AD);

%*****
function y=etmx(B, x)
% function y=etmx(B, x)
% y=B'*x, B nrbafe.
nexternal=size(B,1);
y=[];
yce=zeros (size(B{nexternal,1}{1,3},2), 1);
for i=1:nexternal
    ninternal=size(B{i,1},1);
    yci=zeros (size(B{i,1}{ninternal,2},2), 1);
    for j=1:ninternal
        [md, nd]=size (B{i,1}{j,1});
        n=size(x,1);
        xdi=x(1:md);
        x=x(md+1:n);
        if (~isempty(B{i,1}{j,1})) ydi=B{i,1}{j,1}'*xdi; end;
        if (~isempty(B{i,1}{j,2})) yci=yci+B{i,1}{j,2}'*xdi; end;
        if (~isempty(B{i,1}{j,3})) yce=yce+B{i,1}{j,3}'*xdi; end;
        y=[y;ydi];
    end
    y=[y;yci];
end
y=[y;yce];
function v=coluna(A, je, ji, jr)
% function v=coluna(A, je, ji, jr)
% assemblies v, dense array from A, NCBAF matrix.
% v'=A{je,1}{ji,1}(:,jr); zeros (...) ;A{je,1}{ji,2}(:,jr); zeros (...) ;A{je,1}{ji,3}(:,jr)

```

```

z=0;
nexternal=size(A,1);
for i=1:je-1
    ninternal=size(A{i,1},1);
    for j=1:ninternal
        z=z+size(A{i,1}{j,1},1);
    end
end
v=zeros(z,1);
ninternal=size(A{je,1},1);
z=0;
for j=1:ji-1
    z=z+size(A{je,1}{j,1},1);
end
if (~isempty(A{je,1}{ji,1}))
    v=[v; zeros(z,1); A{je,1}{ji,1}(:,jr)];
end
z=0;
for j=ji+1:ninternal
    z=z+size(A{je,1}{j,1},1);
end
if (~isempty(A{je,1}{ji,2}))
    v=[v; zeros(z,1); A{je,1}{ji,2}(:,jr)];
end
z=0;
for i=je+1:nexternal
    ninternal=size(A{i,1},1);
    for j=1:ninternal
        z=z+size(A{i,1}{j,1},1);
    end
end
if (~isempty(A{je,1}{ji,3}))
    v=[v; zeros(z,1); A{je,1}{ji,3}(:,jr)];
end

function nrdump (B);
% screen dump of all blocks in B, NRBAF
% shows the block external/internal indices.
nexternal=size(B,1)
for i=1:nexternal
    ninternal=size(B{i,1}, 1)
    for j=1:ninternal
        for k=1:3
            [m, n]=size(B{i, 1}{j, k});
            sprintf ('B{%d, 1}{%d, %d} is (%d x %d)', i, j, k, m, n)
        end
    end
end

for i=1:nexternal
    ninternal=size(B{i,1}, 1);
    for j=1:ninternal
        for k=1:3
            sprintf ('B{%d, 1}{%d, %d}', i, j, k)
            B{i, 1}{j, k}
        end
    end
end

function v=org (b, r, bo, Vo)
% function v=org (b, r, bo, Vo)
% combines Vo, bo, array and index array into
% v dense positional array.
% (b U r) must be the entire polihedral set.
% no index can be greater than 99.

bg=10000*b(:,1)+100*b(:,2)+b(:,3);
bog=10000*bo(:,1)+100*bo(:,2)+bo(:,3);
rg=10000*r(:,1)+100*r(:,2)+r(:,3);
ag=sort([bg;rg]);
v=zeros(length(ag),1);
n=length(bog);

```

```

for i=1:n
    k=find(ag==bog(i,1));
    if (~k)
        'index of optimal solution not in the polihedral set.'
    else
        v(k,1)=Vo(i);
    end
end
end

%*****
function [Vo,b,x,r] = simpsimp(A,d,c,b,r)
% function [Vo,b,x,r] = simpsimp(A,d,c,b,r)

[m,n] = size(A);
acabou = 0;

while ~acabou
    B = A(:,b);
    R = A(:,r);

    Rtil = B\R;
    dtil = B\d;
    z = c(b)*Rtil - c(r);

    if ( any(z > 0) )
        indices = find(z > 0);
        [aux,j] = max(z); %maior declive

        k = find( Rtil(:,j) > 0 );

        epsilon = dtil(k) ./ Rtil(k,j);
        [aux,i] = min( epsilon );
        i = k(i);
        b = [b r(j)];
        r = [r b(i)];
        b(i) = [];
        r(j) = [];

    else % acabou
        acabou = 1;
    end %if indices

    x = zeros(n,1);
    x(b) = A(:,b)\d;
    Vo = c * x;
end %while
%*****

function [Vo,b,xb,r] = smain(A,d,c)
% function [Vo,b,xb,r] = smain(A,d,c)

criterio = 2;

trace = 0;

[m,n] = size(A);
aux = diag(sign(d));
Aaux = [A aux];
czeros = zeros(1,n);
uns = ones(1,m);
caux = [czeros uns];
r = 1:n;
b = n+1:n+m;
[Vo,b,xb,r] = simpsimp(Aaux,d,caux,b,r,criterio,trace);
if (Vo == 0)
    raux = find(r <= n);
    r = r(raux);
    [Vo,b,xb,r] = simpsimp(A, d, c, b, r, criterio, trace);
else
    disp('Problema sem solucao');
end

```

```

end %if
if (trace)
    diary;
end

function NR=fnrbafe (m, nec, s, formfactor)
% function NR=fnrbafe (m, s, formfactor)
% builds NR, rectangular, nrbafe with m diagonal external blocks
% each diagonal external block has a random positive number
% of diagonal internal blocks. Internal diagonal (form*mid x nid)
% blocks have 0<=(mid-nid)<=3. Internal coupling (form*mic x nic)
% blocks have 0<mic<=3. External diagonal blocks (form*med x ned)
% are more likely to have (med > ned) - not equal, never
% smaller. External coupling are (mec x nec). nec is an argument
% the blocks are generated in formfactor sub-blocks with descending sparsity.

mininternal=floor(4*rand(m, 1)+1);
nid=floor(7*rand(sum(mininternal), 1)+1);
d=floor(5*rand(sum(mininternal), 1));
mid=nid+d;
k=1; dnic=0; nic=zeros(m, 1);
for i=1:m
    dmde=sum(d(k:k+mininternal(i,1)-1, 1));
    r=0;
    while r<0.8
        % at least 70% of the difference must be internal coupling
        r=rand(1,1);
    end
    nic(i, 1)=floor (r*dmde);
    dnic= dnic+dmde-nic(i,1);
    k=k+mininternal(i,1);
end
NR=cells(m, 1); k=1;
for i=1:m
    NR{i,1}=cells(mininternal(i,1), 3);
    for j=k: k+mininternal(i,1)-1
        for l=1:formfactor
            NR{i,1}{j-k+1, 1}=[NR{i,1}{j-k+1, 1}; rdensa(mid(j), nid(j), s/3)];
            NR{i,1}{j-k+1, 2}=[NR{i,1}{j-k+1, 2}; rdensa(mid(j), nic(i), s/1.5)];
            NR{i,1}{j-k+1, 3}=[NR{i,1}{j-k+1, 3}; rdensa(mid(j), nec, s)];
        end
    end
    k=k+mininternal(i,1);
end
function s=nrstats (samples)
% generates distributions on the samples of TESTE.
% [nexternal, nec, sparsity, formfactor, n, fNR, fd, e, sum(c(b)), ipassos, nreinv]
[ss, sidx]=sort(samples(:,5));
samples=samples(sidx,:);
n=size(samples,1);
p=floor (n/10);
s=[]; v=0; fNR=0; fd=0;
for i=1:n
    v=v+samples(i,5);
    fNR=fNR+samples(i,6);
    fd=fd+samples(i,7);
    if (mod (i,p)==0)
        s=[s;[v/p, fNR/p, fd/p]];
        v=0; fNR=0; fd=0;
    end
end
if (mod (n,p)~=0)
    s=[s;[v/mod(n,p), fNR/mod(n,p), fd/mod(n,p)]];
end
figure(1);
plot (s(:,1), s(:,2));

figure(2);
plot (s(:,1), s(:,2), s(:,1), s(:,3));

figure(3);

```

```

plot (s(:,1), s(:,3)./s(:,2));
s=[
% número de variáveis básicas Expr1001      Expr1002
32      388972 1536059
34      545540 2827635
40      983304 5914264
43      903059 5837216
44      1490673 10923448
45      1330866.66666667      11311155.3333333
48      1626653 16847463
49      2184275 22414597.5
50      1461191 14319447
51      1388960 11162786
54      3358616.5      36155170.5
56      2541218.33333333      29037912.6666667
57      1701708.5      20458559.5
58      2922277.5      35815628.5
59      3052009 33014645
60      2855769 42158721
64      2743241 40310672
65      5734541 87066476
66      3413579 54872106
67      3817117.5      61487013.5
68      3698239 68959035
71      4728164.5      89277812
72      3787874 66881762
73      5931529.33333333      114267749
74      4585111.66666667      92446903.6666667
75      4394660 79077981
76      3804329 87502900
78      7602143 169110790
79      4922152 112618269.666667
81      4768538 92969108.6666667
82      6402512 159791120
83      5896940 148881556
84      6150442.5      142071220
87      5067734 148618660
88      6395030 152214460
89      4657023 103907240
90      5324595.5      134915620
91      5742496 153443390
92      8425236 265848990
97      10103424      323020206.666667
98      9075008.5      337397485
99      11339555      359636830
100     12158178      465530820
103     10399442      421475320
104     10800867      362107150
106     10355007      407562060
107     6484638 203613560
108     11492754      488714150
111     7811526 203286970
112     11324597      422141260
113     13678125      589776530
114     14483048      651514930
117     9380231 396625120
119     14646127      738588510
120     15708931      763842940
121     18254892      894489400
124     11742151      528171190
125     13830937      697593360
127     14803592      766656110
128     17400851      853356575
131     20285322.6666667      1158974733.33333
133     23770608      1276856700
136     16771771      942580570
137     23618035      1415428500
138     15585699      934477550
144     27984166      2110655400
150     29550360      2007910000
151     17967336.5      1151751170

```

```

153     19385382      1024079800
157     21473997      1558274400
165     26120650      2204091000
176     48571275      4830604300
178     26106990      2117464700
183     47633751.5    5245351400
186     39442844      3701403700
192     32456410      3253310800
195     38470989      3939110000
197     39836982      4428578000
205     44443634      4610536600
206     35862694      4111752400
231     69078035      10412086000
];

% speedup
s(:,4)=s(:,3)./s(:,2);
pspeed=polyfit (s(:,1), s(:,4), 2);
% desempenho
pnr=polyfit (s(:,1), s(:,2), 4);
pdenso=polyfit (s(:,1), s(:,3), 4);
s=s(3:size(s,1),:);

figure (1);
plot (s(:,1), s(:,4), 'x', s(:,1), polyval (pspeed, s(:,1)));
title ('speedup');
xlabel ('número de variáveis básicas');
ylabel ('flops(DENSO)/flops(NRBAF)');
figure (2);
plot (s(:,1), s(:,2), 'x', s(:,1), polyval(pnr, s(:,1)));
title ('Desempenho NRBAF');
xlabel ('número de variáveis básicas');
ylabel ('flops(NRBAF)');

figure (3);
plot (s(:,1), s(:,3), 'x', s(:,1), polyval(pdenso, s(:,1)));
title ('Desempenho DENSO');
xlabel ('número de variáveis básicas');
ylabel ('flops(DENSO)');

% generates a square matrix - part of the polihedral set
nexternal=3;
sparsity=0.90;
formfactor=2;
nec=10;

P=fnrbafe (nexternal, nec, sparsity, formfactor); % this is NCBAF - transpose it
Pt=transp(P);
[m,n]=nrsize (P)      % nrsize reports the size for NCBAF
d=rand(n, 1);

% the entire polihedral set
[A, c, b, r]=pplaux (Pt, d);

AD=nr2d(transp(A));
figure (4);
spy (AD');

nexternal=12;

P=fnrbafe (nexternal, nec, sparsity, formfactor); % this is NCBAF - transpose it
Pt=transp(P);
[m,n]=nrsize (P)      % nrsize reports the size for NCBAF
d=rand(n, 1);

% the entire polihedral set
[A, c, b, r]=pplaux (Pt, d);

AD=nr2d(transp(A));
figure (5);
spy (AD');

```

Parte IV

Código C do *solver* para problemas na
forma angular bloqueada

Header files

VETOR.HPP

```
#ifndef __VETOR_HPP__
#define __VETOR_HPP__

#include <stdio.h>

#define _DEBUG_ENTRY_ /* Manda uma mensagem como primeira
                       instrucao da funcao. Nem todas as funcoes tem
                       este header pois algumas sao tao simples que
                       a mensagem ao inves de ajuda, apenas polui.*/
#define _DEBUG_END_ /* Manda uma mensagem antes de retornar. */
#define _DEBUG_ALLOC_ /* Manda mensagens quando alocar/desalocar memoria. */
#define _VERIFY_ALLOC_ /* Verifica todas as alocacoes, sem esta macro,
                       confiamos que o SO "nunca falhe". */
#define _VERIFY_BOUNDS_ /* Verifica os bounds da matriz no operador de acesso
                       e nas funcoes de acesso direto. */

class vetor {
    int n, /* Tamanho corrente. */
        nmax; /* Tamanho maximo - que corresponde ao tamanho alocado. */
    double *V;
public:
    friend void initclassvetor (void);
    /* Inicializa variaveis estaticas da classe. */
    friend int getvetorAlloc (void);
    friend int getvetorInstance (void);
    vetor ();
    /* Construtor da classe - zera dimensoes e faz V apontar para NULL. */
    ~vetor ();
    create (int ni);
    /* Aloca espaco para o vetor, e inicializa as dimensoes.
       Inicialmente a dimensao e' a maxima. */
    void destroy (void);
    /* Remove o vetor da memoria e zera as dimensoes. */
    double &operator () (int i);
    /* Operador de acesso a elemento por referencia.
       Permite acesso transparente aos elementos do vetor. */

    void fprint (FILE *fp, const char *name);
    /* Imprime, com o nome <name>, em formato texto, o vetor em fp. */
    char *fread (FILE *fp, char *name);
    /* Le, em formato texto, o vetor em fp. E retorna o nome lido em name*/

    double *getline (void);
    /* retorna ponteiro para o array interno. */
    double **getpline (void);
    /* retorna ponteiro para o ponteiro do array interno. */

    void resize (int ni);
    /* Se o vetor tem tamanho maximo maior que ni, apenas atualiza n, senao,
       realoca o vetor. */
    int getsize (void);
    /* Retorna a dimensao corrente do vetor. Se retornar -1 indica erro. */
    int getmaxsize (void);
    /* Retorna a dimensao maxima do vetor. Se retornar -1 indica erro. */
    void copy (vetor *a);
    /* funcao de copia de vetor - conheco o operador de passagem por referencia em C++
       mas para as partes fundamentais, quero usar C++ em doses muito pequenas. */
    void matprt (FILE *fp, const char *name);
    /* imprime o vetor no arquivo em formato binario MATLAB nivel 1.0. */
    int update (double val, int i, int j);
    /* Elimina o elemento j e insere val na posicao i. */
    /* Convencao de nomes: funcoes todas em letras minusculas. */
    void setd (double *d, int ni) {V=d; n=ni; nmax=ni; }
    void minus (vetor *v);
    /* subtracao de vetores: this-=v; */
    double prod (vetor *v);
    /* produto entre vetores: d=this*v */
    void div (vetor *v);
    /* divisao elemento a elemento: this=this./v */
};
```

```

    int max (void);
    /* retorna o indice do maior elemento no vetor. */
    int min (void);
    /* retorna o indice do menor elemento no vetor. */
    int first (void);
    /* Retorna o indice do primeiro elemento positivo. */
};
#endif

```

DENSA.HPP

```

#ifndef __DENSA_HPP__
#define __DENSA_HPP__

#include <stdio.h>

#define _DEBUG_ENTRY_ /* Manda uma mensagem como primeira
                       instrucao da funcao. Nem todas as funcoes tem
                       este header pois algumas sao tao simples que
                       a mensagem ao inves de ajuda, apenas polui.*/
#define _DEBUG_OUT_ /* Manda uma mensagem antes de retornar. */
#define _DEBUG_ALLOC_ /* Manda mensagens quando alocar/desalocar memoria. */
#define _VERIFY_ALLOC_ /* Verifica todas as alocacoes, sem esta macro
                       confiamos que o SO "nunca falhe". */
#define _VERIFY_BOUNDS_ /* Verifica os bounds da matriz no operador de acesso
                       e nas funcoes de acesso direto. */

class densa {
    int m, n, /* Tamanho corrente. */
        mmax, nmax; /* Tamanho maximo -
                     que corresponde ao tamanho alocado. */

    double **M;
public:
    friend void initclassdensa (void);
    /* Inicializa variaveis estaticas da classe. */
    friend int getnAlloc (void);
    friend int getnInstance (void);
    densa ();
    /* Construtor da classe - zera dimensoes e faz M apontar para NULL. */
    ~densa ();
    create (int mi, int ni);
    /* Aloca espaco para a matriz, e inicializa as dimensoes.
    Inicialmente a dimensao e' a maxima. */
    void destroy (void);
    /* Remove a matriz da memoria e zera as dimensoes. */
    double &operator () (int i, int j);
    /* Operador de acesso a elemento por referencia.
    Permite acesso transparente aos elementos da matriz. */
    double *getline (int i);
    /* Funcao de acesso direto a elementos da linha -
    para inicializar elementos/combinar linearmente linhas
    sem a necessidade (overhead) do operador de acesso. */
    double **getpline (int i);
    /* Funcao de acesso aa cabeca da linha para fazer substituicao de
    linhas em tempo constante. */

    /* As funcoes de acesso direto (get*), enquanto publicas, subvertem
    o paradigma de orientacao a objeto, mas durante o desenvolvimento
    vou mante-las assim. */
    void fprintf (FILE *fp, const char *name);
    /* Imprime, com o nome <name>, em formato texto, a matriz em fp. */
    char *fread (FILE *fp, char *name);
    /* Le, em formato texto, a matriz em fp. E retorna o nome lido em name*/

    void resize (int mm, int nn);
    /* Redimensiona a matriz. Se desejarmos torna'-la maior que suas dimensoes
    maximas, ocorre realocacao (usando realloc de C), senao, se diminuirmos o
    numero de linhas, as linhas sao liberadas e se diminuirmos o numero de colunas,
    a realocacao nao e' feita. O motivo e': nesta representacao inserir linha
    em uma matriz (mxn) custa O(1) tempo e gera O(n) memoria, enquanto que para coluna,
    custa O(m) tempo e gera O(m) memoria, como o caso esperado e' m=n (matriz quadrada),
    e' muito mais caro realocar colunas. */
    int getsize (int d);
    /* Retorna a dimensao corrente da matriz. d==1 retorna linhas, d=2 retorna colunas
    se retornar -1 indica erro. */

```

```

int getmaxsize (int d);
/* Retorna a dimensao maxima da matriz. d=1 retorna linhas, d=2 retorna colunas
se retornar -1 indica erro. */
int shdown (void);
/* roda para baixo as linhas da matriz :
se a seq. de linhas era 1, 2, ... m passa a ser m, 1, 2, ..., m-1. */
int shup (void);
/* roda para cima as linhas da matriz :
se a seq. de linhas era 1, 2, ... m passa a ser 2, ..., m, 1. */
void matprt (FILE *fp, const char *name);
/* Escreve a matriz em formato binario MATLAB nivel 1.0 O arquivo tem que
ser aberto em modo binario. */
void copy (densa *A);
/* copia matriz. */
/* Convencao de nomes: funcoes todas em letras minusculas. */
};
#endif

```

LINEAR.HPP

```

#ifndef __LINEAR_HPP__
#define __LINEAR_HPP__

#include "vetor.hpp"
#include "densa.hpp"

int vim (densa *V, vetor *x);
/* retorna em x o resultado de y=inv(V)*x, supoe V triangular superior. */
int vtim (densa *V, vetor *x);
/* retorna em x o resultado de y=inv(V')*x, supoe V triangular superior. */
void vtam (densa *V, vetor *a);
/* retorna em a o resultado de y=V'*a. */
void vam (densa *V, vetor *a);
/* retorna em a o resultado de y=V*a */
#endif

```

QR.HPP

```

#ifndef __QR_HPP__
#define __QR_HPP__

#include "densa.hpp"
#include "vetor.hpp"

void vtm (densa *V, vetor *d);
/* retorna em d o resultado de V*d */
void vm (densa *V, vetor *d);
/* retorna em d o resultado de Vd */
int QR (densa *V, densa *C);
/* Faz a fatoracao QR de V, que passa a ser triangular superior e executa as
mesmas rotacoes em C. */
int QRh (densa *V, densa *C);
/* Faz a fatoracao QR de V, que deve ser de Hessenberg e volta a ser triangular
superior e executa as mesmas rotacoes em C. */
double QRx (vetor *x, densa *V, densa *C);
/* Faz as rotacoes para levar x a ||x||*I(:,1), faz as mesmas rotacoes em V e C
Retorna ||x|| */
void givcs (double *gc, double *gs, double x1, double x2);
/* Determina o angulo de rotacao (descrito pelo seu cosseno e seu seno) tq o
vetor (x1,x2) e' transformado em (||x1,x2||, 0) */
void givrt (double **v1, double **v2, int n, double gc, double gs);
/* Faz a rotacao entre v1 e v2 pelo angulo definido por gc e gs. */

#endif

```

RBAF.HPP

```

#ifndef __RBAF_H__
#define __RBAF_H__
#include "densa.hpp"
#include "vetor.hpp"

#define N_DEFAULT 5 /* Quantidade de blocos default para leitura. */

```

```

void initclassrbaf (void);
int getnrbaInstance(void);
int getnrbaAlloc(void);

class rbaf {
    densa **D,      /* diagonal */
    **C;           /* conexao */
    int n,          /* numero de blocos em cada vetor */
    nmax;          /* maximo numero de blocos possivel sem realocacao. */
public:
    rbaf ();
    /* Construtor: inicializa variaveis e incrementa contador de instancias. */
    ~rbaf ();
    /* Destrutor: decrementa contador de instancias. */
    create (int ni);
    /* Aloca os vetores de ni ponteiros para os blocos diagonais e blocos de
    conexao. */
    void createdblock (int i, int m, int n);
    /* Cria o i-esimo bloco diagonal de tamanho mxn */
    void createcblock (int i, int m, int n);
    /* Cria o i-esimo bloco de conexao de tamanho mxn */
    void destroy ();
    /* Libera todos os blocos e vetores */
    densa **getpdblock (int i);
    /* Retorna ponteiro para o ponteiro do i-esimo bloco diagonal. */
    densa *getdblock (int i);
    /* Retorna ponteiro para o i-esimo bloco diagonal. */
    densa **getpcbblock (int i);
    /* Retorna ponteiro para o ponteiro do i-esimo bloco de conexao. */
    densa *getcblock (int i);
    /* Retorna ponteiro para o i-esimo bloco de conexao. */
    void resize (int nn);
    /* Redimensiona os vetores de ponteiros para blocos diagonais e
    blocos de conexao. Se a matriz encolher, apaga os blocos em excesso. */
    int getsize (void);
    /* Retorna a quantidade atual de blocos
    (comprimento dos vetores de ponteiros para blocos) */
    int getmaxsize (void);
    /* Retorna a quantidade maxima de blocos alocaveis sem realocacao dos vetores
    (comprimento alocado dos vetores de ponteiros para blocos) */
    void fread (FILE *fp, char *name);
    /* Le uma matriz blocada do arquivo de entrada.
    (funcao capenga!!) */
    void fprint (FILE *fp, char *name);
    /* Escreve uma matriz blocada no arquivo de saida.
    convencao:
    <name>size= n;          // informa a quantidade de blocos da matriz
    <name>D<i>={...};       // bloco diagonal de indice i
    <name>C<i>={...};       // bloco de conexao de indice i
    */
    int QR (void);
    /* Fatoracao QR da base angular blocada.
    Acrescenta um bloco aa matriz. */
    int upi (int k, vetor *qa, vetor *qc);
    /* Funcao de entrada de linha na forma fatorada da base. k e o indice do
    bloco que vai receber a linha, qa e qc compoe a linha da base original que
    entra na base no bloco diagonal e no bloco de conexao, respectivamente.
    Quando a funcao retorna, qa e qc contem lixo. */
    double upo (int k, vetor *ba, vetor *bc);
    /* Funcao de saida de linha da forma fatorada da base.
    O k-esimo bloco vai ter eliminada a linha que na base original e composta
    por ba no bloco diagonal e bc no bloco de conexao.*/
    void matprt (FILE *fp, char *name);
    /* Imprime a matriz blocada, seguindo a convencao de nomes de fprint
    no formato binario MATLAB nivel 1.0 */
    void copy (rbaf *A);

    int utix (vetor *v);
    int uix (vetor *v);
    int etmx (vetor *v);
    int emx (vetor *v);
};

#endif

```

IVETOR.HPP

```
/* Classe vetor de indices para matrizes na forma RBAF
   Fabio Nakano - 18/08/97
   */

#ifndef __IVETOR_HPP__
#define __IVETOR_HPP__

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define _VERIFY_BOUNDS_
#define _VERIFY_ALLOC_
#define _DEBUG_ENTRY_
#define _DEBUG_END_

typedef struct {
    int b,      /* bloco */
    i;         /* indice */
} dupla;

class ivetor {
    int n,      /* comprimento corrente */
    nmax,      /* comprimento alocado */
    bn,        /* numero de blocos corrente */
    bnmax;     /* numero maximo de blocos */
    dupla **tag, /* ponteiro para o primeiro elemento do bloco */
    *d;        /* ponteiro para dupla */

public:
    friend void initclassivetor (void);
    /* Inicializa variaveis estaticas da classe. */
    friend int getivetornAlloc (void);
    friend int getivetornInstance (void);
    ivetor ();
    /* Construtor da classe - zera dimensoes e faz V apontar para NULL. */
    ~ivetor ();
    create (int bi, int ni);
    /* Aloca espaco para o vetor, e inicializa as dimensoes.
       Inicialmente a dimensao e' a maxima. */
    void destroy (void);
    /* Remove o vetor da memoria e zera as dimensoes. */
    void fprint (FILE *fp, const char *name);
    /* Imprime, com o nome <name>, em formato texto, o vetor em fp. */
    char *fread (FILE *fp, char *name);
    /* Le, em formato texto, o vetor em fp. E retorna o nome lido em name*/
    void resize (int bi, int ni);
    /* Se o vetor tem tamanho maximo maior que ni, apenas atualiza n, senao,
       realoca o vetor. */
    int getblocksize (int bloco);
    /* Retorna a quantidade de elementos no bloco. */
    int getsize (void);
    /* Retorna a dimensao corrente do vetor. Se retornar -1 indica erro. */
    int getmaxsize (void);
    /* Retorna a dimensao maxima do vetor. Se retornar -1 indica erro. */
    int getblock (void);
    /* Retorna a quantidade de blocos do vetor. Se retornar -1 indica erro. */
    int getmaxblock (void);
    /* Retorna a quantidade maxima de blocos do vetor. Se retornar -1 indica erro. */
    void copy (ivetor *a);
    /* funcao de copia de vetor - conheco o operador de passagem por referencia em C++
       mas para as partes fundamentais, quero usar C++ em doses muito pequenas. */
    int updatel (int bloco, int indice, int j);
    /* elimina o j-esimo elemento do vetor (na ordem sequencial) e insere
       (bloco,indice) na primeira linha do bloco. */
    int updaten (int bloco, int indice, int j);
    /* elimina o j-esimo elemento do vetor (na ordem sequencial) e insere
       (bloco,indice) na ultima linha do bloco. */
    dupla &operator () (int i);
    void arrangetags (void);
    /* arruma os tags. */
    int blockpos (int i);
    /* devolve a posicao do elemento de indice sequencial i dentro de seu bloco. */
    /* Convencao de nomes: funcoes todas em letras minusculas. */

```



```
};  
#endif
```

Código C

VETOR.CPP

```
#include <stdio.h>  
#include <stdlib.h>  
#include <ctype.h>  
#include <string.h>  
#include "global.hpp"  
#include "vetor.hpp"  
  
static int nAlloc=0, /* Contador de instancias alocadas. */  
nInstance=0; /* Numero de instancias criadas. */  
/* A ideia com os contadores acima e' saber se temos mais memoria alocada  
que instancias, ou se liberamos memoria mais de uma vez. */  
  
vetor::vetor ()  
{  
    nInstance++;  
    n=0;  
    nmax=0;  
    V=NULL;  
}  
  
vetor::~~vetor ()  
{  
    nInstance--;  
}  
  
void initclassvetor (void)  
{  
    nInstance=0;  
    nAlloc=0;  
}  
  
int getvetornAlloc (void)  
{  
    return (nAlloc);  
}  
  
int getvetornInstance (void)  
{  
    return (nInstance);  
}  
  
vetor::create (int ni)  
{  
#ifdef _DEBUG_ENTRY_  
    printf ("Criando %d+1 instancia.\n", nAlloc);  
    /* ja que tem que ser a primeira instrucao... */  
#endif  
  
    V=(double*) malloc (ni*sizeof (double));  
#ifdef _VERIFY_ALLOC_  
    if (!V) {  
        printf ("Memoria insuficiente %d+1.\n", nAlloc);  
        exit (0);  
    }  
#endif  
    V--; /* sujeira para que o primeiro indice valido seja 1. */  
    nAlloc++;  
    n=ni;  
    nmax=n;  
#ifdef _DEBUG_END_  
    printf ("%d instancia criada %p.\n", nAlloc, V);  
#endif  
}  
  
void vetor::destroy (void)  
{
```

```

#ifdef _DEBUG_ENTRY_
    printf ("Removendo %d instancia %p.\n", nAlloc, V);
#endif
V++;
#ifdef _DEBUG_ALLOC_
    printf ("Liberando vetor %p.\n", V);
#endif
    free (V);
    nAlloc--;
    n=0;
    nmax=0;
    V=NULL;
#ifdef _DEBUG_END_
    printf ("%d instancia removida.\n", nAlloc);
#endif
}

double &vetor::operator () (int i)
{
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>n)) {
        printf ("%p(%d) fora dos limites do vetor.\n", V, i);
        exit (0);
    }
#endif
    return (V[i]);
}

void vetor::fprint (FILE *fp, const char *name)
{
    double *p;
    int i;
    fprintf (fp, "%s=[\n", name);
    p=V;
    p++;
    for (i=1;i<n;i++, p++) fprintf (fp, "%9le\t", *p);
    fprintf (fp, "%9le];\n", *p);
}

char *vetor::fread (FILE *fp, char *name)
{
    int c, in;
    long pos;
    int fisnum, j;
    char *pname;
    double *linha;
    fisnum=0;
    in=0;
    pname=name;
    do {
        c=fgetc (fp);
        if (isalnum(c)) *pname++=c;
    } while ((!feof (fp)) && (c!='('));
    *pname='\0';
    pos=ftell (fp);
    while ((c!='')&&(!feof(fp))) {
        c=fgetc(fp);
        switch (c) {
            case '\t':
            case '\n':
            case ',':
            case ']':
            case ';':
            case ' ': if (fisnum) in++; fisnum=0; break;
            default: if (isdigit (c) || (c=='e') || (c=='E') || (c=='.' ) || (c=='+' ) || (c=='-' ))
                fisnum=1;
                else fisnum=0;
        }
    }
    if (fseek (fp, pos, SEEK_SET)) {
        printf ("Parse error.");
        exit (0);
    }
    this->resize (in);
    linha=V;
    linha++;
}

```

```

    for (j=0;j<(in-1);j++, linha++) {
        if (fscanf (fp, " %lf ", linha)!=1) {
            printf ("syntax error.");
            exit (0);
        }
    }
    if (fscanf (fp, "%lf ]; ", linha)!=1) {
        printf ("syntax error.");
        exit (0);
    }
    return (name);
}

void vetor::resize (int nn)
{
    double *newline; /* ponteiro para linha realocada.*/,
#ifdef _DEBUG_ENTRY_
    printf ("Redimensionando vetor %p\n", V);
#endif
    if (!V) this->create (nn);
    if (nn>nmax) {
#ifdef _DEBUG_ALLOC_
        printf ("Realocando vetor %p\n", V);
#endif
        newline=(double *) realloc (V+1, nn*sizeof (double));
#ifdef _VERIFY_ALLOC_
        if (!newline) {
            printf ("Memoria insuficiente %d+1.\n", nAlloc);
            exit (0);
        }
#endif
        V=newline-1; /* sujeira para que o primeiro indice valido seja 1. */
        nmax=nn;
    }
    n=nn;

#ifdef _DEBUG_END_
    printf ("%p Realocado.\n", V);
#endif
}

double *vetor::getline (void)
{
    return (V);
}

double **vetor::getpline (void)
{
    return (&V);
}

int vetor::getsize (void)
{
    return (n);
}

int vetor::getmaxsize (void)
{
    return (nmax);
}

void vetor::copy (vetor *a)
{
    double *av;
    this->resize (a->getsize()); /* nao reproduz exatamente o vetor a, ja que
                                   ele pode ter tamanho maximo diferente do
                                   tamanho atual, mas isto pode funcionar ate
                                   como um simples garbage collector manual... */

    av=a->getline();
    memcpy (V+1, av+1, n*sizeof(double));
}

void vetor::matprt (FILE *fp, const char *name)
{
    int j;
    long type, mrows, ncols, imagf, namelen;

```



```

double *p;

mrows=1;
ncols=n;          /* por causa do cast implicito. */
type=0;           /* Little Endian, IEEE double, densa. */
imagf=0;          /* matriz real */
namelen=strlen (name)+1;
fwrite (&type, sizeof (long), 1, fp);
fwrite (&mrows, sizeof (long), 1, fp);
fwrite (&ncols, sizeof (long), 1, fp);
fwrite (&imagf, sizeof (long), 1, fp);
fwrite (&namelen, sizeof (long), 1, fp);
fwrite (name, sizeof (char), namelen, fp);
p=V+1;
for (j=1;j<=n;j++, p++) fwrite (p, sizeof (double), 1, fp);
}

int vetor::update (double val, int i, int j)
{
    double *pj, *pi, *src, *dest;
    int len;
#ifdef _VERIFY_BOUNDS_
    if ((j<=0)|| (j>(n+1)) || (i<=0) || (i>(n+1))) {
        printf ("%p(%d) ou %p(%d) fora dos limites do vetor.\n", V, j, V, i);
        exit (0);
    }
#endif
    len=0; src=NULL; dest=NULL;
    pi=V+i;
    pj=V+j;
    if (pj>pi) {
        src=pi;
        dest=pi+1;
        len=pj-pi;
    }
    else {
        if (pj<pi) {
            pi=pi-1;
            src=pj+1;
            dest=pj;
            len=pi-pj;
        }
    }
    memmove (dest, src, len*sizeof(double));
    *pi=val;
    return (0);
}

void vetor::minus (vetor *v)
{
    double *v1, *v2;
    int i;
    if (n!=v->getsize()) {
        printf ("%p minus: dimensions do not agree.", this->V);
        exit (0);
    }
    v1=this->V+1;
    v2=v->V+1;
    for (i=1;i<=n;i++, v1++, v2++) (*v1)-=(*v2);
}

double vetor::prod(vetor *v)
{
    int i;
    double r, *a, *b;
#ifdef _VERIFY_BOUNDS_
    if (n!=v->n) {
        printf ("%p prod: dimensi do not agree.", V);
        exit (0);
    }
#endif
    r=0; a=V+1; b=v->V+1;
    for (i=1;i<=n;i++, a++, b++) r+=(*a)*(*b);
    return (r);
}

```

```

void vetor::div(vetor *v)
{
    int i;
    double *a, *b;
#ifdef _VERIFY_BOUNDS_
    if (n!=v->n) {
        printf ("%p prod: dimensi do not agree.", V);
        exit (0);
    }
#endif
    a=V+1; b=v->V+1;
    for (i=1;i<=n;i++, a++, b++) (*a)/=(*b);
}

int vetor::max(void)
{
    int i, imax;
    double *a, max;
    if (!n) return (0);
    a=V+1; max=*a; imax=1;
    a++;
    for (i=2;i<=n;i++, a++) if ((*a)>max) { max=*a; imax=i; }
    return (imax);
}

int vetor::min(void)
{
    int i, imin;
    double *a, min;
    if (!n) return (0);
    a=V+1; min=*a; imin=1;
    a++;
    for (i=2;i<=n;i++, a++) if ((*a)<min) { min=*a; imin=i; }
    return (imin);
}

int vetor::first(void)
{
    int i;
    double *a;
    if (!n) return (0);
    a=V+1;
    for (i=1;i<=n;i++, a++) if ((*a)>0.0) return (i);
    return (0);
}

```

DENSA.CPP

```

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include "global.hpp"
#include "densa.hpp"

/* states of the file reader machine. */
#define INIT      0
#define NUMBER    1
#define PUNCT     2
#define SPACE     3
#define NEWLINE   4

static int nAlloc=0, /* Contador de instancias alocadas. */
nInstance=0;        /* Numero de instancias criadas. */
/* A ideia com os contadores acima e' saber se temos mais memoria alocada
que instancias, ou se liberamos memoria mais de uma vez. */

densa::densa ()
{
    nInstance++;
    m=0;
    n=0;
    mmax=0;
}

```

```

        nmax=0;
        M=NULL;
    }

densa::~densa ()
{
    nInstance--;
}

void initclassdensa (void)
{
    nInstance=0;
    nAlloc=0;
}

int getnAlloc (void)
{
    return (nAlloc);
}

int getnInstance (void)
{
    return (nInstance);
}

densa::create (int mi, int ni)
{
    int i;
    double **linha;      /* ponteiro para uma linha. */
#ifdef _DEBUG_ENTRY_
    printf ("Criando %d+1 instancia.\n", nAlloc);
/* ja que tem que ser a primeira instrucao... */
#endif

    /* Aloca vetor de ponteiros de linha. */
#ifdef _DEBUG_ALLOC_
    printf ("Alocando vetor de ponteiros da %d+1 instancia.\n", nAlloc);
#endif
    M=(double **) malloc ((mi+1)*sizeof (double *));
#ifdef _VERIFY_ALLOC_
    if (!M) {
        printf ("Memoria insuficiente %d+1.\n", nAlloc);
        exit (0);
    }
#endif
    linha=M;
    *linha=NULL;
    linha++;
    for (i=1;i<=mi;i++, linha++) {
        (*linha)=(double*) malloc (ni*sizeof (double));
#ifdef _VERIFY_ALLOC_
        if (!(*linha)) {
            printf ("Memoria insuficiente %d+1.\n", nAlloc);
            exit (0);
        }
#endif
#ifdef _DEBUG_ALLOC_
        printf ("%d-esimo vetor %p da matriz %p alocado.\n", i, *linha, M);
#endif
        (*linha)--;
        /* sujeira para que o primeiro indice valido seja 1. */
    }
    nAlloc++;
    m=mi;
    n=ni;
    mmax=m;
    nmax=n;
#ifdef _DEBUG_END_
    printf ("%d instancia criada %p.\n", nAlloc, M);
/* ja que tem que ser a primeira instrucao... */
#endif
}

void densa::destroy (void)
{
    int i;

```

```

    double **linha;
#ifdef _DEBUG_ENTRY_
    printf ("Removendo %d instancia %p.\n", nAlloc, M);
#endif
    linha=M;
    linha++;
    for (i=1;i<=m;i++, linha++) {
        (*linha)++;
    }
#ifdef _DEBUG_ALLOC_
    printf ("Liberando %d-esima linha %p da matriz %p.\n", i, *linha, M);
#endif
    free (*linha);
}
free (M);
nAlloc--;
m=0;
n=0;
mmax=0;
nmax=0;
M=NULL;
#ifdef _DEBUG_END_
    printf ("%d instancia removida.\n", nAlloc);
/* ja que tem que ser a primeira instrucao... */
#endif
}

double &densa::operator () (int i, int j)
{
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>m)) {
        printf ("%p(%d,%d) fora dos limites da matriz.\n", M, i, j);
        exit (0);
    }
    if ((j<=0)|| (j>n)) {
        printf ("%p(%d,%d) fora dos limites da matriz.\n", M, i, j);
        exit (0);
    }
#endif
    return (M[i][j]);
}

double *densa::getline (int i)
{
#ifdef _DEBUG_ENTRY_
    printf ("Retornando M(%d, :).%p\n", i, M);
#endif
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>m)) {
        printf ("%p(%d,:) fora dos limites da matriz.\n", M, i);
        exit (0);
    }
#endif
    return (M[i]);
}

double **densa::getpline (int i)
{
#ifdef _DEBUG_ENTRY_
    printf ("Retornando &M(%d, :).%p\n", i, M);
#endif
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>m)) {
        printf ("%p(%d,:) fora dos limites da matriz.\n", M, i);
        exit (0);
    }
#endif
    return (&(M[i]));
}

void densa::fprint (FILE *fp, const char *name)
{
    double *p;
    int i, j;
    fprintf (fp, "%s=[\n", name);
    for (i=1;i<=m;i++) {
        p=M[i];

```

```

        p++;
        for (j=1;j<n;j++, p++) fprintf (fp, "%9le\t", *p);
        fprintf (fp, "%9le\n", *p);
    }
    fputs (");\n", fp);
}

char *densa::fread (FILE *fp, char *name)
{
    int c, im, in;
    long pos;
    int estado, i, j;
    char *pname;
    double *linha;
    im=0;
    in=0;
    estado=INIT;
    pname=name;
    do {
        c=fgetc (fp);
        if (isalnum(c)) *pname++=c;
    } while ((!feof (fp)) && (c!='('));
    *pname='\0';
    pos=ftell (fp);
    while ((c!=']')&&(!feof(fp))) {
        c=fgetc(fp);
        switch (c) {
            case ',':
            case '\t':
            case ' ': if (estado==NUMBER) in++; break;
            case ']':
            case ';': if (estado==NEWLINE) break;
            case '\n': if (estado!=INIT) im++; if (estado==NUMBER) in++; break;
        }
        switch (c) {
            case '\n': if (estado!=INIT) estado=NEWLINE; break;
            case '\t':
            case ' ': if (estado!=INIT) estado=SPACE; break;
            case ']':
            case ',':
            case ';': estado=PUNCT; break;
            default: if (isdigit (c)|| (c=='e')|| (c=='E')|| (c=='.' )|| (c=='+' )|| (c=='-' ))
                estado=NUMBER;
            else {
                printf ("Invalid character read in %s.\n", name);
                exit (0);
            }
        }
    }
    if ((c!=']')&&(in%im)) {
        printf ("error reading matrix.");
        return (NULL);
    }
    in/=im;
    if (fseek (fp, pos, SEEK_SET)) {
        printf ("Parse error.");
        exit (0);
    }
    this->resize (im, in);
    for (i=1;i<=im;i++) {
        linha=M[i];
        linha++;
        for (j=0;j<(in-1);j++, linha++) {
            if (fscanf (fp, " %lf ", linha)!=1) {
                printf ("syntax error.");
                exit (0);
            }
        }
        if (i!=m) {
            if (fscanf (fp, " %lf ; ", linha)!=1) {
                printf ("syntax error.");
                exit (0);
            }
        }
        else {
            if (fscanf (fp, " %lf ]; ", linha)!=1) {

```

```

        printf ("syntax error.");
        exit (0);
    }
}
}
return (name);
}

void densa::resize (int mm, int nn)
{
    int i, mx;
    double **linha,          /* ponteiro para uma linha. */
           **newM,           /* ponteiro para vetor de ponteiros realocados. */
           *newline;         /* ponteiro para linha realocada. */
#ifdef _DEBUG_ENTRY_
    printf ("Realocando matriz %p\n", M);
#endif
    if (!M) this->create (mm, nn);
    if (nn>nmax) {
        linha=M;
        *linha=NULL;
        linha++;
        for (i=1;i<=mmax;i++, linha++) {
            newline=(double *) realloc ((*linha)+1, nn*sizeof (double));
#ifdef _VERIFY_ALLOC_
            if (!newline) {
                printf ("Memoria insuficiente %d+1.\n", nAlloc);
                exit (0);
            }
#endif
#ifdef _DEBUG_ALLOC_
            printf ("%d-esimo vetor %p da matriz %p realocado para %p.\n", i, *linha+1, M,
newline);
#endif
            *linha=newline-1;
            /* sujeira para que o primeiro indice valido seja 1. */
        }
        nmax=nn;
    }
    if (mm>m) {
        mx=(mm>mmax)?mmax:mm;
        linha=M+m+1;
        for (i=m+1;i<=mx;i++, linha++) {
            (*linha)=(double*) malloc (nmax*sizeof (double));
#ifdef _VERIFY_ALLOC_
            if (!(*linha)) {
                printf ("Memoria insuficiente %d+1.\n", nAlloc);
                exit (0);
            }
#endif
#ifdef _DEBUG_ALLOC_
            printf ("%d-esimo vetor %p da matriz %p realocado.\n", i, *linha, M);
#endif
            (*linha)--;
            /* sujeira para que o primeiro indice valido seja 1. */
        }
    }
    if (mm<m) {
        linha=M+mm+1;
        for (i=mm+1;i<=m;i++, linha++) {
            (*linha)++;
#ifdef _DEBUG_ALLOC_
            printf ("Liberando %d-esima linha %p da matriz %p.\n", i, *linha, M);
#endif
            free (*linha);
            *linha=NULL;
        }
    }
    if (mm>mmax) {
        /* Realoca vetor de ponteiros de linha. */
#ifdef _DEBUG_ALLOC_
        printf ("Realocando vetor de ponteiros da %d+1 instancia.\n", nAlloc);
#endif
        newM=(double **) realloc (M, (mm+1)*sizeof (double *));
#ifdef _VERIFY_ALLOC_

```

```

        if (!newM) {
            printf ("Memoria insuficiente %d+1.\n", nAlloc);
            exit (0);
        }
#endif
        M=newM;
        linha=M+mmax+1;
        for (i=mmax+1;i<=mm;i++, linha++) {
            (*linha)=(double*) malloc (nmax*sizeof (double));
#ifdef _VERIFY_ALLOC_
            if (!(*linha)) {
                printf ("Memoria insuficiente %d+1.\n", nAlloc);
                exit (0);
            }
#endif
#ifdef _DEBUG_ALLOC_
            printf ("%d-esimo vetor %p da matriz %p realocado.\n", i, *linha, M);
#endif
            (*linha)--;
            /* sujeira para que o primeiro indice valido seja 1. */
        }
        mmax=mm;
    }
    m=mm;
    n=nn;

#ifdef _DEBUG_END_
    printf ("%p Realocado.\n", M);
#endif
}

int densa::getsize (int d)
{
    switch (d) {
        case 1: return (m);
        case 2: return (n);
    }
    return (-1);
}

int densa::getmaxsize (int d)
{
    switch (d) {
        case 1: return (mmax);
        case 2: return (nmax);
    }
    return (-1);
}

int densa::shdown (void)
{
    double *tmp;
    int i;
    tmp=M[m];
    for (i=m;i>1;i--) M[i]=M[i-1];
    M[1]=tmp;
    return (OK);
}

int densa::shup (void)
{
    double *tmp;
    int i;
    tmp=M[1];
    for (i=1;i<m;i++) M[i]=M[i+1];
    M[m]=tmp;
    return (OK);
}

void densa::matprt (FILE *fp, const char *name)
{
    int i, j;
    long type, mrows, ncols, imägf, namelen;

    mrows=m;
    ncols=n;          /* por causa do cast implicit. */

```

```

type=0;          /* Little Endian, IEE double, densa. */
imagf=0;         /* matriz real */
namelen=strlen (name)+1;
fwrite (&type, sizeof (long), 1, fp);
fwrite (&mrows, sizeof (long), 1, fp);
fwrite (&ncols, sizeof (long), 1, fp);
fwrite (&imagf, sizeof (long), 1, fp);
fwrite (&namelen, sizeof (long), 1, fp);
fwrite (name, sizeof (char), namelen, fp);
for (i=1;i<=n;i++) {
    for (j=1;j<=m;j++) fwrite (&(M[j][i]), sizeof (double), 1, fp);
}
}

void densa::copy (densa *A)
{
    int i;
    double **pa, **pb;
    this->resize (A->m, A->n);
    pa=A->getpline(1);
    pb=this->getpline(1);
    for (i=1;i<=m;i++, pa++, pb++) memcpy (*pb+1, *pa+1, n*sizeof (double));
}

```

LINEAR.CPP

/* rotinas basicas.
Fabio Nakano - 05.05.1997*/

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "global.hpp"
#include "densa.hpp"
#include "vetor.hpp"
#include "linear.hpp"

```

```

int vim (densa *V, vetor *x)
{
    int i, j, n;
    double *xp,          /* aponta para a variavel que estamos resolvendo. */
           **plinha, *linha, /* aponta para a linha da matriz. */
           *xx, *xc;      /* percorre o vetor x das variaveis resolvidas. */
    n=x->getsize();
#ifdef _VERIFY_BOUNDS_
    if ((n!=V->getsize(1))||(V->getsize(1)!=V->getsize(2))) {
        printf ("vim - dimensions must agree.\n");
        exit (0);
    }
#endif
    plinha=V->getpline(n);
    xp=x->getline()+n; xx=xp;
    for (i=n; i>0;i--, xp--, plinha--) {
        xc=xx;
        linha=*plinha+n;
        for (j=n; j>i;j--, xc--, linha--) (*xp)-=(*xc)*(*linha);
        (*xp)/=(*linha);
    }
    return (OK);
}

```

```

int vtim (densa *V, vetor *x)
{
    int i, j, n;
    double *xp,          /* aponta para a variavel que estamos calculando */
           **plinha, *linha, /* aponta para a linha da matriz. */
           *xx, *xc;      /* percorre o vetor x das variaveis resolvidas */
    xp=x->getline();
    xp++;
    n=x->getsize();
#ifdef _VERIFY_BOUNDS_
    if ((n!=V->getsize(1))||(V->getsize(1)!=V->getsize(2))) {
        printf ("vtim - dimensions must agree.\n");
        exit (0);
    }
}

```



```

#endif
    plinha=V->getpline(1);
    xx=x->getline()+1;
    for (i=1; i<=n;i++, xp++, plinha++) {
        xc=xx+i;
        linha=*plinha;
        linha+=i;
        (*xp)/=(*linha);
        linha++;
        for (j=i+1; j<=n;j++, xc++, linha++) (*xc)-=(*xp)*(*linha);
    }
    return (OK);
}

void vtam (densa *V, vetor *a)
{
    int i, j, n, nv;
    vetor b; /* o resultado. */
    register double ai; /* o elemento de a que vai ser usado. */
    double **plinha, *linha, *pb, *pbx, *pa;
    double **psa, **psb;
    n=a->getsize();
    nv=V->getsize(2);
#ifdef _VERIFY_BOUNDS_
    if (n!=V->getsize(1)) {
        printf ("vtam - dimensions must agree.\n");
        exit (0);
    }
#endif
    b.create(nv);
    plinha=V->getpline(1);
    linha=*plinha+1;
    pbx=b.getline()+1;
    pb=pbx;
    pa=a->getline()+1;
    ai=*pa;
    for (j=1;j<=nv;j++, linha++, pb++) (*pb)=(*linha)*ai;
    /* tinha que escolher entre inicializar com zeros (um loop) ou inicializar com
    o primeiro valor valido (um loop - uma iteracao no loop seguinte. Optei pela
    segunda. */
    plinha++;
    pa++;
    for (i=2;i<=n;i++, plinha++, pa++) {
        linha=*plinha+1;
        pb=pbx;
        ai=*pa;
        for (j=1;j<=nv;j++, linha++, pb++) (*pb)+=(*linha)*ai;
    }
    a->resize (nv);
    psa=a->getpline();
    psb=b.getpline();
    pb=*psb;
    *psb=*psa;
    *psa=pb;
    b.destroy ();
}

void vam (densa *V, vetor *a)
{
    int i, j, n, nv;
    vetor b; /* o resultado. */
    double **plinha, *linha, *pa, *pax, *pb;
    double **psa, **psb;
    n=a->getsize();
    nv=V->getsize(1);
#ifdef _VERIFY_BOUNDS_
    if (n!=V->getsize(2)) {
        printf ("vam - dimensions must agree.\n");
        exit (0);
    }
#endif
    b.create(nv);
    plinha=V->getpline(1);
    linha=*plinha;
    pax=a->getline()+1;
    pb=b.getline()+1;

```

```

for (i=1;i<=nv;i++, plinha++, pb++) {
    linha=*plinha; linha++;
    pa=pax;
    *pb=0.0;
    for (j=1;j<=n;j++, linha++, pa++) (*pb)+=(*linha)*(*pa);
}
a->resize (nv);
psa=a->getpline();
psb=b.getpline();
pb=*psb;
*psb=*psa;
*psa=pb;
b.destroy ();
}

```

QR.CPP

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "global.hpp"
#include "densa.hpp"
#include "vetor.hpp"
#include "linear.hpp"
#include "qr.hpp"

int QR (densa *V, densa *C)
/* vou duplicar o codigo para cobrir o caso em que C=NULL, esperando que isso
aumente a eficiencia da funcao. Isto e' algo que seria interessante testar. */
{
    int i, j, mv, nv, nc;
    double **pv1, **pv2, **pc1, **pc2;
    double gc, gs;
    mv=V->getsize (1);
    nv=V->getsize (2);
    pv1=V->getpline(1);
    if (C) {
#ifdef _VERIFY_BOUNDS_
        if (mv!=C->getsize(1)) {
            printf ("QR - dimensions must agree.\n");
            exit (0);
        }
#endif
        nc=C->getsize (2);
        pc1=C->getpline(1);
        for (i=1;i<=nv;i++, pv1++, pc1++) {
            pv2=pv1+1;
            pc2=pc1+1;
            for (j=i+1;j<=mv;j++, pv2++, pc2++) {
                givcs (&gc, &gs, *(*pv1+i), *(*pv2+i));
                givrt (pv1, pv2, nv, gc, gs);
                givrt (pc1, pc2, nc, gc, gs);
            }
        }
        return (OK);
    }
    for (i=1;i<=nv;i++, pv1++) {
        pv2=pv1+1;
        /* getpline (i+1), i==nv dispara a
        protecao de acesso. */
        for (j=i+1;j<=mv;j++, pv2++) {
            givcs (&gc, &gs, *(*pv1+i), *(*pv2+i));
            givrt (pv1, pv2, nv, gc, gs);
        }
    }
    return (OK);
}

int QRh (densa *V, densa *C)
/* vou duplicar o codigo para cobrir o caso em que C=NULL, esperando que isso
aumente a eficiencia da funcao. Isto e' algo que seria interessante testar. */
{
    int i, mv, nv, nc;
    double **pv1, **pv2, **pc1, **pc2;
    double gc, gs;

```

```

mv=V->getsize (1);
nv=V->getsize (2);
pv1=V->getpline(1);
if (C) {
#ifdef _VERIFY_BOUNDS_
    if (mv!=C->getsize(1)) {
        printf ("QRh - dimensions must agree.\n");
        exit (0);
    }
#endif
    nc=C->getsize (2);
    pc1=C->getpline(1);
    for (i=1;i<mv;i++, pv1++, pc1++) {
        pv2=pv1+1;
        pc2=pc1+1;
        givcs (&gc, &gs, *(*pv1+i), *(*pv2+i));
        givrt (pv1, pv2, nv, gc, gs);
        givrt (pc1, pc2, nc, gc, gs);
    }
    return (OK);
}
for (i=1;i<mv;i++, pv1++) {
    pv2=pv1+1;
    /* getpline (i+1), i==nv dispara a
       protecao de acesso. */
    givcs (&gc, &gs, *(*pv1+i), *(*pv2+i));
    givrt (pv1, pv2, nv, gc, gs);
}
return (OK);
}

double QRx (vetor *x, densa *V, densa *C)
/* vou duplicar o codigo para cobrir o caso em que C=NULL, esperando que isso
aumente a eficiencia da funcao. Isto e' algo que seria interessante testar. */
{
    int j, m, nv, nc;
    double **pv, **pc, *px, a, b;
    double gc, gs;
    m=x->getsize();
#ifdef _VERIFY_BOUNDS_
    if (m!=V->getsize(1)) {
        printf ("QRx - dimensions must agree.\n");
        exit (0);
    }
#endif
    px=x->getline()+m;
    pv=V->getpline(m);
    nv=V->getsize(2);
    if (C) {
#ifdef _VERIFY_BOUNDS_
        if (m!=C->getsize(1)) {
            printf ("QRx - dimensions must agree.\n");
            exit (0);
        }
#endif
        nc=C->getsize (2);
        pc=C->getpline(m);
        for (j=m;j>1;j--, px--, pv--, pc--) {
            givcs (&gc, &gs, *(px-1), *px);
            a=gc*(*(px-1))-gs*(*px);
            b=gs*(*(px-1))+gc*(*px);
            *px=b;
            /* sei que isso vai valer zero sempre,
               mas pode ser um bom indicador de erro. */
            *(px-1)=a;
            givrt (pv-1, pv, nv, gc, gs);
            givrt (pc-1, pc, nc, gc, gs);
        }
        return (*px);
    }
    for (j=m;j>1;j--, px--, pv--) {
        givcs (&gc, &gs, *(px-1), *px);
        a=gc*(*(px-1))-gs*(*px);
        b=gs*(*(px-1))+gc*(*px);
        *px=b;
        /* sei que isso vai valer zero sempre,
           mas pode ser um bom indicador de erro. */
        *(px-1)=a;
        givrt (pv-1, pv, nv, gc, gs);
    }
}

```

```

    }
    return (*px);
}

void givcs (double *gc, double *gs, double x1, double x2)
{
    long double tau, gcos, gsen;
    if (fabs (x2)<EPS) {
        gcos=1.0; gsen=0.0;
    }
    else {
        if (fabs(x1)>=fabs(x2)) {
            tau=-x2/x1;
            gcos=1.0/sqrt(1+tau*tau);
            gsen=gcos*tau;
        }
        else {
            tau=-x1/x2;
            gsen=1.0/sqrt(1+tau*tau);
            gcos=gsen*tau;
        }
    }
    *gc=gcos; *gs=gsen;
}

void givrt (double **v1, double **v2, int n, double gc, double gs)
/* Esta funcao e' realmente esquisita, ja' que nao quero ter o trabalho de
copiar regioes de memoria, logo, trocar os ponteiros e' a solucao, mas nao
quero fazer alocao diretamente aqui, logo, uso as abstracoes de vetor e
de matriz. A esquisitice esta em que as classes nao tem nenhuma relacao
(nao sao parentes) apesar de serem estruturalmente semelhantes e estou
trocando estruturas internas - e' como trocar as "tripas" entre dois bichos
que tem pouca semelhanca um com o outro. MAS NAO PRECISAVA DE DOIS VETORES
AUXILIARES...*/
{
    vetor a, b;
    double *va, *vb, *vv1, *vv2;
    double **pv, *tmp;
    int i;
    a.create (n);
    b.create (n);
    va=a.getline(); vb=b.getline(); vv1=*v1; vv2=*v2;
    va++; vb++; vv1++; vv2++;
    for (i=1;i<=n;i++, va++, vb++, vv1++, vv2++) {
        *va=gc*(**vv1)-gs*(**vv2);
        *vb=gs*(**vv1)+gc*(**vv2);
    }
    pv=a.getpline();
    tmp=*pv;
    *pv=*v1;
    *v1=tmp;
    pv=b.getpline();
    tmp=*pv;
    *pv=*v2;
    *v2=tmp;
    a.destroy();
    b.destroy();
}
/*
void main (void)
{
    FILE *fpOut, *fpIn;
    densa B, C;
    vetor x, y;
    char nome[80];
    fpOut=fopen ("t.m", "wt");
    if (!fpOut) perror ("t.m");
    fpIn=fopen ("in.m", "rt");
    if (!fpIn) perror ("in.m");
    printf ("Li %s.\n", C.fread (fpIn, nome));
    printf ("Li %s.\n", B.fread (fpIn, nome));
    printf ("Li %s.\n", x.fread (fpIn, nome));
    fclose (fpIn);
    C.fprint (fpOut, "C");
    x.fprint (fpOut, "b");
    y.copy (&x);
}

```

```

    QR (&C, &B);
    B.fprint (fpOut, "B");
    fprintf (fpOut, "normx=%lf\n", QRx (&x, &C, &B));
    B.fprint (fpOut, "Bx");
    QRh (&C, &B);
    B.fprint (fpOut, "Bk");
    fclose (fpOut);
    y.destroy ();
    C.destroy ();
    B.destroy ();
}
*/

```

RBAF.CPP

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>

#include "global.hpp"
#include "densa.hpp"
#include "vetor.hpp"
#include "linear.hpp"
#include "qr.hpp"
#include "rbaf.hpp"

static int nInstance=0, nAlloc=0;

int getnrbaInstance(void) { return (nInstance); }
int getnrbaAlloc(void) { return (nAlloc); }

void initclassrbaf (void)
{
    nInstance=0;
    nAlloc=0;
}

rbaf::rbaf ()
{
    D=NULL;
    C=NULL;
    n=0;
    nmax=0;
    nInstance++;
}

rbaf::~~rbaf ()
{
    nInstance--;
}

rbaf::create (int ni)
{
#ifdef _DEBUG_ENTRY_
    printf ("Criando %d+1 instancia.\n", nAlloc);
    /* ja que tem que ser a primeira instrucao... */
#endif

    D=(densa**) calloc (ni, sizeof (densa*));
#ifdef _VERIFY_ALLOC_
    if (!D) {
        printf ("Memoria insuficiente %d+1.\n", nAlloc);
        exit (0);
    }
#endif
    D--; /* sujeira para que o primeiro indice valido seja 1. */
    C=(densa**) calloc (ni, sizeof (densa*));
#ifdef _VERIFY_ALLOC_
    if (!C) {
        printf ("Memoria insuficiente %d+1.\n", nAlloc);
        exit (0);
    }
#endif
}

```

```

C--; /* sujeira para que o primeiro indice valido seja 1. */
nAlloc++;
n=ni;
nmax=n;
#ifdef _DEBUG_END_
    printf ("%d instancia criada %p.\n", nAlloc, D);
#endif
}

void rbaf::createdblock (int i, int m, int n)
{
    D[i]=(densa*) calloc (1, sizeof (densa));
#ifdef _VERIFY_ALLOC_
    if (!D[i]) {
        printf ("Memoria insuficiente. %p\n", D);
        exit (0);
    }
#endif
    D[i]->create (m, n);
}

void rbaf::createcblock (int i, int m, int n)
{
    C[i]=(densa*) calloc (1, sizeof (densa));
#ifdef _VERIFY_ALLOC_
    if (!C[i]) {
        printf ("Memoria insuficiente. %p\n", C);
        exit (0);
    }
#endif
    C[i]->create (m, n);
}

void rbaf::destroy ()
{
    densa **p;
    int i;
#ifdef _DEBUG_ENTRY_
    printf ("Removendo %d instancia %p.\n", nAlloc, D);
#endif
    if (!D) return;
    D++;
#ifdef _DEBUG_ALLOC_
    printf ("Liberando vetor %p.\n", D);
#endif
    p=D; for (i=1;i<=n;i++, p++) if (*p) (*p)->destroy();
    free (D);
    D=NULL;
    C++;
#ifdef _DEBUG_ALLOC_
    printf ("Liberando vetor %p.\n", C);
#endif
    p=C; for (i=1;i<=n;i++, p++)
        if (*p) (*p)->destroy();
    free (C);
    nAlloc--;
    n=0;
    nmax=0;
    C=NULL;
#ifdef _DEBUG_END_
    printf ("%d instancia removida.\n", nAlloc);
#endif
}

densa **rbaf::getpdblock (int i)
{
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>n)) {
        printf ("%p(%d) fora dos limites do vetor.\n", D, i);
        exit (0);
    }
#endif
    return (&D[i]);
}

densa *rbaf::getdblock (int i)

```

```

{
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>n)) {
        printf ("%p(%d) fora dos limites do vetor.\n", D, i);
        exit (0);
    }
#endif
    return (D[i]);
}

densa **rbaf::getpcblock (int i)
{
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>n)) {
        printf ("%p(%d) fora dos limites do vetor.\n", C, i);
        exit (0);
    }
#endif
    return (&C[i]);
}

densa *rbaf::getcblock (int i)
{
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>n)) {
        printf ("%p(%d) fora dos limites do vetor.\n", C, i);
        exit (0);
    }
#endif
    return (C[i]);
}

void rbaf::resize (int nn)
{
    int i;
    densa **newline; /* ponteiro para linha realocada. */
    densa **pp, **pc, **pd;
#ifdef _DEBUG_ENTRY_
    printf ("Redimensionando vetor %p\n", D);
#endif
    if (!D) this->create (nn);
    if (nn>nmax) {
        D++;
#ifdef _DEBUG_ALLOC_
        printf ("Realocando vetor %p\n", D);
#endif
        newline=(densa **) realloc (D, nn*sizeof (densa*));
#ifdef _VERIFY_ALLOC_
        if (!newline) {
            printf ("Memoria insuficiente %d+1.\n", nAlloc);
            exit (0);
        }
#endif
        pp=newline+n;
        for (i=n+1; i<=nn;i++, pp++) *pp=NULL;
        D=newline-1; /* sujeira para que o primeiro indice valido seja 1. */
        C++;
        newline=(densa **) realloc (C, nn*sizeof (densa*));
#ifdef _VERIFY_ALLOC_
        if (!newline) {
            printf ("Memoria insuficiente %d+1.\n", nAlloc);
            exit (0);
        }
#endif
        pp=newline+n;
        for (i=n+1; i<=nn;i++, pp++) *pp=NULL;
        C=newline-1; /* sujeira para que o primeiro indice valido seja 1. */
        nmax=nn;
    }
    else {
        if (nn!=n) {
            pc=&C[nn+1]; pd=&D[nn+1];
            for (i=nn+1;i<=n;i++, pc++, pd++) {
                if (*pc) (*pc)->destroy(); *pc=NULL;
                if (*pd) (*pd)->destroy(); *pd=NULL;
            }
        }
    }
}

```

```

    }
}
n=nn;

#ifdef _DEBUG_END_
    printf ("%p Realocado.\n", D);
#endif
}

int rbaf::getsize (void)
{
    return (n);
}

int rbaf::getmaxsize (void)
{
    return (nmax);
}

void rbaf::fread (FILE *fp, char *name)
/* esta funcao esta muito capenga e le apenas matrizes de 3 blocos. */
{
    int i, j;
    char iname[80], *ic;
    densa *tmp;
    fgets (iname, 80, fp);
    ic=strchr (iname, '=');
    if (!ic) {
        printf ("Matrix %s size not found.\n", iname);
        exit (0);
    }
    *ic='\0';
    ic++;
    if (sscanf (ic, " %d ; ", &i)!=1) {
        printf ("Matrix %s size not found.\n", iname);
        exit (0);
    }
    ic=strstr (iname, "size");
    if (!ic) {
        printf ("%s: bad matrix size name.\n", iname);
        exit (0);
    }
    ic='\0';
    strcpy (name, iname);
    this->destroy();
    this->create (i);
    for (i=1;i<=n;i++) {
        tmp= (densa*) calloc(1, sizeof (densa));
        tmp->fread (fp, iname);
        ic=strrchr (iname, 'D');
        if (!ic) {
            printf ("%s: bad block name.\n", iname);
            exit (0);
        }
        *ic='\0';
        ic++;
        if (sscanf (ic, "%d ", &j)!=1) {
            printf ("Error reading block %s.\n", iname);
            exit (0);
        }
        D[i]= tmp;

        tmp= (densa*) calloc(1, sizeof (densa));
        tmp->fread (fp, iname);
        ic=strrchr (iname, 'C');
        if (!ic) {
            printf ("%s: bad block name:\n", iname);
            exit (0);
        }
        *ic='\0';
        ic++;
        if (sscanf (ic, "%d ", &j)!=1) {
            printf ("Error reading block %s.\n", iname);
            exit (0);
        }
        C[i]= tmp;
    }
}

```



```

    }
}

void rbaf::fprint (FILE *fp, char *name)
{
    int i;
    char iname[80];
    fprintf (fp, "%ssize=%d;\n", name, n);
    for (i=1;i<=n;i++) {
        sprintf (iname, "%sD%d", name, i);
        if (D[i]) D[i]->fprint (fp, iname);
        sprintf (iname, "%sC%d", name, i);
        if (C[i]) C[i]->fprint (fp, iname);
    }
}

void rbaf::matprt (FILE *fp, char *name)
/* E o cumulo do encapsulamento querer escrever um escalar e ter que converte-lo
em matriz para escrever uma matriz de um unico elemento!!! */
{
    int i;
    char iname[80];
    densa size;
    size.create (1,1);
    size(1,1)=n;
    sprintf (iname, "%ssize", name);
    size.matprt (fp, iname);
    for (i=1;i<=n;i++) {
        sprintf (iname, "%sD%d", name, i);
        if (D[i]) D[i]->matprt (fp, iname);
        sprintf (iname, "%sC%d", name, i);
        if (C[i]) C[i]->matprt (fp, iname);
    }
    size.destroy ();
}

int rbaf::QR ()
/* inicialmente uma matriz de n=3 blocos diagonais tem a seg. estrutura:
| D1   C1 |
| D2   C2 |
| D3   C3 |
depois de fatorada passa a ter a seg. estrutura:
| D1   C1 |
| D2   C2 |
| D3   C3 |
|      C4 |
D4 == NULL
C4 tem tamanho conhecido pois e' feito com os excessos dos blocos 1 a 3
antes da fatoracao.
Como notacao vamos usar C[n+1] = S. */
{
    int ms, ns, md, nd, nc, i, j;
    double *tmp, **pc, **ps;
    ms=0;
    nc=C[1]->getsize(2);
    for (i=1;i<=n; i++) {
        md=D[i]->getsize (1);
        nd=D[i]->getsize (2);
#ifdef _VERIFY_BOUNDS_
        if (md<nd) {
            printf ("D(%d) dimensions mismatch.\nCheck basic block dimensions.\n",
                    i);
            exit (0);
        }
#endif
        ms+=(md-nd);
    }
#ifdef _VERIFY_BOUNDS_
    if (ms!=nc) {
        printf ("C(%d) dimensions mismatch.\nCheck basic block dimensions.\n",
                n+1);
        exit (0);
    }
#endif
    this->resize (n+1);
    if (_heapchk ()!=_HEAPOK) {

```

```

        printf ("\n\nBAD depois do resize\n\n");
        exit (0);
    }
    this->createcblock (n, ms, nc);
    ps=C[n]->getpline (1);
    for (i=1;i<n; i++) {          /* fatora cada par D[i], C[i] */
        if (_heapchk ()!=_HEAPOK) {
            printf ("\n\nBAD %d\n\n", i);
            exit (0);
        }
        ms=D[i]->getsize (1);
        ns=D[i]->getsize (2);
        ::QR (D[i], C[i]);
        if (ns<ms) pc=C[i]->getpline (ns+1);
        for (j=ns+1; j<=ms; j++, ps++, pc++) {
            /* troca linhas de C[i] com linhas de C[n] */
            tmp=*pc;
            *pc=*ps;
            *ps=tmp;
        }
        D[i]->resize (ns, ns);
        C[i]->resize (ns, nc);
    }
    ::QR (C[n], NULL);
    return (OK);
}

int rbafe::upi (int k, vetor *qa, vetor *qc)
{
    int md, nd, mc, nc, ms, ns;
    densa *pc, *pd, *ps;
    double *tmp, **p, **v;
#ifdef _VERIFY_BOUNDS_
    if (k>n) {
        printf ("No %p(%d) block.=n", k, D);
        exit (0);
    }
#endif
    pc=C[k]; pd=D[k]; ps=C[n];
    md=pd->getsize (1); nd=pd->getsize (2);
    mc=pc->getsize (1); nc=pc->getsize (2);
    ms=ps->getsize (1); ns=ps->getsize (2);
    pd->resize (md+1, nd);
    pc->resize (mc+1, nc);
    pd->shdown();
    pc->shdown();
    p=pd->getpline(1);
    v=qa->getpline();
    memcpy (*p+1, *v+1, nd*sizeof (double));
    /*
    da' pau por causa da politica de realocacao se os tamanhos maximos forem diferentes, trocar
    as tripas nao basta e as linhas da matriz passam a ter comprimentos diferentes, o que e'
    pessimo pois causa muitos erros.
    */
    p=pc->getpline(1);
    v=qc->getpline();
    memcpy (*p+1, *v+1, nc*sizeof (double));
    // tmp=*p; *p=*v; *v=tmp;
    QRh (pd, pc);
    ps->resize (ms+1, ns);
    ps->shdown();
    v=ps->getpline (1);
    p=pc->getpline (mc+1);
    tmp=*p; *p=*v; *v=tmp; /* aqui ate' da' para fazer a troca das tripas pois C[i] tem o mesmo
                           numero de colunas de C[n+1]. */
    QRh (ps, NULL);
    pc->resize (mc, nc);
    pd->resize (md, nd);
    return (OK);
}

double rbafe::upo (int k, vetor *ba, vetor *bc)
{
    vetor qa, qc;
    densa *pc, *pd, *ps;

```

```

    int i, ms, ns, nc, mc, nd, md;
    double *pq, *pb;
    double **lc, *tmp, **ls;
    double d;
#ifdef _VERIFY_BOUNDS_
    if (k>n) {
        printf ("No %p(%d) block.=n", k, D);
        exit (0);
    }
#endif
    pc=C[k]; pd=D[k]; ps=C[n];
    mc=pc->getsize (1); nc=pc->getsize (2);
    md=pd->getsize (1); nd=pd->getsize (2);
    ms=ps->getsize (1); ns=ps->getsize (2);

/* UITET */
    qa.copy (ba);
    ::vtim (pd, &qa);
    qc.copy (&qa);
    ::vtam (pc, &qc);
    pq=qc.getline(); pq++;
    pb=bc->getline(); pb++;
    ns=bc->getsize();
    for (i=1;i<=ns;i++, pq++, pb++) (*pq)=(*pb)-(*pq);
    ::vtim (ps, &qc);
    qa.fprint (stdout, "qa");
    qc.fprint (stdout, "qc");

/* QR "as avessas" */
    qa.resize (md+1);
    qa(md+1)=QRx(&qc, ps, NULL);
    pd->resize (md+1, nd);
    pb=pd->getline (md+1)+1;
    for (i=1;i<=nd;i++, pb++) *pb=0.0;
    pc->resize (mc+1, nc);
    lc=pc->getpline (mc+1);
    ls=ps->getpline (1);
    tmp=*lc; *lc=*ls; *ls=tmp; // aqui da para trocar as tripas
    d=QRx (&qa, pd, pc);
    ps->shup ();
    ps->resize (ms-1, ns);
    pd->shup();
    pd->resize (md, nd);
    pc->shup();
    pc->resize (mc, nc);
    qa.destroy ();
    qc.destroy ();
    return (d);
}

int rbafe::uix (vetor *v)
{
    int i, j, k, nct;
    vetor aux, // vou fazer umas sujeiras com ele, mas para isso preciso
                mexer nos membros da instancia - esta funcao vai ser
                "friend" de vetor. */
        xc,
        xd;
    densa **pc, **pd;
    double *px, *py;
    nct=0;
    pc=C+1;
    for (i=1;i<=n;i++, pc++) nct+=(*pc->getsize(1);
#ifdef _VERIFY_BOUNDS_
    if (nct!=v->getsize()) {
        printf ("rbafe.vim: Dimensions do not agree.\n");
        exit (0);
    }
#endif
    aux.setd (v->getline()+nct-C[n]->getsize(2), C[n]->getsize(2));
    pc=C+n;
    ::vim(*pc, &aux);
    pc--;
    pd=D+n-1;
    xd.setd (aux.getline(), aux.getsize());
    for (i=n-1;i>0;i--, pc--, pd--) {

```

```

        xd.setd (xd.getline()-(*pc)->getsize(1), (*pc)->getsize(1));
        xc.copy (&aux);
        ::vam (*pc, &xc);
        k=xc.getsize ();
        px=xd.getline()+1;
        py=xc.getline()+1;
        for (j=1; j<=k; j++, px++, py++) (*px)-=(*py);
        xc.destroy ();
        ::vim (*pd, &xd);
    }
    return (OK);
}

int rbaf::utix (vetor *v)
{
    int i, j, k, nct, nc;
    vetor aux, /* vou fazer umas sujeiras com ele, mas para isso preciso
                mexer nos membros da instancia - esta funcao vai ser
                "friend" de vetor. */

        xc,
        xd;
    densa **pc, **pd;
    double *px, *py;
    nct=0;
    pc=C+1;
    for (i=1; i<=n; i++, pc++) nct+=(*pc)->getsize(1);
#ifdef _VERIFY_BOUNDS_
    if (nct!=v->getsize()) {
        printf ("rbaf.vtim: Dimensions do not agree.\n");
        exit (0);
    }
#endif
    pc=C+1;
    pd=D+1;
    aux.setd (v->getline()+nct-C[n]->getsize(2), C[n]->getsize(2));
    nc=0;
    for (i=1; i<=n; i++, pc++, pd++) {
        xd.setd (v->getline()+nc, (*pd)->getsize(2));
        nc+=(*pd)->getsize(2);
        ::vtim(*pd, &xd);
        xc.copy (&xd);
        ::vtam (*pc, &xc);
        k=xc.getsize ();
        px=aux.getline()+1;
        py=xc.getline()+1;
        for (j=1; j<=k; j++, px++, py++) (*px)-=(*py);
        xc.destroy ();
    }
    ::vtim (*pc, &aux);
    return (OK);
}

int rbaf::emx (vetor *v)
/* nao posso usar ::vam, ::vtam para fragmentos de v pois essas funcoes
realocam o vetor. */
{
    int i, j, k, nct, nc, ns, mcd, ncd, nx;
    vetor x; /* a resposta */
    densa **pc, **pd;
    double *pvc, *pvd, *pppc, *pppd, *px, **ppc, **ppd;
    register double r;
    nct=0;
    nx=0;
    pd=D+1;
    pc=C+1;
    ns=(*pc)->getsize(2);
    for (i=1; i<=n; i++, pd++, pc++) {
        if (*pd) nct+=(*pd)->getsize(2);
        if (*pc) nx+=(*pc)->getsize(1);
    }
#ifdef _VERIFY_BOUNDS_
    if ((nct+ns)!=v->getsize()) {
        printf ("rbaf.vam: Dimensions do not agree.\n");
        exit (0);
    }
#endif
}

```

```

pc=C+1;
pd=D+1;
x.create (nx);
px=x.getline()+1;
nc=1;
for (i=1;i<=n;i++, pc++, pd++) {
    ppc=(*pc)->getpline(1);
    ppd=(*pd)?(*pd)->getpline(1):NULL;
    mcd=(*pc)->getsize(1);
    for (j=1;j<=mcd;j++, ppc++, ppd++, px++) {
        pvc=v->getline()+nct+1;
        pvd=v->getline()+nc;
        r=0.0;
        if (*pd) {
            pppd=ppd+1;
            ncd=(*pd)->getsize(2);
            for (k=1;k<=ncd;k++, pppd++, pvd++) r+=((*pppd)*(*pvd));
        }
        if (*pc) {
            pppc=ppc+1;
            ncd=(*pc)->getsize(2);
            for (k=1;k<=ncd;k++, pppc++, pvc++) r+=((*pppc)*(*pvc));
        }
        *px=r;
    }
    nc+=((*pd)?(*pd)->getsize(2):0);
}
/*
    ppc=(*pc)->getpline(1);
    mcd=(*pc)->getsize(1);
    for (j=1;j<=mcd;j++, ppc++, px++) {
        pvc=v->getline()+nct-ns+1;
        pppc=ppc+1;
        r=0.0;
        ncd=(*pc)->getsize(2);
        for (k=1;k<=ncd;k++, pppc++, pvc++) r+=((*pppc)*(*pvc));
        *px=r;
    }
*/
ppc=x.getpline();
ppd=v->getpline();
px=(*ppd);
v->setd (*ppc, x.getsize());
*ppc=px;
x.destroy();
return (OK);
}

int rbaf::etmx (vetor *v)
/* nao posso usar ::vam, ::vtam para fragmentos de v pois essas funcoes
realocam o vetor. */
{
    int i, j, k, nct, nc, ns, mcd, ncd, nn, ms, md;
    vetor x; /* a resposta */
    densa **pc, **pd;
    double *pvc, *pvd, *pppc, *pppd, *px, *pxc, *pzd, **ppc, **ppd;
    nn=(D[n]?n:(n-1));
    nct=0;
    pd=D+1;
    /* Hipotese de matriz quadrada! */
    for (i=1;i<=nn;i++, pd++) nct+=(*pd)->getsize(2);
    ms=C[n]->getsize(2);
    nct+=ms;
    ns=C[n]->getsize(1);
#ifdef _VERIFY_BOUNDS_
    if (nct!=v->getsize()) {
        printf ("rbaf.vtam: Dimensions do not agree.\n");
        exit (0);
    }
#endif
    x.create (nct);
    px=x.getline()+1;
    for (i=1;i<=nct;i++, px++) *px=0.0;
    pc=C+1;
    pd=D+1;
    px=x.getpline();
    nc=1;
    md=1;

```

```

for (i=1;i<=nn;i++, pc++, pd++) { /* para cada bloco. */
    ppc=(*pc)->getpline(1);
    ppd=(*pd)->getpline(1);
    mcd=(*pd)->getsize(1);
    pvd=v->getline()+nc;
    for (j=1;j<=mcd;j++, ppc++, ppd++, pvd++) { /* para cada linha*/
        pxc=px+nct-ms+1;
        pxd=px+md;
        pppc=*ppc+1;
        pppd=*ppd+1;
        ncd=(*pd)->getsize(2);
        for (k=1;k<=ncd;k++, pppd++, pxd++) (*pxd)+=((*pppd)*(*pvd));
        ncd=(*pc)->getsize(2);
        for (k=1;k<=ncd;k++, pppc++, pxc++) (*pxc)+=((*pppc)*(*pvd));
    }
    md+=(*pd)->getsize(2);
    nc+=(*pc)->getsize(1);
}
if (nn!=n) {
    ppc=(*pc)->getpline(1);
    mcd=(*pc)->getsize(1);
    pvc=v->getline()+nct-ns+1;
    for (j=1;j<=mcd;j++, ppc++, pvc++) {
        pxc=x.getline()+nct-ms+1;
        pppc=*ppc+1;
        ncd=(*pc)->getsize(2);
        for (k=1;k<=ncd;k++, pppc++, pxc++) (*pxc)+=((*pppc)*(*pvc));
    }
}
ppc=x.getpline();
ppd=v->getpline();
px=(*ppc);
(*ppc)=(*ppd);
(*ppd)=px;
x.destroy();
return (OK);
}

void rbf::copy (rbf *A)
{
    int i;
    densa **pa, **pb, **pc, **pd;
    this->resize (A->n);
    pa=(this->D)+1;
    pb=(this->C)+1;
    pc=(A->D)+1;
    pd=(A->C)+1;
    for (i=1;i<=n;i++, pa++, pb++, pc++, pd++) {
        if (*pc) {
            if (!(*pa)) (*pa)=(densa*) calloc (1, sizeof (densa));
            (*pa)->copy (*pc);
        }
        if (*pd) {
            if (!(*pb)) (*pb)=(densa*) calloc (1, sizeof (densa));
            (*pb)->copy (*pd);
        }
    }
}

#ifdef _TESTE_RBAF_
void main (int argc, char *argv[])
{
    FILE *fpIn, *fpOut;
    char name[80];
    rbf B, U;
    vetor ba, bc, qa, qc, y;
    _heapinfo heap;
    if (argc!=3) {
        puts ("Argument count mismatch.");
        puts ("Usage: rbf <inputfilename> <outputfilename>");
        exit (0);
    }
    fpIn=fopen (argv[1], "rt");
    if (!fpIn) {
        perror (argv[1]);
        exit (0);
    }
}

```

```

    }
    fpOut=fopen (argv[2], "wb");
    if (!fpOut) {
        perror (argv[2]);
        exit (0);
    }
    B.fread (fpIn, name);
    ba.fread (fpIn, name);
    bc.fread (fpIn, name);
    y.fread (fpIn, name);
    qa.copy (&ba);
    qc.copy (&bc);
    U.copy (&B);
    B.matprt (fpOut, "B");
    qa.matprt (fpOut, "qa");
    qc.matprt (fpOut, "qc");
    y.matprt (fpOut, "y");
    U.QR ();
    U.matprt (fpOut, "U");
//    U.upo (2, &ba, &bc);
//    U.upi (2, &qa, &qc);
    U.matprt (fpOut, "UP");
    U.utix(&y);
    U.uix(&y);
    B.emx(&y);
    y.matprt (fpOut, "x");
    heap._pentry=NULL;
    if (_heapchk ()!=_HEAPOK) {
        printf ("\n\nBAD\n\n");
        exit (0);
    }
    B.destroy();
    U.destroy();
    qa.destroy ();
    qc.destroy ();
    ba.destroy ();
    bc.destroy ();
    y.destroy ();
    fclose (fpIn);
    fclose (fpOut);
}
#endif

```

IVETOR.CPP

```

/* Classe vetor de indices para matrizes na forma RBAF
   Fabio Nakano - 18/08/97
*/

#include "ivetor.hpp"
#include "vetor.hpp"

// #define _TEST_IVETOR_

static int nAlloc=0, /* Contador de instancias alocadas. */
nInstance=0; /* Numero de instancias criadas. */
/* A ideia com os contadores acima e' saber se temos mais memoria alocada
   que instancias, ou se liberamos memoria mais de uma vez. */

ivetor::ivetor ()
{
    nInstance++;
    n=0;
    nmax=0;
    bn=0;
    bnmax=0;
    tag=NULL;
    d=NULL;
}

ivetor::~ivetor ()
{
    nInstance--;
}

```

```

void initclassivector (void)
{
    nInstance=0;
    nAlloc=0;
}

int getivetornAlloc (void)
{
    return (nAlloc);
}

int getivetornInstance (void)
{
    return (nInstance);
}

ivector::create (int bi, int ni)
{
#ifdef _DEBUG_ENTRY_
    printf ("Criando %d+1 instancia.\n", nAlloc);
/* ja que tem que ser a primeira instrucao... */
#endif
    d=(dupla*) malloc (ni*sizeof (dupla));
#ifdef _VERIFY_ALLOC_
    if (!d) {
        printf ("Memoria insuficiente %d+1.\n", nAlloc);
        exit (0);
    }
#endif
    d--; /* sujeira para que o primeiro indice valido seja 1. */
    tag=(dupla **) malloc ((bi+1)*sizeof (dupla *));
#ifdef _VERIFY_ALLOC_
    if (!tag) {
        printf ("Memoria insuficiente %d+1.\n", nAlloc);
        exit (0);
    }
#endif
    tag--; /* sujeira para que o primeiro indice valido seja 1. */
    nAlloc++;
    bn=bi;
    bnmax=bi;
    n=ni;
    nmax=n;
#ifdef _DEBUG_END_
    printf ("%d instancia criada %p.\n", nAlloc, tag);
#endif
}

void ivector::destroy (void)
{
#ifdef _DEBUG_ENTRY_
    printf ("Removendo %d instancia %p.\n", nAlloc, tag);
#endif
    tag++;
#ifdef _DEBUG_ALLOC_
    printf ("Liberando vetor %p.\n", tag);
#endif
    free (tag);
    d++;
#ifdef _DEBUG_ALLOC_
    printf ("Liberando vetor %p.\n", d);
#endif
    free (d);
    nAlloc--;
    n=0;
    nmax=0;
    bn=0;
    bnmax=0;
    tag=NULL;
    d=NULL;
#ifdef _DEBUG_END_
    printf ("%d instancia removida.\n", nAlloc);
#endif
}

```



```

void ivetor::fprint (FILE *fp, const char *name)
{
    dupla *pd;
    int j;
    fprintf (fp, "%s=[\n", name);
    pd=d;
    pd++;
    for (j=1;j<n;j++, pd++) fprintf (fp, "%c%8d\t%8d\n", ((pd==tag[pd->b])?'T':' '), pd->b, pd->i);
    fprintf (fp, "%c%8d\t%8d ];\n", ((pd==tag[pd->b])?'T':' '), pd->b, pd->i);
}

char *ivetor::fread (FILE *fp, char *name)
{
    int c, in, nblocks;
    long pos;
    int fisnum, j;
    char *pname, *ic, iname[80];
    dupla *pdx;

    fgets (iname, 80, fp);
    ic=strchr (iname, '=');
    if (!ic) {
        printf ("iVetor %s size not found.\n", iname);
        exit (0);
    }
    *ic='\0';
    ic++;
    if (sscanf (ic, " %d ; ", &nblocks)!=1) {
        printf ("iVetor %s size not found.\n", iname);
        exit (0);
    }
    ic=strstr (iname, "size");
    if (!ic) {
        printf ("%s: bad iVetor size name.\n", iname);
        exit (0);
    }

    fisnum=0;
    in=0;
    pname=name;
    do {
        c=fgetc (fp);
        if (isalnum(c)) *pname++=c;
    } while ((!feof (fp)) && (c!='('));
    *pname='\0';
    pos=ftell (fp);
    while ((c!=']')&&(!feof(fp))) {
        c=fgetc(fp);
        switch (c) {
            case '\t':
            case '\n':
            case ',':
            case ']':
            case ';':
            case ' ': if (fisnum) in++; fisnum=0; break;
            default: if (isdigit (c) || (c=='.' ) || (c=='+' ) || (c=='-'))
                fisnum=1;
                else fisnum=0;
        }
    }
    if (fseek (fp, pos, SEEK_SET)) {
        printf ("Parse error.");
        exit (0);
    }
    if (in%2) {
        printf ("syntax error");
        exit (0);
    }
    in/=2;
    this->resize (nblocks, in);
    pdx=d; pdx++;
    for (j=1;j<in;j++, pdx++) {
        if (fscanf (fp, " %d %d", &(pdx->b), &(pdx->i))!=2) {
            printf ("syntax error.");
            exit (0);
        }
    }
}

```

```

    }
    if (fscanf (fp, "%d %d ]; ", &(pdx->b), &(pdx->i))!=2) {
        printf ("syntax error.");
        exit (0);
    }
    // tag=(dupla **) malloc ((pdx->b+1)*sizeof (dupla *));
    // bn=pdx->b+1;
    // bnmax=bn+1;
    // tag--;
    arrangetags();
    return (name);
}

void ivetor::resize (int nbn, int nn)
{
    dupla *newline; /* ponteiro para linha realocada. */
    dupla **newblock;
    int i;
#ifdef _DEBUG_ENTRY_
    printf ("Redimensionando vetor %p\n", tag);
#endif
    if (!tag) this->create (nbn, nn);
    if (nn>bnmax) {
#ifdef _DEBUG_ALLOC_
        printf ("Realocando vetor %p\n", tag);
#endif
        newline=(dupla *) realloc (d+1, nn*sizeof (dupla));
#ifdef _VERIFY_ALLOC_
        if (!newline) {
            printf ("Memoria insuficiente %d+1.\n", nAlloc);
            exit (0);
        }
#endif
        d=newline-1; /* sujeira para que o primeiro indice valido seja 1. */
        bnmax=nn;
    }
    n=nn;
    if (nbn>bnmax) {
        newblock=(dupla **) realloc (tag+1, (nn+1)*sizeof (dupla));
#ifdef _VERIFY_ALLOC_
        if (!newblock) {
            printf ("Memoria insuficiente %d+1.\n", nAlloc);
            exit (0);
        }
#endif
        tag=newblock-1; /* sujeira para que o primeiro indice valido seja 1. */
        for (i=bnmax+1;i<=nbn;i++) tag[i]=tag[bnmax];
        bnmax=nbn;
    }
    bn=nbn;

#ifdef _DEBUG_END_
    printf ("%p Realocado.\n", tag);
#endif
}

int ivetor::getblocksize (int bloco)
{
    return (tag[bloco+1]-tag[bloco]);
}

int ivetor::getsize (void)
{
    return (n);
}

int ivetor::getmaxsize (void)
{
    return (bnmax);
}

int ivetor::getblock (void)
{
    return (bn);
}

```

```

int ivetor::getmaxblock (void)
{
    return (bnmax);
}

void ivetor::copy (ivetor *a)
{
    dupla *av, **pav;
    dupla *bv, **pbv;
    this->resize (a->getblock(), a->getsize()); /* nao reproduz exatamente o
                                                vetor a, ja que ele pode
                                                ter tamanho maximo diferen-
                                                te do tamanho atual, mas
                                                isto pode funcionar ate
                                                como um simples garbage
                                                collector manual... */

    av=a->d+1;
    bv=this->d+1;
    memcpy (bv, av, n*sizeof(dupla));
    pav=a->tag+1;
    pbv=this->tag+1;
    arrangetags();
}

int ivetor::update1 (int bloco, int indice, int j)
/* retorna o indice sequencial do elemento que entra.
   o indice sequencial do elemento que sai e' j. */
{
    dupla *pj, *pi, *src, *dest;
    int len, k, r;
#ifdef _VERIFY_BOUNDS_
    if ((j<=0)|| (j>n)) {
        printf ("%p(%d) fora dos limites do vetor.\n", tag, j);
        exit (0);
    }
#endif
    len=0; src=NULL; dest=NULL;
    pi=tag[bloco]; r=pi-d;
    pj=&(d[j]);
    if (pj>pi) {
        src=pi;
        dest=pi+1;
        len=pj-pi;
        for (k=bloco+1; k<=pj->b; k++) tag[k]+=1;
//      if (bloco!=(pi+1)->b) tag[bloco]=pi;
    }
    else {
        if (pj<pi) {
            pi=pi-1;
            src=pj+1;
            dest=pj;
            len=pi-pj;
            for (k=pj->b+1; k<=bloco; k++) tag[k]-=1;
//      if (bloco!=(pi-1)->b) tag[bloco]=pi;
        }
        else tag[pj->b]=pj;
    }
    memmove (dest, src, len*sizeof(dupla));
    pi->b=bloco;
    pi->i=indice;
    return (r);
}

int ivetor::updaten (int bloco, int indice, int j)
/* retorna o indice sequencial do elemento que entra. */
{
    dupla *pj, *pi, *src, *dest;
    int len, k, r;
#ifdef _VERIFY_BOUNDS_
    if ((j<=0)|| (j>n)) {
        printf ("%p(%d) fora dos limites do vetor.\n", tag, j);
        exit (0);
    }
#endif
    len=0; src=NULL; dest=NULL;
    bloco++;
}

```

```

pi=tag[bloco]; r=pi-d;
pj=&(d[j]);
if (pj>pi) {
    src=pi;
    dest=pi+1;
    len=pj-pi;
    for (k=bloco;k<=pj->b;k++) tag[k]+=1;
}
else {
    if (pj<pi) {
        pi=pi-1;
        src=pj+1;
        dest=pj;
        len=pi-pj;
        for (k=pj->b+1;k<bloco;k++) tag[k]-=1;
    }
    else tag[pj->b]=pj+1;
}
memmove (dest, src, len*sizeof(dupla));
pi->b=bloco-1;
pi->i=indice;
return (r);
}

dupla &ivector::operator () (int i)
{
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>n)) {
        printf ("%p(%d) fora dos limites do vetor.\n", tag, i);
        exit (0);
    }
#endif
    return (d[i]);
}

void ivector::arrangetags (void)
{
    dupla *pd;
    int j, k, lastblock;
    pd=d+n;
    if (pd->b>bn) {
        puts ("bad tag allocation.");
        exit (0);
    }
    lastblock=pd->b+1;
    for (k=lastblock;k<=(bn+1);k++) tag[k]=d+n+1;
    for (j=0;j<n;j++, pd--) {
        if (pd->b!=lastblock) {
            for (k=pd->b+1;k<=lastblock;k++) tag[k]=pd+1;
            lastblock=pd->b;
        }
    }
    for (k=lastblock;k>0;k--) tag[k]=d+1;
}

int ivector::blockpos(int i)
{
    dupla *p, *t;
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>n)) {
        printf ("ivector %p(%d) acesso fora dos limites do vetor.", tag, i);
        exit (0);
    }
#endif
    p=&d[i];
    t=tag[d[i].b];
    return (p-t+1);
}

#ifdef _TEST_IVETOR_
void main (void)
{
    FILE *fpOut, *fpIn;
    int i, j;
    ivector b, c;
    vetor d;

```

```

char name[30];
b.create (10, 30);
d.create (30);
// for (i=1;i<=30;i++) {
//     b(i).i=i;
//     b(i).b=(i-1)/5;
//     b(i).b+=(((i-1)/5)>2)?2:1;
//     b(i).b+=2;
//     d(i)=(double) i+0.5;
// }
// b.arrangetags();

c.create (6, 20);
for (i=1;i<=20;i++) {
    c(i).i=i;
    c(i).b=i/4+1;
}
c.arrangetags();

fpIn=fopen ("t.in", "rt");
b.fread (fpIn, name);
fclose (fpIn);

fpOut=fopen ("ttt.txt", "wt");
b.fprint (fpOut, "ttt");
j=11;
i=b.update1 (4, 11, j);
d.update (1.22, i, j);
j=22;
i=b.updaten (1, 10, j);
d.update (2.34, i, j);
j=12;
i=b.updaten (9, 4, 12);
d.update (3.46, i, j);
// b.updaten (3, 20, 2);
// b.updaten (8, 5, 5);
// b.updaten (8, 6, 25);
b.fprint (fpOut, "b");
d.fprint (fpOut, "d");
fclose (fpOut);

b.destroy ();
d.destroy ();
c.destroy ();
}
#endif

```

SIMPLEX.CPP

```

#include <malloc.h>
#include "rbaf.hpp"
#include "ivetor.hpp"
#include "vetor.hpp"

void heap_dump(int ipassos);

int monta (rbaf *A, rbaf *B, vetor *c, vetor *cb, ivetor *b)
/* Monta B e cb selecionando de A e c os elementos listados em b.
*/
{
    int i, j, k,
        nblock, /* quantidade de blocos. */
        m, nd, nc, /* dimensoes dos blocos. */
        offsetc; /* offset do custo. */
    densa *D, *C, *AD, *AC;
    double **pddest, **pcdest;
    nblock=b->getblock();
    k=1;
    offsetc=0;
    for (i=1;i<=nblock;i++, offsetc+=AD->getsize (1)) {
        m=b->getblocksize (i);

        AD=A->getdblock(i);

```

```

        nd=AD->getsize (2);
        D=B->getdblock (i);
        D->resize (m, nd);
        pddest=D->getpline (1);

        AC=A->getcblock(i);
        nc=AC->getsize (2);
        C=B->getcblock (i);
        C->resize (m, nc);
        pcdest=C->getpline (1);

        for (j=1;j<=m;j++, pddest++, pcdest++, k++) {
            memcpy (*pddest+1, AD->getline ((*b)(k).i)+1, nd*sizeof (double));
            memcpy (*pcdest+1, AC->getline ((*b)(k).i)+1, nc*sizeof (double));
            (*cb)(k)=(*c)(offsetc+(*b)(k).i);
        }
    }
    return (0);
}

int linha (rbaf *A, int bloco, int indice, vetor *v)
/* em V a linha densa correspondente ao bloco "bloco" linha "indice" de A. */
{
    int i, ic, id, nv, nb;
    double *pvdest, *psrc;
    nb=A->getsize();
    nv=0;
    for (i=1;i<=nb;i++) {
        nv+=(A->getdblock(i))->getsize(2);
    }
    ic=nv+1;
    nv+=(A->getcblock(1))->getsize(2);
    id=1;
    for (i=1;i<bloco;i++) {
        id+=(A->getdblock(i))->getsize(2);
    }
    v->resize(nv);
    pvdest=v->getline()+1;
    for (i=1;i<=nv;i++, pvdest++) *pvdest=0.0;

    pvdest=v->getline();
    pvdest+=id;
    psrc=(A->getdblock(bloco))->getline(indice)+1;
    memcpy (pvdest, psrc, sizeof (double) * (A->getdblock(bloco))->getsize (2));

    pvdest=v->getline();
    pvdest+=ic;
    psrc=(A->getcblock(bloco))->getline(indice)+1;
    memcpy (pvdest, psrc, sizeof (double) * (A->getcblock(bloco))->getsize (2));
    return (0);
}

int alloc (rbaf *A, rbaf *B, vetor *cb, ivetor *b)
/* Aloca B e cb.*/
{
    int i,
        nblock, /* quantidade de blocos. */
        m, nd, nc; /* dimensoes dos blocos. */
    densa *AD, *AC;
    nblock=b->getblock();
    B->resize (nblock);
    cb->create(b->getsize());
    for (i=1;i<=nblock;i++) {
        m=b->getblocksize (i);

        AD=A->getdblock(i);
        nd=AD->getsize (2);
        B->createdblock (i, m, nd);

        AC=A->getcblock(i);
        nc=AC->getsize (2);
        B->createcblock (i, m, nc);
    }
    return (0);
}

```

```

void _check(int ipassos)
{
    heap_dump (ipassos);
    switch( _heapchk() ) {
        case _HEAPOK:
            printf( "OK - heap is good\n" );
            break;
        case _HEAPEMPTY:
            printf( "OK - heap is empty\n" );
            break;
        case _HEAPBADBEGIN:
            printf( "ERROR - heap is damaged. %d\n", ipassos );
            exit (0);
            break;
        case _HEAPBADNODE:
            printf( "ERROR - bad node in heap. %d\n", ipassos );
            exit (0);
            break;
    }
}

void heap_dump(int ipassos)
{
    struct _heapinfo h_info;
    int heap_status;

    h_info._pentry = NULL;
    for(;;) {
        heap_status = _heapwalk( &h_info );
        if( heap_status != _HEAPOK ) break;
        printf( " %s block at %Fp of size %4.4X\n",
            (h_info._useflag == _USEDENTRY ? "USED" :
"FREE"),
            h_info._pentry, h_info._size );
    }

    switch( heap_status ) {
        case _HEAPEND:
            printf( "OK - end of heap\n" );
            break;
        case _HEAPEMPTY:
            printf( "OK - heap is empty\n" );
            break;
        case _HEAPBADBEGIN:
            printf( "ERROR - heap is damaged. %d\n", ipassos );
            exit (0);
            break;
        case _HEAPBADPTR:
            printf( "ERROR - bad pointer to heap. %d\n", ipassos );
            exit (0);
            break;
        case _HEAPBADNODE:
            printf( "ERROR - bad node in heap. %d\n", ipassos );
            exit (0);
    }
}

int min (vetor *dtil, vetor *rtilj)
{
    int i, n, imin;
    double *pdtil, *prtilj, min;
    imin=0; min=1e308;
    n=dtil->getsize();
    pdtil=dtil->getline()+1;
    prtilj=rtilj->getline()+1;
    for (i=1;i<=n;i++, pdtil++, prtilj++) {
        if (*prtilj>0.0) {
            if ((*pdtil/(*prtilj))<min) {
                min=(*pdtil/(*prtilj));
                imin=i;
            }
        }
    }
    return (imin);
}

```

```

void swap (rbaf *B, int ib, int tiraB, rbaf *R, int jb, int tiraR,
           vetor *jd, vetor *jc, vetor *id, vetor *ic)
/* remove de B a linha correspondente a tiraB do bloco ib e faz uma copia em id..ic;
   remove de R a linha correspondente a tiraR do bloco jb e faz uma copia em jd..jc;
   insere em B, como primeira linha do bloco jb a linha tirada de R;
   insere em R, como ultima linha do bloco ib a linha tirada de B.
*/
{
    double *bd, *bc,
           *rd, *rc,
           *aux,      /* para copia dos vetores. */
           **pp;
    densa *D, *C;
    int nd, nc, m, i;

/* Tira da base. */
    D=B->getdblock (ib);
    nd=D->getsize(2);
    m=D->getsize(1);
    pp=D->getpline(tiraB);
    bd=*pp;
    for (i=tiraB;i<m;i++, pp++) *pp=*(pp+1);
    *pp= ((double*) malloc (nd*sizeof(double)))-1;
    D->resize (m-1, nd);

    C=B->getcblock (ib);
    nc=C->getsize(2);
    pp=C->getpline(tiraB);
    bc=*pp;
    for (i=tiraB;i<m;i++, pp++) *pp=*(pp+1);
    *pp= ((double*) malloc (nc*sizeof(double)))-1;
    C->resize (m-1, nc);
/* Faz copias: id=bd; ic=bc */
    id->resize (nd);
    aux=id->getline();
    memcpy (aux+1, bd+1, nd*sizeof(double));

    ic->resize (nc);
    aux=ic->getline();
    memcpy (aux+1, bc+1, nc*sizeof(double));
/* Tira de R. */
    D=R->getdblock (jb);
    nd=D->getsize(2);
    m=D->getsize(1);
    pp=D->getpline(tiraR);
    rd=*pp;
    for (i=tiraR;i<m;i++, pp++) *pp=*(pp+1);
    *pp= ((double*) malloc (nd*sizeof(double)))-1;
    D->resize (m-1, nd);

    C=R->getcblock (jb);
    nc=C->getsize(2);
    pp=C->getpline(tiraR);
    rc=*pp;
    for (i=tiraR;i<m;i++, pp++) *pp=*(pp+1);
    *pp= ((double*) malloc (nc*sizeof(double)))-1;
    C->resize (m-1, nc);
/* Faz copias: jd=rd; jc=rc */
    jd->resize (nd);
    aux=jd->getline();
    memcpy (aux+1, rd+1, nd*sizeof(double));

    jc->resize (nc);
    aux=jc->getline();
    memcpy (aux+1, rc+1, nc*sizeof(double));
/* Poe em B */
    D=B->getdblock (jb);
    nd=D->getsize(2);
    m=D->getsize(1);
    D->resize(m+1, nd);
    D->shdown();
    pp=D->getpline(1);
    free (*pp+1);
    *pp=rd;

```



```

C=B->getcblock (jb);
nc=C->getsize(2);
m=C->getsize(1);
C->resize(m+1, nc);
C->shdown();
pp=C->getpline(1);
free (*pp+1);
*pp=rc;
/* Poe em R */
D=R->getdblock (ib);
nd=D->getsize(2);
m=D->getsize(1);
D->resize(m+1, nd);
pp=D->getpline(m+1);
free (*pp+1);
*pp=bd;

C=R->getcblock (ib);
nc=C->getsize(2);
m=C->getsize(1);
C->resize(m+1, nc);
pp=C->getpline(m+1);
free (*pp+1);
*pp=bc;
}

void main (void)
{
    FILE *fpIn, *fpOut;
    char name[80];
    rbaf A, B, R, U;
    ivetor b, r;
    vetor c, d, cb, cr, z, dtil, rtilj;
    vetor jd, jc, id, ic;
    int i, ib, ii,
        j, jb, ji,
        indice;
    int npassos, ipassos;
    double vcb, vcr;
    fpIn=fopen ("teste401.m", "rt");
    if (!fpIn) {
        perror ("teste401.m");
        exit (0);
    }
    A.fread (fpIn, name);
    b.fread (fpIn, name);
    r.fread (fpIn, name);
    c.fread (fpIn, name);
    d.fread (fpIn, name);
    fclose (fpIn);

    /* aqui eu sei o tamanho do problema!! */
    alloc (&A, &B, &cb, &b);
    alloc (&A, &R, &cr, &r);
    monta (&A, &B, &c, &cb, &b);
    monta (&A, &R, &c, &cr, &r);
    U.copy (&B);
    U.QR();

    jc.create(10);
    jd.create(10);
    ic.create(10);
    id.create(10);

    npassos=50;
    ipassos=0;
    while (npassos>ipassos) {
        ipassos++;

        //      monta (&A, &B, &c, &cb, &b);
        //      monta (&A, &R, &c, &cr, &r);

        fpOut=fopen ("base.out", "at");
        if (!fpOut) {
            perror ("base.out");
            exit (0);

```

```

    }
    fprintf (fpOut, "passo=%d\n", ipassos);
    b.fprint (fpOut, "b");
    r.fprint (fpOut, "r");
    //      B.fprint (fpOut, "B");
    //      R.fprint (fpOut, "R");
    fclose (fpOut);

    //      U.copy (&B);
    //      U.QR();
    z.copy (&cb);
    B.etmx(&z);
    U.utix(&z);
    U.uix(&z);
    R.emx(&z);
    z.minus (&cr);
    j=z.max();

    //      j=z.first();
    jb=r(j).b;
    ji=r(j).i;
    vcr=cr(j);

    linha (&A, jb, ji, &rtilj);
    dtil.copy (&d);
    U.utix (&dtil);
    U.uix (&dtil);
    B.emx (&dtil);

    if (z(j)<1e-12) {
        fpOut=fopen ("base.out", "at");
        if (!fpOut) {
            perror ("base.out");
            exit (0);
        }
        fprintf (fpOut, "solucao otima\n");
        b.fprint (fpOut, "b");
        dtil.fprint (fpOut, "xb");
        fclose (fpOut);
        break;
    }

    U.utix (&rtilj);
    U.uix (&rtilj);
    B.emx (&rtilj);

    i=min(&dtil, &rtilj);

    fpOut=fopen ("base.out", "at");
    if (!fpOut) {
        perror ("base.out");
        exit (0);
    }
    fprintf (fpOut, "j=%d\n", j);
    fprintf (fpOut, "i=%d\n", i);
    fclose (fpOut);

    if (i==0) {
        fpOut=fopen ("base.out", "at");
        if (!fpOut) {
            perror ("base.out");
            exit (0);
        }
        fprintf (fpOut, "ppl ilimitado\n");
        b.fprint (fpOut, "b");
        dtil.fprint (fpOut, "xb");
        fclose (fpOut);
        break;
    }
    ib=b(i).b;
    ii=b(i).i;
    vcb=cb(i);

    swap (&B, ib, b.blockpos(i), &R, jb, r.blockpos (j),
        &jd, &jc, &id, &ic);

```

```

        indice=b.update1 (jb, ji, i);
        cb.update (vcr, indice, i);
        indice=r.updaten (ib, ii, j);
        cr.update (vcb, indice, j);
        U.upo (ib, &id, &ic);
        U.upi (jb, &jd, &jc);
    }
    A.destroy();
    B.destroy();
    R.destroy();
    c.destroy();
    d.destroy();
    cb.destroy();
    cr.destroy();
    z.destroy();
    rtilj.destroy();
    dtil.destroy();
    _check(ipassos);
}

```

Código MATLAB

DRIVE.M

```

clear all
global V1 V2 V3 W1 W2 W3 S ...
        B1 B2 B3 C1 C2 C3 ...
        RD1 RD2 RD3 RC1 RC2 RC3 ...
        AD1 AD2 AD3 AC1 AC2 AC3 ...
        c1 c2 c3 cr cb ...
        bb bi rb ri c d ...
        sm sn sd kb pm pn;

teste
montabr (bb, bi, rb, ri);
sn=0; sm=0;pm=0; pn=0;
for k=1:kb
    kstr= int2str(k);
    eval ([ 'mk,nk]=size(B' kstr ');'])
    sm=sm+mk; sn=sn+nk;
    eval ([ 'mk,nk]=size(AD' kstr ');'])
    pm=pm+mk;
    pn=pn+nk;
end

pm=pm+size(AC1, 1);

sd=sm-sn;
sqrbafe
seu

```

SIMPLEX.M

```

%function [Vo,b,x,r] = simplex(criterio,trace)

global AD1 AD2 AD3 AC1 AC2 AC3;
global B1 B2 B3 C1 C2 C3;
global V1 V2 V3 W1 W2 W3 S;
global RD1 RD2 RD3 RC1 RC2 RC3;
global bb bi rb ri c d;
global cb cr;
global sm sn sd kb pm pn;

criterio=1;
trace=0;

    acabou = 0;
    while ~acabou
        % montabr (bb, bi, rb, ri);
        % sqrbafe;

        x=cb';
        etmx;
        x=y;
    end

```

Parte V

Código C do *solver* para problemas na
forma angular bloqueada aninhada

```
/* GLOBAL.HPP Nao existe .CPP correspondente. Aqui vao as constantes comuns a todos os
modulos.
Fabio Nakano - 05.05.1997 */
#ifndef __GLOBAL_HPP__
#define __GLOBAL_HPP__
#define OK 0
#define T_MATRIX 1 /* Matriz de doubles */
#define T_CELLARRAY 2 /* Matriz de matrizes */
#define EPS 1e-10
#endif
```

```

#ifndef __DENSE_HPP__
#define __DENSE_HPP__

#include <stdio.h>

/*#define _DEBUG_ENTRY_      Manda uma mensagem como primeira
                             instrucao da funcao. Nem todas as funcoes tem
                             este header pois algumas são tao simples que
                             a mensagem ao inves de ajuda, apenas polui.*/
#define _DEBUG_OUT_         Manda uma mensagem antes de retornar. */
#define _DEBUG_ALLOC_       Manda mensagens quando alocar/desalocar memoria. */
#define _VERIFY_ALLOC_      Verifica todas as alocações, sem esta macro
                             confiamos que o SO "nunca falhe". */
#define _VERIFY_BOUNDS_     /* Verifica os bounds da matriz no operador de acesso
                             e nas funcoes de acesso direto. */

class densa {
    int m, n,          /* Tamanho corrente. */
        mmax, nmax;    /* Tamanho maximo -
                        que corresponde ao tamanho alocado. */

    double **M;

public:
    friend void initclassdensa (void);
    /* Inicializa variaveis estaticas da classe. */
    friend int getnAlloc (void);
    friend int getnInstance (void);
    densa ();
    /* Construtor da classe - zera dimensoes e faz M apontar para NULL. */
    ~densa ();
    create (int mi, int ni);
    /* Aloca espaco para a matriz, e inicializa as dimensoes.
    Inicialmente a dimensao e' a maxima. */
    void destroy (void);
    /* Remove a matriz da memoria e zera as dimensoes. */
    double &operator () (int i, int j);
    /* Operador de acesso a elemento por referencia.
    Permite acesso transparente aos elementos da matriz. */
    double *getline (int i);
    /* Funcao de acesso direto a elementos da linha -
    para inicializar elementos/combinar linearmente linhas
    sem a necessidade (overhead) do operador de acesso. */
    double **gettpline (int i);
    /* Funcao de acesso aa cabeca da linha para fazer substituicao de
    linhas em tempo constante. */

    /* As funcoes de acesso direto (get*), enquanto publicas, subvertem
    o paradigma de orientacao a objeto, mas durante o desenvolvimento
    vou mante-las assim. */
    void fprint (FILE *fp, const char *name);
    /* Imprime, com o nome <name>, em formato texto, a matriz em fp. */
    char *fscan (FILE *fp, char *name);
    /* Le, em formato texto, a matriz em fp. E retorna o nome lido em name*/

    void resize (int mm, int nn);
    /* Redimensiona a matriz. Se desejarmos torna-la maior que suas dimensoes
    maximas, ocorre realocacao (usando realloc de C), senao, se diminuirmos o
    numero de linhas, as linhas sao liberadas e se diminuirmos o numero de colunas,
    a realocacao nao e' feita. O motivo e': nesta representacao inserir linha
    em uma matriz (mxn) custa O(1) tempo e gera O(n) memoria, enquanto que para coluna,
    custa O(m) tempo e gera O(m) memoria, como o caso esperado e' m=n (matriz quadrada),
    e' muito mais caro realocar colunas. */
    int getsize (int d);
    /* Retorna a dimensao corrente da matriz. d=1 retorna linhas, d=2 retorna colunas
    se retornar -1 indica erro. */
    int getmaxsize (int d);
    /* Retorna a dimensao maxima da matriz. d=1 retorna linhas, d=2 retorna colunas
    se retornar -1 indica erro. */

```

```

int shdown (void);
/* roda para baixo as linhas da matriz :
se a seq. de linhas era 1, 2, ... m passa a ser m, 1, 2, ..., m-1. */
int shup (void);
/* roda para cima as linhas da matriz :
se a seq. de linhas era 1, 2, ... m passa a ser 2, ..., m, 1. */
void matprt (FILE *fp, const char *name);
/* Escreve a matriz em formato binario MATLAB nivel 1.0 O arquivo tem que
ser aberto em modo binario. */
void copy (densa *A);
/* copia matriz. */
void cellwr (FILE *fp, const char *name);
/* escreve matriz em formato binario. */
int cellrd (FILE *fp, char *name);
/* le matriz em formato binario. */
/* Convencao de nomes: funcoes todas em letras minusculas. */
int moverows (densa *src, int i0);
/* move as linhas do final de src, a partir da i0-esima
para this, corrige o numero de linhas das matrizes,
mas o comprimento maximo das linhas tem que ser
igual. */
};
#endif

```

```

#ifndef __LINEAR_HPP__
#define __LINEAR_HPP__

#include "vetor.hpp"
#include "densa.hpp"

int vim (densa *V, vetor *x);
/* retorna em x o resultado de  $y = \text{inv}(V)x$ , supoe V triangular superior. */
int vtim (densa *V, vetor *x);
/* retorna em x o resultado de  $y = \text{inv}(V')x$ , supoe V triangular superior. */
void vtam (densa *V, vetor *a);
/* retorna em a o resultado de  $y = V'a$ . */
void vam (densa *V, vetor *a);
/* retorna em a o resultado de  $y = V*a$  */
#endif

```



```

#ifndef __QR_HPP__
#define __QR_HPP__

#include "densa.hpp"
#include "vetor.hpp"

void vtm (densa *V, vetor *d);
/* retorna em d o resultado de V*d */
void vm (densa *V, vetor *d);
/* retorna em d o resultado de Vd */
int QR (densa *V, densa *C, densa *E);
/* Faz a fatoracao QR de V, que passa a ser triangular superior e executa as
mesmas rotacoes em C e E. */
int QRh (densa *V, densa *C, densa *E);
/* Faz a fatoracao QR de V, que deve ser de Hessenberg e volta a ser triangular
superior e executa as mesmas rotacoes em C e E. */
double QRx (vetor *x, densa *V, densa *C, densa *E);
/* Faz as rotacoes para levar x a ||x||*I(:,1), faz as mesmas rotacoes em V, C e E
Retorna ||x|| */
void givcs (double *gc, double *gs, double x1, double x2);
/* Determina o angulo de rotacao (descrito pelo seu cosseno e seu seno) tq o
vetor (x1,x2) e transformado em (||(x1,x2)||, 0) */
void givrt (double **v1, double **v2, int n, double gc, double gs);
/* Faz a rotacao entre v1 e v2 pelo angulo definido por gc e gs. */

#endif

```

```

#ifndef __XRBAF_H__
#define __XRBAF_H__
#include "densa.hpp"
#include "vetor.hpp"

// #define _TESTE_XRBAF_

#define N_DEFAULT 5          /* Quantidade de blocos default para leitura. */

void initclassrbaf (void);
int getxbafInstance(void);
int getxbafAlloc(void);

class xrbaf {
    densa **D,          /* diagonal */
    **C,                /* acoplamento interno */
    **E;                /* acoplamento externo */
    int n,               /* numero de blocos em cada vetor */
    nmax;                /* maximo numero de blocos possivel sem realocacao. */
public:
    xrbaf ();
    /* Construtor: inicializa variaveis e incrementa contador de instancias. */
    ~xrbaf ();
    /* Destrutor: decrementa contador de instancias. */
    create (int ni);
    /* Aloca os vetores de ni ponteiros para os blocos diagonais e blocos de
    conexao. */
    void createdblock (int i, int m, int n);
    /* Cria o i-esimo bloco diagonal de tamanho mxn */
    void createcblock (int i, int m, int n);
    /* Cria o i-esimo bloco de acoplamento interno de tamanho mxn */
    void createeblock (int i, int m, int n);
    /* Cria o i-esimo bloco de acoplamento externo de tamanho mxn */
    void destroy ();
    /* Libera todos os blocos e vetores */
    densa **getpdblock (int i);
    /* Retorna ponteiro para o ponteiro do i-esimo bloco diagonal. */
    densa *getdblock (int i);
    /* Retorna ponteiro para o i-esimo bloco diagonal. */
    densa **getpcblock (int i);
    /* Retorna ponteiro para o ponteiro do i-esimo bloco de acoplamento interno. */
    densa *getcblock (int i);
    /* Retorna ponteiro para o i-esimo bloco de acoplamento interno. */
    densa **getpeblock (int i);
    /* Retorna ponteiro para o ponteiro do i-esimo bloco acoplamento externo. */
    densa *geteblock (int i);
    /* Retorna ponteiro para o i-esimo bloco acoplamento externo. */

    void cellwr (FILE *fp, const char *name);
    /* Escreve xrbaf em formato binario */
    int cellrd (FILE *fp, char *name);
    /* Le xrbaf em formato binario. */

    void resize (int nn);
    /* Redimensiona os vetores de ponteiros para blocos diagonais e
    blocos de acoplamento. Se a matriz encolher, apaga os blocos em excesso. */
    int getsize (void);
    /* Retorna a quantidade atual de blocos
    (comprimento dos vetores de ponteiros para blocos) */
    int getmaxsize (void);
    /* Retorna a quantidade maxima de blocos alocaveis sem realocacao dos vetores
    (comprimento alocado dos vetores de ponteiros para blocos) */

    void cellwr (FILE *fp, const char *name);
    int cellrd (FILE *fp, char *name);

    void copy (xrbaf *A);

};

#endif

```



```

#ifndef __NRBAF_H__
#define __NRBAF_H__
#include "xrbaf.hpp"
#include "vetor.hpp"

// #define _TESTE_NRBAF_

void initclassrbaf (void);

int getnrbaInstance(void);
int getnrbaAlloc(void);

class nrba {
    xrbaf **X;
    int n,          /* numero de blocos em cada vetor */
        nmax;      /* maximo numero de blocos possivel sem realocacao. */
public:
    nrba ();
    /* Construtor: inicializa variaveis e incrementa contador de instancias. */
    ~nrba ();
    /* Destrutor: decrementa contador de instancias. */
    create (int ni);
    /* Cria o vetor e inicializa as variaveis de acordo. */
    void createblock (int i, int nx);
    /* Cria bloco xrbaf */
    void destroy ();
    /* Libera todos os blocos e vetores */
    xrbaf **getpbblock (int i);
    /* Retorna ponteiro para o ponteiro do i-esimo bloco. */
    xrbaf *getblock (int i);
    /* Retorna ponteiro para o i-esimo bloco. */

    void cellwr (FILE *fp, const char *name);
    /* Escreve nrba em formato binario */
    int cellrd (FILE *fp, char *name);
    /* Le nrba em formato binario. */

    void resize (int nn);
    /* Redimensiona o vetor de ponteiros
       Se a matriz encolher, apaga os blocos em excesso. */
    int getsize (void);
    /* Retorna a quantidade atual de blocos
       (comprimento dos vetores de ponteiros para blocos) */
    int getmaxsize (void);
    /* Retorna a quantidade maxima de blocos alocaveis sem realocacao dos vetores
       (comprimento alocado dos vetores de ponteiros para blocos) */
    int getxsize (int i);
    void xresize (int i, int nn);
    densa *getdblock (int i, int j);
    densa *getcblock (int i, int j);
    densa *geteblock (int i, int j);

    void cellwr (FILE *fp, const char *name);
    int cellrd (FILE *fp, char *name);

    void copy (nrba *A);
    int QR ();
    int upi (int ke, int ki, vetor *qd, vetor *qc, vetor *qe);
    void uix (vetor *x);
    void utix (vetor *x);
    void uitet(int ke, int ki, vetor *bdi, vetor *bci, vetor *bce);
    void emx(vetor *x);
    void etmx(vetor *x);
    double upo(int he, int hi, vetor *bdi, vetor *bci, vetor *bce);
    int getsize (int dim);
};

```

#endif

```

#ifndef __VETOR_HPP__
#define __VETOR_HPP__

#include <stdio.h>

/*
#define _DEBUG_ENTRY_      Manda uma mensagem como primeira
                           instrucao da funcao. Nem todas as funcoes tem
                           este header pois algumas sao tao simples que
                           a mensagem ao inves de ajuda, apenas polui.*/
#define _DEBUG_END_        Manda uma mensagem antes de retornar. */
#define _DEBUG_ALLOC_      Manda mensagens quando alocar/desalocar memoria. */
#define _VERIFY_ALLOC_     Verifica todas as alocacoes, sem esta macro,
                           confiamos que o SO "nunca falhe". */
#define _VERIFY_BOUNDS_    /* Verifica os bounds da matriz no operador de acesso
                           e nas funcoes de acesso direto. */

class vetor {
    int n,          /* Tamanho corrente. */
        nmax;       /* Tamanho maximo - que corresponde ao tamanho alocado. */
    double *V;
public:
    friend void initclassvetor (void);
    /* Inicializa variaveis estaticas da classe. */
    friend int getvetorAlloc (void);
    friend int getvetorInstance (void);
    vetor ();
    /* Construtor da classe - zera dimensoes e faz V apontar para NULL. */
    ~vetor ();
    create (int ni);
    /* Aloca espaco para o vetor, e inicializa as dimensoes.
    Inicialmente a dimensao e' a maxima. */
    void destroy (void);
    /* Remove o vetor da memoria e zera as dimensoes. */
    double &operator () (int i);
    /* Operador de acesso a elemento por referencia.
    Permite acesso transparente aos elementos do vetor. */

    void fprintf (FILE *fp, const char *name);
    /* Imprime, com o nome <name>, em formato texto, o vetor em fp. */
    char *fread (FILE *fp, char *name);
    /* Le, em formato texto, o vetor em fp. E retorna o nome lido em name*/

    double *getline (void);
    /* retorna ponteiro para o array interno. */
    double **getpline (void);
    /* retorna ponteiro para o ponteiro do array interno. */

    void resize (int ni);
    /* Se o vetor tem tamanho maximo maior que ni, apenas atualiza n, senao,
    realoca o vetor. */
    int getsize (void);
    /* Retorna a dimensao corrente do vetor. Se retornar -1 indica erro. */
    int getmaxsize (void);
    /* Retorna a dimensao maxima do vetor. Se retornar -1 indica erro. */
    void copy (vetor *a);
    /* funcao de copia de vetor - conheco o operador de passagem por referencia em C++
    mas para as partes fundamentais, quero usar C++ em doses muito pequenas. */
    void matprt (FILE *fp, const char *name);
    /* imprime o vetor no arquivo em formato binario MATLAB nivel 1.0. */
    int update (double val, int i, int j);
    /* Elimina o elemento j e insere val na posicao i. */
    /* Convencao de nomes: funcoes todas em letras minusculas. */
    void setd (double *d, int ni) {V=d; n=ni; nmax=ni; }
    void minus (vetor *v);
    /* subtracao de vetores: this-=v; */
    double prod (vetor *v);
    /* produto entre vetores: d=this*v' */

```

```

void div (vetor *v);
/* divisao elemento a elemento: this=this./v */
int max (void);
/* retorna o indice do maior elemento no vetor. */
int min (void);
/* retorna o indice do menor elemento no vetor. */
int first (void);
/* Retorna o indice do primeiro elemento positivo. */
void cellwr (FILE *fp, const char *name);
int cellrd (FILE *fp, char *name);
/* Le e escreve vetores em formato binario. */
void cdot (double c);
/* Produto por constante: this*=c. */
void cplus (double c);
/* Soma com uma constante: this+=c. */
void fill (double a);
/* Inicializa os elementos do vetor com o valor a. */
void dot (vetor *v);
/* Produto elemento a elemento dos vetores: this=this.*v */
void plus (vetor *v);
/* Soma de vetores: this+=this. */
void vcat (vetor *v);
/* concatena vetores this=[this, v] */
double vnorm2 (void);
/* norma 2 do vetor */
};
#endif

```

```

/* Classe vetor de indices para matrizes na forma NRBAF
"
    Fabio Nakano - 10/02/99
"
    */
"

#ifndef __IVETOR_HPP__
#define __IVETOR_HPP__
"

#include <stdio.h>
"
#include <stdlib.h>
"
#include <string.h>
"
#include <ctype.h>
"

#include "vvetor.hpp"
"
#include "nrbafe.hpp"

#define _VERIFY_BOUNDS_
// #define _VERIFY_ALLOC_
// #define _DEBUG_ENTRY_
// #define _DEBUG_END_

typedef struct {
    int be,          /* bloco externo */
        bi,          /* bloco interno */
        i;           /* indice */
} tripla;

class ivetor {
    int n,           /* comprimento corrente */
        nmax;        /* comprimento alocado */
    vvvetor xtag,    /* tag externo */
        tag;         /* tag interno */
    tripla *t;       /* vetor de triplas */
public:
    friend void initclassivetor (void);
    /* Inicializa variaveis estaticas da classe. */
    friend int getivetornAlloc (void);
    friend int getivetornInstance (void);
    ivetor ();
    /* Construtor da classe - zera dimensoes e faz V apontar para NULL. */
    ~ivetor ();
    create (nrbafe *A);
    /* Aloca espaco para o vetor, e inicializa as dimensoes.
    Inicialmente a dimensao e' a maxima. Supoe que a estrutura de A seja
    completa para o problema ie: A contem todos os blocos que compoe o
    problema (poliedro) */
    void destroy (void);
    /* Remove o vetor da memoria e zera as dimensoes. */
    int getblocksize (int be, int bi);
    /* Retorna a quantidade de elementos no bloco. */
    int getsize (void);
    int getmaxsize (void);
    int updatel (int be, int bi, int indice, int j);
    /* elimina o j-esimo elemento do vetor (na ordem sequencial) e insere
    (bloco,indice) na primeira linha do bloco. */
    int updaten (int be, int bi, int indice, int j);
    /* elimina o j-esimo elemento do vetor (na ordem sequencial) e insere
    (bloco,indice) na ultima linha do bloco. */
    void arrangetags (void);
    /* arruma os tags. */
    int blockpos(int i);

```



```

/* devolve a posicao do elemento de indice sequencial i dentro de seu bloco. */
/* Convencao de nomes: funcoes todas em letras minusculas. */
void txttrd (FILE *fp);
void txtwr (FILE *fp);
int &operator () (int i, int j, int k);
/* retorna o indice armazenado na k-esima posicao do j-esimo bloco interno
do i-esimo bloco externo do vetor . */
tripla &operator () (int i);
};
#endif

```

```

/* classe vveter: Vetor de void */
"
    Preciso dessa classe para fazer os tags da classe ivetor,
"
    que agora sao dois.
"
    */
"

#include <stdio.h>
#include <stdlib.h>

/*
#define _VERIFY_BOUNDS_
#define _VERIFY_ALLOC_
#define _DEBUG_ENTRY_
#define _DEBUG_END_
*/

#ifdef __VVETOR_HPP__
#define __VVETOR_HPP__

class vveter {
public:
    void **V;
    int n, nmax;
    vveter () {V=NULL, n=0; nmax=0; };
    void resize (int nn);
    void create (int nn);
    void destroy(void);
    void * &operator () (int i);
    int getsize () { return (n); };
};

#endif

```

```

/*DENSEA.CPP*/
#include <stdio.h>
"
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include "global.hpp"
#include "densea.hpp"

/* states of the file reader machine. */
#define INIT      0
#define NUMBER    1
#define PUNCT     2
#define SPACE     3
#define NEWLINE   4

static int nAlloc=0, /* Contador de instancias alocadas. */
nInstance=0; /* Numero de instancias criadas. */
/* A ideia com os contadores acima e' saber se temos mais memoria alocada
que instancias, ou se liberamos memoria mais de uma vez. */

densea::densea ()
{
    nInstance++;
    m=0;
    n=0;
    mmax=0;
    nmax=0;
    M=NULL;
}

densea::~densea ()
{
    nInstance--;
}

void initclassdensea (void)
{
    nInstance=0;
    nAlloc=0;
}

int getnAlloc (void)
{
    return (nAlloc);
}

int getnInstance (void)
{
    return (nInstance);
}

densea::create (int mi, int ni)
{
    int i;
    double **linha; /* ponteiro para uma linha. */
#ifdef _DEBUG_ENTRY_
    printf ("Criando %d+1 instancia.\n", nAlloc);
/* ja que tem que ser a primeira instrucao... */
#endif

    /* Aloca vetor de ponteiros de linha. */
#ifdef _DEBUG_ALLOC_
    printf ("Alocando vetor de ponteiros da %d+1 instancia.\n", nAlloc);
#endif
    M=(double **) malloc ((mi+1)*sizeof (double *));
#ifdef _VERIFY_ALLOC_
    if (!M) {
        printf ("Memoria insuficiente %d+1.\n", nAlloc);
        exit (0);
    }
}

```

```

#endif
    linha=M;
    *linha=NULL;
    linha++;
    for (i=1;i<=mi;i++, linha++) {
        (*linha)=(double*) malloc (ni*sizeof (double));
#ifdef _VERIFY_ALLOC_
        if (!(*linha)&&ni) {
            printf ("Memoria insuficiente %d+1.\n", nAlloc);
            exit (0);
        }
#endif
#ifdef _DEBUG_ALLOC_
        printf ("%d-esimo vetor %p da matriz %p alocado.\n", i, *linha, M);
#endif
        (*linha)--;
        /* sujeira para que o primeiro indice valido seja 1. */
    }
    nAlloc++;
    m=mi;
    n=ni;
    mmax=m;
    nmax=n;
#ifdef _DEBUG_END_
    printf ("%d instancia criada %p.\n", nAlloc, M);
/* ja que tem que ser a primeira instrucao... */
#endif
}

void densa::destroy (void)
{
    int i;
    double **linha;
#ifdef _DEBUG_ENTRY_
    printf ("Removendo %d instancia %p.\n", nAlloc, M);
#endif
    linha=M;
    linha++;
    for (i=1;i<=m;i++, linha++) {
        (*linha)++;
#ifdef _DEBUG_ALLOC_
        printf ("Liberando %d-esima linha %p da matriz %p.\n", i, *linha, M);
#endif
    }
    free (*linha);
    free (M);
    nAlloc--;
    m=0;
    n=0;
    mmax=0;
    nmax=0;
    M=NULL;
#ifdef _DEBUG_END_
    printf ("%d instancia removida.\n", nAlloc);
/* ja que tem que ser a primeira instrucao... */
#endif
}

double &densa::operator () (int i, int j)
{
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)||i>m) {
        printf ("%p(%d,%d) fora dos limites da matriz.\n", M, i, j);
        exit (0);
    }
    if ((j<=0)||j>n) {
        printf ("%p(%d,%d) fora dos limites da matriz.\n", M, i, j);
        exit (0);
    }
#endif
    return (M[i][j]);
}

```

```

double *densa::getline (int i)
{
#ifdef _DEBUG_ENTRY_
    printf ("Retornando M(%d, :).%p\n", i, M);
#endif
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>m)) {
        printf ("%p(%d,:) fora dos limites da matriz.\n", M, i);
        exit (0);
    }
#endif
    return (M[i]);
}

double **densa::getpline (int i)
{
#ifdef _DEBUG_ENTRY_
    printf ("Retornando &M(%d, :).%p\n", i, M);
#endif
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>m)) {
        printf ("%p(%d,:) fora dos limites da matriz.\n", M, i);
        exit (0);
    }
#endif
    return (&(M[i]));
}

void densa::fprintf (FILE *fp, const char *name)
{
    double *p;
    int i, j;
    fprintf (fp, "%s=\n", name);
    for (i=1;i<=m;i++) {
        p=M[i];
        p++;
        for (j=1;j<n;j++, p++) fprintf (fp, "%9le\t", *p);
        fprintf (fp, "%9le\n", *p);
    }
    fputs ("];\n", fp);
}

char *densa::fscan (FILE *fp, char *name)
{
    int c, im, in;
    long pos;
    int estado, i, j;
    char *pname;
    double *linha;
    im=0;
    in=0;
    estado=INIT;
    pname=name;
    do {
        c=fgetc (fp);
        if (isalnum(c)) *pname++=c;
    } while ((!feof (fp)) && (c!='('));
    *pname='\0';
    pos=ftell (fp);
    while ((c!=']')&&(!feof(fp))) {
        c=fgetc(fp);
        switch (c) {
            case ',':
            case '\t':
            case ' ': if (estado==NUMBER) in++; break;
            case ']':
            case ';': if (estado==NEWLINE) break;
            case '\n': if (estado!=INIT) im++; if (estado==NUMBER) in++; break;
        }
        switch (c) {
            case '\n': if (estado!=INIT) estado=NEWLINE; break;
            case '\t':
            case ' ': if (estado!=INIT) estado=SPACE; break;
        }
    }
}

```

```

        case ']':
        case ',':
        case ';': estado=PUNCT; break;
        default: if (isdigit (c) || (c=='e') || (c=='E') || (c=='.')) || (c=='+') || (c=='-'))
            estado=NUMBER;
        else {
            printf ("Invalid character read in %s.\n", name);
            exit (0);
        }
    }
}
if ((c!=']') && (in%im)) {
    printf ("error reading matrix.");
    return (NULL);
}
in/=im;
if (fseek (fp, pos, SEEK_SET)) {
    printf ("Parse error.");
    exit (0);
}
this->resize (im, in);
for (i=1; i<=im; i++) {
    linha=M[i];
    linha++;
    for (j=0; j<(in-1); j++, linha++) {
        if (fscanf (fp, " %lf ", linha)!=1) {
            printf ("syntax error.");
            exit (0);
        }
    }
    if (i!=m) {
        if (fscanf (fp, "%lf ; ", linha)!=1) {
            printf ("syntax error.");
            exit (0);
        }
    }
    else {
        if (fscanf (fp, "%lf ]; ", linha)!=1) {
            printf ("syntax error.");
            exit (0);
        }
    }
}
return (name);
}

void densa::resize (int mm, int nn)
{
    int i, mx;
    double **linha,          /* ponteiro para uma linha. */
            **newM,          /* ponteiro para vetor de ponteiros realocados. */
            *newline;        /* ponteiro para linha realocada. */
#ifdef _DEBUG_ENTRY_
    printf ("Realocando matriz %p\n", M);
#endif
    if (!M) this->create (mm, nn);
    if (nn>nmax) {
        linha=M;
        *linha=NULL;
        linha++;
        for (i=1; i<=mm; i++, linha++) {
            newline=(double *) realloc ((*linha)+1, nn*sizeof (double));
#ifdef _VERIFY_ALLOC_
            if (!newline) {
                printf ("Memoria insuficiente %d+1.\n", nAlloc);
                exit (0);
            }
#endif
#ifdef _DEBUG_ALLOC_
            printf ("%d-esimo vetor %p da matriz %p realocado para %p.\n", i, *linha+1, M, newline);
#endif
            *linha=newline-1;

```

```

        /* sujeira para que o primeiro indice valido seja 1. */
    }
    nmax=nn;
}
if (mm>m) {
    mx=(mm>nmax)?nmax:mm;
    linha=M+m+1;
    for (i=m+1;i<=mx;i++, linha++) {
        (*linha)=(double*) malloc (nmax*sizeof (double));
#ifdef _VERIFY_ALLOC_
        if (!(*linha)&&nmax) {
            printf ("Memoria insuficiente %d+1.\n", nAlloc);
            exit (0);
        }
#endif
#ifdef _DEBUG_ALLOC_
        printf ("%d-esimo vetor %p da matriz %p realocado.\n", i, *linha, M);
#endif
        (*linha)--;
        /* sujeira para que o primeiro indice valido seja 1. */
    }
}
if (mm<m) {
    linha=M+mm+1;
    for (i=mm+1;i<=m;i++, linha++) {
        (*linha)++;
#ifdef _DEBUG_ALLOC_
        printf ("Liberando %d-esima linha %p da matriz %p.\n", i, *linha, M);
#endif
        free (*linha);
        *linha=NULL;
    }
}
if (mm>nmax) {
    /* Realoca vetor de ponteiros de linha. */
#ifdef _DEBUG_ALLOC_
    printf ("Realocando vetor de ponteiros da %d+1 instancia.\n", nAlloc);
#endif
    newM=(double **) realloc (M, (mm+1)*sizeof (double *));
#ifdef _VERIFY_ALLOC_
    if (!newM) {
        printf ("Memoria insuficiente %d+1.\n", nAlloc);
        exit (0);
    }
#endif
    M=newM;
    linha=M+nmax+1;
    for (i=nmax+1;i<=mm;i++, linha++) {
        (*linha)=(double*) malloc (nmax*sizeof (double));
#ifdef _VERIFY_ALLOC_
        if (!(*linha)&&nmax) {
            printf ("Memoria insuficiente %d+1.\n", nAlloc);
            exit (0);
        }
#endif
#ifdef _DEBUG_ALLOC_
        printf ("%d-esimo vetor %p da matriz %p realocado.\n", i, *linha, M);
#endif
        (*linha)--;
        /* sujeira para que o primeiro indice valido seja 1. */
    }
    nmax=mm;
}
m=mm;
n=nn;

#ifdef _DEBUG_END_
    printf ("%p Realocado.\n", M);
#endif
}

int densa::getsize (int d)
{

```

```

        switch (d) {
            case 1: return (m);
            case 2: return (n);
        }
        return (-1);
    }

int densa::getmaxsize (int d)
{
    switch (d) {
        case 1: return (mmax);
        case 2: return (nmax);
    }
    return (-1);
}

int densa::shdown (void)
{
    double *tmp;
    int i;
    tmp=M[m];
    for (i=m;i>1;i--) M[i]=M[i-1];
    M[1]=tmp;
    return (OK);
}

int densa::shup (void)
{
    double *tmp;
    int i;
    tmp=M[1];
    for (i=1;i<m;i++) M[i]=M[i+1];
    M[m]=tmp;
    return (OK);
}

void densa::matprt (FILE *fp, const char *name)
{
    int i, j;
    long type, mrows, ncols, imagf, namelen;

    mrows=m;
    ncols=n;          /* por causa do cast implicito. */
    type=0;           /* Little Endian, IEE double, densa. */
    imagf=0;          /* matriz real */
    namelen=strlen (name)+1;
    fwrite (&type, sizeof (long), 1, fp);
    fwrite (&mrows, sizeof (long), 1, fp);
    fwrite (&ncols, sizeof (long), 1, fp);
    fwrite (&imagf, sizeof (long), 1, fp);
    fwrite (&namelen, sizeof (long), 1, fp);
    fwrite (name, sizeof (char), namelen, fp);
    for (i=1;i<=n;i++) {
        for (j=1;j<=m;j++) fwrite (&(M[j][i]), sizeof (double), 1, fp);
    }
}

void densa::copy (densa *A)
{
    int i;
    double **pa, **pb;
    this->resize (A->m, A->n);
    pa=A->getpline(1);
    pb=this->getpline(1);
    for (i=1;i<=m;i++, pa++, pb++) memcpy (*pb+1, *pa+1, n*sizeof (double));
}

void densa::cellwr (FILE *fp, const char *name)
{
    int i, j;
    long type, mrows, ncols;

    mrows=m;

```



```

        ncols=n;          /* por causa do cast implicito. */
        type=T_MATRIX;
        fprintf (fp, "%s%c", name, '\0');
        fwrite (&type, sizeof (long), 1, fp);
        fwrite (&mrows, sizeof (long), 1, fp);
        fwrite (&ncols, sizeof (long), 1, fp);
        for (i=1;i<=n;i++) {
            for (j=1;j<=m;j++) fwrite (&(M[j][i]), sizeof (double), 1, fp);
        }
    }

int densa::cellrd (FILE *fp, char *name)
{
    int i, j;
    long type, mrows, ncols;

    i=0;
    do {
        name[i]=fgetc(fp);
    } while (name[i++]!='\0');

    fread (&type, sizeof (long), 1, fp);
    fread (&mrows, sizeof (long), 1, fp);
    fread (&ncols, sizeof (long), 1, fp);
    if (type!=T_MATRIX) {
        printf ("densa::cellrd: type mismatch.\n");
        exit (0);
    }
    this->resize (mrows, ncols);
    for (i=1;i<=n;i++) {
        for (j=1;j<=m;j++) fread (&(M[j][i]), sizeof (double), 1, fp);
    }
    return (0);
}

int densa::moverows (densa *src, int i0)
{
    int nrows, i, il;
    double **newM, **psrc, **pdest;
    if (i0>src->m) {
        printf ("%p->moverows(%p, %d): invalid index.\n",
            // this, src, i0);
        // exit (0);
        // return (0);          /* this is not an error */
    }
    if (nmax==0) {
        n=src->n; nmax=src->nmax;
    }
    if (nmax!=src->nmax) {
        printf ("%p->moverows(%p, %d): different row length.\n",
            this, src, i0);
        exit (0);
    }
    il=src->getsize (1);
    nrows=il-i0+1;
    if (nmax<(m+nrows)) { /* realloc */
        newM=(double **) realloc (M, (m+nrows+1)*sizeof (double *));
#ifdef _VERIFY_ALLOC_
        if (!newM) {
            printf ("%p->moverows(%p, %d): Memoria insuficiente %d+1.\n",
                this, src, i0);
            exit (0);
        }
#endif
        M=newM;
        nmax=m+nrows;
    }
    pdest=&M[m+1];
    psrc=&(src->M[i0]);
    for (i=i0;i<=il;i++, pdest++, psrc++) { /* moving rows */
        *pdest=*psrc;
        *psrc=NULL;
    }
}

```

```
m+=nrows;
src->m-=nrows;
return (0);
}
```

```

/* LINEAR.CPP
"
Fabio Nakano - 05.05.1997*/
"

#include <stdio.h>
"
#include <stdlib.h>
#include <math.h>
#include "global.hpp"
#include "densa.hpp"
#include "vetor.hpp"
#include "linear.hpp"

int vim (densa *V, vetor *x)
{
    int i, j, n;
    double *xp, /* aponta para a variavel que estamos resolvendo. */
            **plinha, *linha, /* aponta para a linha da matriz. */
            *xx, *xc; /* percorre o vetor x das variaveis resolvidas. */
    n=x->getsize();
#ifdef _VERIFY_BOUNDS_
    if ((n!=V->getsize(1)) || (V->getsize(1)!=V->getsize(2))) {
        printf ("vim - dimensions must agree.\n");
        exit (0);
    }
#endif
    if (!n) return (OK);
    plinha=V->getpline(n);
    xp=x->getline()+n; xx=xx;
    for (i=n; i>0;i--, xp--, plinha--) {
        xc=xx;
        linha=*plinha+n;
        for (j=n; j>i;j--, xc--, linha--) (*xp)-=(*xc)*(*linha);
        (*xp)/=(*linha);
    }
    return (OK);
}

int vtim (densa *V, vetor *x)
{
    int i, j, n;
    double *xp, /* aponta para a variavel que estamos calculando */
            **plinha, *linha, /* aponta para a linha da matriz. */
            *xx, *xc; /* percorre o vetor x das variaveis resolvidas */
    xp=x->getline();
    xp++;
    n=x->getsize();
#ifdef _VERIFY_BOUNDS_
    if ((n!=V->getsize(1)) || (V->getsize(1)!=V->getsize(2))) {
        printf ("vtim - dimensions must agree.\n");
        exit (0);
    }
#endif
    if (!n) return (OK);
    plinha=V->getpline(1);
    xx=x->getline()+1;
    for (i=1; i<=n;i++, xp++, plinha++) {
        xc=xx+i;
        linha=*plinha;
        linha+=i;
        (*xp)/=(*linha);
        linha++;
        for (j=i+1; j<=n;j++, xc++, linha++) (*xc)-=(*xp)*(*linha);
    }
    return (OK);
}

void vtam (densa *V, vetor *a)
{
    int i, j, n, nv;
    vetor b; /* o resultado. */

```

```

    register double ai; /* o elemento de a que vai ser usado. */
    double **plinha, *linha, *pb, *pbx, *pa;
//    double **psa, **psb;
    n=a->getsize();
    nv=V->getsize (2);
#ifdef _VERIFY_BOUNDS_
    if (n!=V->getsize(1)) {
        printf ("vtam - dimensions must agree.\n");
        exit (0);
    }
#endif
    if (!nv) {
        a->resize(0);
        return;
    }
    b.create(nv);
    plinha=V->getpline(1);
    linha=*plinha+1;
    pbx=b.getline()+1;
    pb=pbx;
    pa=a->getline()+1;
    ai=*pa;
    for (j=1;j<=nv;j++, linha++, pb++) (*pb)=(*linha)*ai;
    /* tinha que escolher entre inicializar com zeros (um loop) ou inicializar com
    o primeiro valor valido (um loop - uma iteracao no loop seguinte. Optei pela
    segunda. */
    plinha++;
    pa++;
    for (i=2;i<=n;i++, plinha++, pa++) {
        linha=*plinha+1;
        pb=pbx;
        ai=*pa;
        for (j=1;j<=nv;j++, linha++, pb++) (*pb)+=(*linha)*ai;
    }
    a->destroy ();
    a->setd (b.getline(), b.getsize()); /* b must not be resized */
    // b.destroy (); B must not be destroyed !!!!
}

void vam (densa *V, vetor *a)
{
    int i, j, n, nv;
    vetor b; /* o resultado. */
    double **plinha, *linha, *pa, *pax, *pb;
//    double **psa, **psb;
    n=a->getsize();
    nv=V->getsize (1);
#ifdef _VERIFY_BOUNDS_
    if (n!=V->getsize(2)) {
        printf ("vam - dimensions must agree.\n");
        exit (0);
    }
#endif
    if (!nv) {
        a->resize(0);
        return ;
    }
    b.create(nv);
    plinha=V->getpline(1);
    linha=*plinha;
    pax=a->getline()+1;
    pb=b.getline()+1;
    for (i=1;i<=nv;i++, plinha++, pb++) {
        linha=*plinha; linha++;
        pa=pax;
        *pb=0.0;
        for (j=1;j<=n;j++, linha++, pa++) (*pb)+=(*linha)*(*pa);
    }
    a->destroy ();
    a->setd (b.getline(), b.getsize()); /* b must not be resized
    n and nmax must be the same. */
    /* a->resize (nv); da pau quando troca arrays de tamanhos diferentes!!!
    psa=a->getpline();

```

```
psb=b.getpline();
pb=*psb;
*psb=*psa;
*psa=pb;
b.destroy (); */
}
```

```

/*QR.CPP*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "global.hpp"
#include "densa.hpp"
#include "vetor.hpp"
#include "linear.hpp"
#include "qr.hpp"

int QR (densa *V, densa *C, densa *E)
/* vou duplicar o codigo para cobrir o caso em que C=NULL, esperando que isso
aumente a eficiencia da funcao. Isto e' algo que seria interessante testar. */
{
    int i, j, mv, nv, nc, ne;
    double **pv1, **pv2, **pc1, **pc2, **pe1, **pe2;
    double gc, gs;
    if ((!C)||(!C->getsize(2))) { C=E; E=NULL; }
    mv=V->getsize (1);
    nv=V->getsize (2);
    if (!mv||!nv) return (OK);    /* nothing to do */
    pv1=V->getpline(1);
    if (E) {
#ifdef _VERIFY_BOUNDS_
        if (mv!=C->getsize(1)) {
            printf ("QR - dimensions must agree.\n");
            exit (0);
        }
        if (mv!=E->getsize(1)) {
            printf ("QR - dimensions must agree.\n");
            exit (0);
        }
    }
#endif
    nc=C->getsize (2);
    ne=E->getsize (2);
    pc1=C->getpline(1);
    pe1=E->getpline(1);
    for (i=1;i<=nv;i++, pv1++, pc1++, pe1++) {
        pv2=pv1+1;
        pc2=pc1+1;
        pe2=pe1+1;
        for (j=i+1;j<=mv;j++, pv2++, pc2++, pe2++) {
            givcs (&gc, &gs, (*pv1+i), (*pv2+i));
            givrt (pv1, pv2, nv, gc, gs);
            givrt (pc1, pc2, nc, gc, gs);
            givrt (pe1, pe2, ne, gc, gs);
        }
    }
    return (OK);
}
else {
    if ((C)&&(C->getsize(2))) {
#ifdef _VERIFY_BOUNDS_
        if (mv!=C->getsize(1)) {
            printf ("QR - dimensions must agree.\n");
            exit (0);
        }
    }
#endif
    nc=C->getsize (2);
    pc1=C->getpline(1);
    for (i=1;i<=nv;i++, pv1++, pc1++) {
        pv2=pv1+1;
        pc2=pc1+1;
        for (j=i+1;j<=mv;j++, pv2++, pc2++) {
            givcs (&gc, &gs, (*pv1+i), (*pv2+i));
            givrt (pv1, pv2, nv, gc, gs);
            givrt (pc1, pc2, nc, gc, gs);
        }
    }
    return (OK);
}
}

```

```

    }
}
for (i=1;i<=nv;i++, pv1++) {
    pv2=pv1+1;
    /* getpline (i+1), i==nv dispara a
       protecao de acesso. */
    for (j=i+1;j<=mv;j++, pv2++) {
        givcs (&gc, &gs, *(*pv1+i), *(*pv2+i));
        givrt (pv1, pv2, nv, gc, gs);
    }
}
return (OK);
}

int QRh (densa *V, densa *C, densa *E)
/* vou duplicar o codigo para cobrir o caso em que C==NULL, esperando que isso
aumente a eficiencia da funcao. Isto e' algo que seria interessante testar. */
{
    int i, mv, nv, nc, ne;
    double **pv1, **pv2, **pc1, **pc2, **pe1, **pe2;
    double gc, gs;
    if (!C) { C=E; E=NULL; }
    mv=V->getsize (1);
    nv=V->getsize (2);
    pv1=V->getpline(1);
    if (E) {
#ifdef _VERIFY_BOUNDS_
        if (mv!=C->getsize(1)) {
            printf ("QRh - dimensions must agree.\n");
            exit (0);
        }
        if (mv!=E->getsize(1)) {
            printf ("QRh - dimensions must agree.\n");
            exit (0);
        }
#endif
        nc=C->getsize (2);
        ne=E->getsize (2);
        pc1=C->getpline(1);
        pe1=E->getpline(1);
        for (i=1;i<mv;i++, pv1++, pc1++, pe1++) {
            pv2=pv1+1;
            pc2=pc1+1;
            pe2=pe1+1;
            givcs (&gc, &gs, *(*pv1+i), *(*pv2+i));
            givrt (pv1, pv2, nv, gc, gs);
            givrt (pc1, pc2, nc, gc, gs);
            givrt (pe1, pe2, ne, gc, gs);
        }
        return (OK);
    }
    else {
        if (C) {
#ifdef _VERIFY_BOUNDS_
            if (mv!=C->getsize(1)) {
                printf ("QRh - dimensions must agree.\n");
                exit (0);
            }
#endif
            nc=C->getsize (2);
            pc1=C->getpline(1);
            for (i=1;i<mv;i++, pv1++, pc1++) {
                pv2=pv1+1;
                pc2=pc1+1;
                givcs (&gc, &gs, *(*pv1+i), *(*pv2+i));
                givrt (pv1, pv2, nv, gc, gs);
                givrt (pc1, pc2, nc, gc, gs);
            }
            return (OK);
        }
    }
    for (i=1;i<mv;i++, pv1++) {
        pv2=pv1+1;
        /* getpline (i+1), i==nv dispara a
           protecao de acesso. */

```

```

        givcs (&gc, &gs, *(*pv1+i), *(*pv2+i));
        givrt (pv1, pv2, nv, gc, gs);
    }
    return (OK);
}

double QRx (vetor *x, densa *V, densa *C, densa *E)
/* vou duplicar o codigo para cobrir o caso em que C=NULL, esperando que isso
aumente a eficiencia da funcao. Isto e' algo que seria interessante testar. */
{
    int j, m, nv, nc, ne;
    double **pv, **pc, **pe, *px, a, b;
    double gc, gs;
    if (!C) { C=E; E=NULL; }
    m=x->getsize();
#ifdef _VERIFY_BOUNDS_
    if (m!=V->getsize(1)) {
        printf ("QRx - dimensions must agree.\n");
        exit (0);
    }
#endif
    px=x->getline()+m;
    pv=V->getpline(m);
    nv=V->getsize(2);
    if (E) {
#ifdef _VERIFY_BOUNDS_
        if (m!=C->getsize(1)) {
            printf ("QRx - dimensions must agree.\n");
            exit (0);
        }
        if (m!=E->getsize(1)) {
            printf ("QRx - dimensions must agree.\n");
            exit (0);
        }
#endif
    }
    nc=C->getsize (2);
    pc=C->getpline(m);
    ne=E->getsize (2);
    pe=E->getpline(m);
    for (j=m;j>1;j--, px--, pv--, pc--, pe--) {
        givcs (&gc, &gs, *(px-1), *px);
        a=gc*(*(px-1))-gs*(*px);
        b=gs*(*(px-1))+gc*(*px);
        *px=b;          /* sei que isso vai valer zero sempre,
                           mas pode ser um bom indicador de erro. */
        *(px-1)=a;
        givrt (pv-1, pv, nv, gc, gs);
        givrt (pc-1, pc, nc, gc, gs);
        givrt (pe-1, pe, ne, gc, gs);
    }
    return (*px);
}
else {
    if (C) {
#ifdef _VERIFY_BOUNDS_
        if (m!=C->getsize(1)) {
            printf ("QRx - dimensions must agree.\n");
            exit (0);
        }
#endif
    }
    nc=C->getsize (2);
    pc=C->getpline(m);
    for (j=m;j>1;j--, px--, pv--, pc--) {
        givcs (&gc, &gs, *(px-1), *px);
        a=gc*(*(px-1))-gs*(*px);
        b=gs*(*(px-1))+gc*(*px);
        *px=b;          /* sei que isso vai valer zero sempre,
                           mas pode ser um bom indicador de erro. */
        *(px-1)=a;
        givrt (pv-1, pv, nv, gc, gs);
        givrt (pc-1, pc, nc, gc, gs);
    }
    return (*px);
}

```



```

    }
}
for (j=m;j>1;j--, px--, pv--) {
    givcs (&gc, &gs, *(px-1), *px);
    a=gc*(*(px-1))-gs*(*px);
    b=gs*(*(px-1))+gc*(*px);
    *px=b;          /* sei que isso vai valer zero sempre,
                     mas pode ser um bom indicador de erro. */
    *(px-1)=a;
    givrt (pv-1, pv, nv, gc, gs);
}
return (*px);
}

void givcs (double *gc, double *gs, double x1, double x2)
{
    long double tau, gcos, gsen;
    if (fabs (x2)<EPS) {
        gcos=1.0; gsen=0.0;
    }
    else {
        if (fabs(x1)>=fabs(x2)) {
            tau=-x2/x1;
            gcos=1.0/sqrt(1+tau*tau);
            gsen=gcos*tau;
        }
        else {
            tau=-x1/x2;
            gsen=1.0/sqrt(1+tau*tau);
            gcos=gsen*tau;
        }
    }
    *gc=gcos; *gs=gsen;
}

void givrt (double **v1, double **v2, int n, double gc, double gs)
/* Esta funcao e' realmente esquisita, ja' que nao quero ter o trabalho de
copiar regioes de memoria, logo, trocar os ponteiros e' a solucao, mas nao
quero fazer alocao diretamente aqui, logo, uso as abstracoes de vetor e
de matriz. A esquisitice esta em que as classes nao tem nenhuma relacao
(nao sao parentes) apesar de serem estruturalmente semelhantes e estou
trocando estruturas internas - e' como trocar as "tripas" entre dois bichos
que tem pouca semelhanca um com o outro. MAS NAO PRECISAVA DE DOIS VETORES
AUXILIARES...*/
{
    vetor a, b;
    double *va, *vb, *vv1, *vv2;
    double **pv, *tmp;
    int i;
    a.create (n);
    b.create (n);
    va=a.getline(); vb=b.getline(); vv1=*v1; vv2=*v2;
    va++; vb++, vv1++; vv2++;
    for (i=1;i<=n;i++, va++, vb++, vv1++, vv2++) {
        *va=gc*(*vv1)-gs*(*vv2);
        *vb=gs*(*vv1)+gc*(*vv2);
    }
    pv=a.getpline();
    tmp=*pv;
    *pv=*v1;
    *v1=tmp;
    pv=b.getpline();
    tmp=*pv;
    *pv=*v2;
    *v2=tmp;
    a.destroy();
    b.destroy();
}
/*
void main (void)
{
    FILE *fpOut, *fpIn;
    densa B, C;

```

```

vetor x, y;
char nome[80];
fpOut=fopen ("t.m", "wt");
if (!fpOut) perror ("t.m");
fpIn=fopen ("in.m", "rt");
if (!fpIn) perror ("in.m");
printf ("Li %s.\n", C.fread (fpIn, nome));
printf ("Li %s.\n", B.fread (fpIn, nome));
printf ("Li %s.\n", x.fread (fpIn, nome));
fclose (fpIn);
C.fprint (fpOut, "C");
x.fprint (fpOut, "b");
y.copy (&x);
QR (&C, &B);
B.fprint (fpOut, "B");
fprintf (fpOut, "normx=%lf\n", QRx (&x, &C, &B));
B.fprint (fpOut, "Bx");
QRh (&C, &B);
B.fprint (fpOut, "Bk");
fclose (fpOut);
y.destroy ();
C.destroy ();
B.destroy ();
}
*/

```

```

/*XRBAF.CPP*/
#include <stdio.h>
"
#include <stdlib.h>
"
#include <string.h>
"
#include <malloc.h>
"

#include "global.hpp"
"
#include "densa.hpp"
"
#include "xrbafe.hpp"
"

static int nInstance=0, nAlloc=0;
"

int getxrbafeInstance(void) { return (nInstance); }
"
int getxrbafeAlloc(void) { return (nAlloc); }

void initclassxrbafe (void)
{
    nInstance=0;
    nAlloc=0;
}

xrbafe::xrbafe ()
{
    D=NULL;
    C=NULL;
    E=NULL;
    n=0;
    nmax=0;
    nInstance++;
}

xrbafe::~xrbafe ()
{
    nInstance--;
}

xrbafe::create (int ni)
{
#ifdef _DEBUG_ENTRY_
    printf ("Criando %d+1 instancia.\n", nAlloc);
/* ja que tem que ser a primeira instrucao... */
#endif

    D=(densa**) calloc (ni, sizeof (densa*));
#ifdef _VERIFY_ALLOC_
    if (!D) {
        printf ("Memoria insuficiente %d+1.\n", nAlloc);
        exit (0);
    }
#endif
    D--; /* sujeira para que o primeiro indice valido seja 1. */
    C=(densa**) calloc (ni, sizeof (densa*));
#ifdef _VERIFY_ALLOC_
    if (!C) {
        printf ("Memoria insuficiente %d+1.\n", nAlloc);
        exit (0);
    }
#endif
    C--; /* sujeira para que o primeiro indice valido seja 1. */
    E=(densa**) calloc (ni, sizeof (densa*));
#ifdef _VERIFY_ALLOC_

```

```

        if (!E) {
            printf ("Memoria insuficiente %d+1.\n", nAlloc);
            exit (0);
        }
    #endif
    E--; /* sujeira para que o primeiro indice valido seja 1. */
    nAlloc++;
    n=ni;
    nmax=n;
    #ifdef _DEBUG_END_
        printf ("%d instancia criada %p.\n", nAlloc, D);
    #endif
}

void xrbaf::createdblock (int i, int m, int n)
{
    D[i]=(densa*) calloc (1, sizeof (densa));
    #ifdef _VERIFY_ALLOC_
        if (!D[i]) {
            printf ("Memoria insuficiente. %p\n", D);
            exit (0);
        }
    #endif
    D[i]->create (m, n);
}

void xrbaf::createcblock (int i, int m, int n)
{
    C[i]=(densa*) calloc (1, sizeof (densa));
    #ifdef _VERIFY_ALLOC_
        if (!C[i]) {
            printf ("Memoria insuficiente. %p\n", C);
            exit (0);
        }
    #endif
    C[i]->create (m, n);
}

void xrbaf::createeblock (int i, int m, int n)
{
    E[i]=(densa*) calloc (1, sizeof (densa));
    #ifdef _VERIFY_ALLOC_
        if (!E[i]) {
            printf ("Memoria insuficiente. %p\n", E);
            exit (0);
        }
    #endif
    E[i]->create (m, n);
}

void xrbaf::destroy ()
{
    densa **p;
    int i;
    #ifdef _DEBUG_ENTRY_
        printf ("Removendo %d instancia %p.\n", nAlloc, D);
    #endif
    #ifdef _DEBUG_ALLOC_
        if (D) {
            D++;
            printf ("Liberando vetor %p.\n", D);
        }
    #endif
    p=D; for (i=1;i<=n;i++, p++) {
        if (*p) {
            (*p)->destroy();
            free (*p);
        }
    }
    free (D);
    D=NULL;

    if (C) {
        C++;
    }
}

```

```

#ifdef _DEBUG_ALLOC_
    printf ("Liberando vetor %p.\n", C);
#endif
    p=C; for (i=1;i<=n;i++, p++) {
        if (*p) {
            (*p)->destroy();
            free (*p);
        }
    }
    free (C);
    C=NULL;
}
if (E) {
    E++;
#ifdef _DEBUG_ALLOC_
    printf ("Liberando vetor %p.\n", E);
#endif
    p=E; for (i=1;i<=n;i++, p++) {
        if (*p) {
            (*p)->destroy();
            free (*p);
        }
    }
    free (E);
    E=NULL;
}
nAlloc--;
n=0;
nmax=0;
#ifdef _DEBUG_END_
    printf ("%d instancia removida.\n", nAlloc);
#endif
}

densa **xrbaf::getpdblock (int i)
{
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>n)) {
        printf ("%p(%d) fora dos limites do vetor.\n", D, i);
        exit (0);
    }
#endif
    return (&D[i]);
}

densa *xrbaf::getdblock (int i)
{
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>n)) {
        printf ("%p(%d) fora dos limites do vetor.\n", D, i);
        exit (0);
    }
#endif
    return (D[i]);
}

densa **xrbaf::getpcblock (int i)
{
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>n)) {
        printf ("%p(%d) fora dos limites do vetor.\n", C, i);
        exit (0);
    }
#endif
    return (&C[i]);
}

densa *xrbaf::getcblock (int i)
{
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>n)) {
        printf ("%p(%d) fora dos limites do vetor.\n", C, i);
        exit (0);
    }

```

```

    }
#endif
    return (C[i]);
}

densa **xrbaf::getpeblock (int i)
{
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>n)) {
        printf ("%p(%d) fora dos limites do vetor.\n", E, i);
        exit (0);
    }
#endif
    return (&E[i]);
}

densa *xrbaf::geteblock (int i)
{
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>n)) {
        printf ("%p(%d) fora dos limites do vetor.\n", 'E', i);
        exit (0);
    }
#endif
    return (E[i]);
}

void xrbaf::resize (int nn)
{
    int i;
    densa **newline; /* ponteiro para linha realocada. */
    densa **pp, **pc, **pd, **pe;
#ifdef _DEBUG_ENTRY_
    printf ("Redimensionando vetor %p\n", D);
#endif
    if (!D) this->create (nn);
    if (nn>nmax) {
        D++;
#ifdef _DEBUG_ALLOC_
        printf ("Realocando vetor %p\n", D);
#endif
        newline=(densa **) realloc (D, nn*sizeof (densa*));
#ifdef _VERIFY_ALLOC_
        if (!newline) {
            printf ("Memoria insuficiente %d+1.\n", nAlloc);
            exit (0);
        }
#endif
        pp=newline+n;
        for (i=n+1; i<=nn;i++, pp++) *pp=NULL;
        D=newline-1; /* sujeira para que o primeiro indice valido seja 1. */

        C++;
        newline=(densa **) realloc (C, nn*sizeof (densa*));
#ifdef _VERIFY_ALLOC_
        if (!newline) {
            printf ("Memoria insuficiente %d+1.\n", nAlloc);
            exit (0);
        }
#endif
        pp=newline+n;
        for (i=n+1; i<=nn;i++, pp++) *pp=NULL;
        C=newline-1; /* sujeira para que o primeiro indice valido seja 1. */

        E++;
        newline=(densa **) realloc (E, nn*sizeof (densa*));
#ifdef _VERIFY_ALLOC_
        if (!newline) {
            printf ("Memoria insuficiente %d+1.\n", nAlloc);
            exit (0);
        }
#endif
        pp=newline+n;

```

```

        for (i=n+1; i<=nn;i++, pp++) *pp=NULL;
        E=newline-1; /* sujeira para que o primeiro indice valido seja 1. */

        nmax=nn;
    }
    else {
        if (nn!=n) {
            pc=C+nn+1; pd=D+nn+1;pe=E+nn+1;
            for (i=nn+1;i<=n;i++, pc++, pd++, pe++) {
                if (*pc) {
                    (*pc)->destroy();
                    free (*pc);
                    *pc=NULL;
                }
                if (*pd) {
                    (*pd)->destroy();
                    free (*pd);
                    *pd=NULL;
                }
                if (*pe) {
                    (*pe)->destroy();
                    free (*pe);
                    *pe=NULL;
                }
            }
        }
    }
    n=nn;

#ifdef _DEBUG_END_
    printf ("%p Realocado.\n", D);
#endif
}

int xrbaf::getsize (void)
{
    return (n);
}

int xrbaf::getmaxsize (void)
{
    return (nmax);
}

int xrbaf::cellrd (FILE *fp, char *name)
{
    int i;
    long type, mrows, ncols;

    i=0;
    do {
        name[i]=fgetc(fp);
    } while (name[i++]);
    fread (&type, sizeof (long), 1, fp);
    fread (&mrows, sizeof (long), 1, fp);
    fread (&ncols, sizeof (long), 1, fp);
    if (type!=T_CELLARRAY) {
        printf ("xrbaf::cellrd: type mismatch.\n");
        exit (0);
    }
    if (ncols!=3) {
        printf ("xrbaf::cellrd: size mismatch ncols=%d.\n", ncols);
        exit (0);
    }
    this->destroy ();
    this->create (mrows);
    for (i=1;i<=n;i++) {
        D[i]=(densa *) calloc (1, sizeof (densa));
        D[i]->cellrd (fp, name);
        C[i]=(densa *) calloc (1, sizeof (densa));
        C[i]->cellrd (fp, name);
        E[i]=(densa *) calloc (1, sizeof (densa));
        E[i]->cellrd (fp, name);
    }
}

```

```

    }
    return (0);
}

void xrbaf::cellwr (FILE *fp, const char *name)
{
    int i;
    long type, mrows, ncols;
    densa Z;

    Z.create (0,0);
    mrows=n;
    ncols=3;          /* por causa do cast implicito. */
    type=T_CELLARRAY;
    fprintf (fp, "%s%c", name, '\0');
    fwrite (&type, sizeof (long), 1, fp);
    fwrite (&mrows, sizeof (long), 1, fp);
    fwrite (&ncols, sizeof (long), 1, fp);
    for (i=1;i<=n;i++) {
        D[i]?D[i]->cellwr (fp, ""):Z.cellwr (fp, "");
        C[i]?C[i]->cellwr (fp, ""):Z.cellwr (fp, "");
        E[i]?E[i]->cellwr (fp, ""):Z.cellwr (fp, "");
    }
}

void xrbaf::copy (xrbaf *A)
{
    int i;
    densa **pa, **pb, **pc, **pd, **pe1, **pe2;
    this->resize (A->n);
    pa=(this->D)+1;
    pb=(this->C)+1;
    pe2=(this->E)+1;
    pc=(A->D)+1;
    pd=(A->C)+1;
    pe1=(A->E)+1;
    for (i=1;i<=n;i++, pa++, pb++, pc++, pd++, pe1++, pe2++) {
        if (*pc) {
            if (!(*pa)) (*pa)=(densa*) calloc (1, sizeof (densa));
            (*pa)->copy (*pc);
        }
        if (*pd) {
            if (!(*pb)) (*pb)=(densa*) calloc (1, sizeof (densa));
            (*pb)->copy (*pd);
        }
        if (*pe1) {
            if (!(*pe2)) (*pe2)=(densa*) calloc (1, sizeof (densa));
            (*pe2)->copy (*pe1);
        }
    }
}

#ifdef _TESTE_XRBAF_
void main (int argc, char *argv[])
{
    FILE *fpIn, *fpOut;
    char name[80];
    xrbaf B;
    // _heapinfo heap;
    if (argc!=3) {
        puts ("Argument count mismatch.");
        puts ("Usage: xrbaf <inputfilename> <outputfilename>");
        exit (0);
    }
    fpIn=fopen (argv[1], "rb");
    if (!fpIn) {
        perror (argv[1]);
        exit (0);
    }
    fpOut=fopen (argv[2], "wb");
    if (!fpOut) {
        perror (argv[2]);
        exit (0);
    }
}

```



```
    }  
    B.cellrd (fpIn, name);  
    puts (name);  
    B.cellwr (fpOut, "teste");  
    B.destroy ();  
    fclose (fpIn);  
    fclose (fpOut);  
}  
#endif
```

```

/*NRBAF.CPP*/

#include <stdio.h>
"
#include <stdlib.h>
"
#include <string.h>
"
#include <malloc.h>

#include "global.hpp"
#include "vetor.hpp"
#include "linear.hpp"
#include "qr.hpp"
#include "xrbaf.hpp"
#include "nrbaf.hpp"

heap_dump()
{
    struct _heapinfo h_info;
    int heap_status, i;

    h_info._pentry = NULL;
    i=1;
    for(;;) {
        if (i==20) { i=1; getchar();}
        i++;
        heap_status = _heapwalk( &h_info );
        if( heap_status != _HEAPOK ) break;
        printf( " %s block at %Fp of size %4.4X\n",
            (h_info._useflag == _USEDENTRY ? "USED" : "FREE"),
            h_info._pentry, h_info._size );
    }

    switch( heap_status ) {
    case _HEAPEND:
        printf( "OK - end of heap\n" );
        break;
    case _HEAPEMPTY:
        printf( "OK - heap is empty\n" );
        break;
    case _HEAPBADBEGIN:
        printf( "ERROR - heap is damaged\n" );
        exit (0);
        break;
    case _HEAPBADPTR:
        printf( "ERROR - bad pointer to heap\n" );
        exit (0);
        break;
    case _HEAPBADNODE:
        printf( "ERROR - bad node in heap\n" );
        exit (0);
    }
}

static int nInstance=0, nAlloc=0;

int getnrbafInstance(void) { return (nInstance); }
int getnrbafAlloc(void) { return (nAlloc); }

void initclassnrbaf (void)
{
    nInstance=0;
    nAlloc=0;
}

nrbaf::nrbaf ()
{
    X=NULL;
    n=0;
    nmax=0;
    nInstance++;
}

```

```

}

nrbafe::~nrbafe ()
{
    nInstance--;
}

nrbafe::create (int ni)
{
#ifdef _DEBUG_ENTRY_
    printf ("Criando %d+1 instancia.\n", nAlloc);
/* ja que tem que ser a primeira instrucao... */
#endif

    X=(xrbaf**) calloc (ni, sizeof (xrbaf*));
#ifdef _VERIFY_ALLOC_
    if (!X) {
        printf ("Memoria insuficiente %d+1.\n", nAlloc);
        exit (0);
    }
#endif
    X--; /* sujeira para que o primeiro indice valido seja 1. */
    nAlloc++;
    n=ni;
    nmax=n;
#ifdef _DEBUG_END_
    printf ("%d instancia criada %p.\n", nAlloc, X);
#endif
}

void nrbafe::createblock (int i, int nx)
{
    X[i]=(xrbaf *) calloc (1, sizeof (xrbaf));
#ifdef _VERIFY_ALLOC_
    if (!X[i]) {
        printf ("Memoria insuficiente. %p\n", X);
        exit (0);
    }
#endif
    X[i]->create (nx);
}

void nrbafe::destroy ()
{
    xrbaf **p;
    int i;
#ifdef _DEBUG_ENTRY_
    printf ("Removendo %d instancia %p.\n", nAlloc, X);
#endif
    if (X) {
        X++;
#ifdef _DEBUG_ALLOC_
        printf ("Liberando vetor %p.\n", X);
#endif
    }
    p=X; for (i=1;i<=n;i++, p++) {
        if (*p) {
            (*p)->destroy();
            free (*p);
        }
    }
    free (X);
    X=NULL;
    nAlloc--;
    n=0;
    nmax=0;
#ifdef _DEBUG_END_
    printf ("%d instancia removida.\n", nAlloc);
#endif
}

xrbaf **nrbafe::getpblock (int i)
{

```

```

#ifdef _VERIFY_BOUNDS_
    if ((i<=0)||i>n) {
        printf ("%p(%d) fora dos limites do vetor.\n", X, i);
        exit (0);
    }
#endif
return (&X[i]);
}

xrbaf *nrbafe::getblock (int i)
{
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)||i>n) {
        printf ("%p(%d) fora dos limites do vetor.\n", X, i);
        exit (0);
    }
#endif
return (X[i]);
}

void nrbafe::resize (int nn)
{
    int i;
    xrbaf **newline;      /* ponteiro para linha realocada. */
    xrbaf **pp;
#ifdef _DEBUG_ENTRY_
    printf ("Redimensionando vetor %p\n", X);
#endif
    if (!X) this->create (nn);
    if (nn>nmax) {
        X++;
#ifdef _DEBUG_ALLOC_
        printf ("Realocando vetor %p\n", X);
#endif
        newline=(xrbaf **) realloc (X, nn*sizeof (xrbaf*));
#ifdef _VERIFY_ALLOC_
        if (!newline) {
            printf ("Memoria insuficiente %d+1.\n", nAlloc);
            exit (0);
        }
#endif
        pp=newline+n;
        for (i=n+1; i<=nn;i++, pp++) *pp=NULL;
        X=newline-1; /* sujeira para que o primeiro indice valido seja 1. */

        nmax=nn;
    }
    else {
        if (nn!=n) {
            pp=X+nn+1;
            for (i=nn+1;i<=n;i++, pp++) {
                if (*pp) {
                    (*pp)->destroy();
                    free (*pp);
                    *pp=NULL;
                }
            }
        }
    }
    n=nn;

#ifdef _DEBUG_END_
    printf ("%p Realocado.\n", X);
#endif
}

int nrbafe::getsize (void)
{
    return (n);
}

int nrbafe::getmaxsize (void)
{

```

```

        return (nmax);
    }

    int nrbafe::getxsize (int i)
    {
        return (X[i]->getsize());
    }

    void nrbafe::xresize (int i, int nn)
    {
        X[i]->resize(nn);
    }

    densa *nrbafe::getdblock (int i, int j)
    {
        return (X[i]->getdblock(j));
    }

    densa *nrbafe::getcblock (int i, int j)
    {
        return (X[i]->getcblock(j));
    }

    densa *nrbafe::geteblock (int i, int j)
    {
        return (X[i]->geteblock(j));
    }

    int nrbafe::cellrd (FILE *fp, char *name)
    {
        int i;
        long type, mrows, ncols;

        i=0;
        do {
            name[i]=fgetc(fp);
        } while (name[i++] != '\0');
        fread (&type, sizeof (long), 1, fp);
        fread (&mrows, sizeof (long), 1, fp);
        fread (&ncols, sizeof (long), 1, fp);
        if (type!=T_CELLARRAY) {
            printf ("nrbafe::cellrd: type mismatch.\n");
            exit (0);
        }
        if (ncols!=1) {
            printf ("nrbafe::cellrd: this is not nrbafe: ncols=%d.\n", ncols);
            exit (0);
        }
        this->destroy ();
        this->create (mrows);
        for (i=1; i<=n; i++) {
            X[i]=(xrbaf *) calloc (1, sizeof (xrbaf));
            X[i]->cellrd (fp, name);
        }
        return (0);
    }

    void nrbafe::cellwr (FILE *fp, const char *name)
    {
        int i;
        long type, mrows, ncols;

        mrows=n;
        ncols=1; /* por causa do cast implicitito. */
        type=T_CELLARRAY;
        fprintf (fp, "%s%c", name, '\0');
        fwrite (&type, sizeof (long), 1, fp);
        fwrite (&mrows, sizeof (long), 1, fp);
        fwrite (&ncols, sizeof (long), 1, fp);
        for (i=1; i<=n; i++) {
            X[i]->cellwr (fp, "");
        }
    }

```

```

void nrbafe::copy (nrbafe *A)
{
    int i;
    xrbaf **pa, **pb;
    this->resize (A->n);
    pa=(this->X)+1;
    pb=(A->X)+1;
    for (i=1;i<=n;i++, pa++, pb++) {
        if (*pb) {
            if (!(*pa)) (*pa)=(xrbaf*) calloc (1, sizeof (xrbaf));
            (*pa)->copy (*pb);
        }
    }
}

int nrbafe::QR ()
{
    int nexternal, ninternal, i, k, nc, nd, ne, me;
    xrbaf *pxr, *pex;
    nexternal=n;
    this->resize (nexternal+1);
    X[nexternal+1]=(xrbaf *) calloc (1, sizeof (xrbaf));
    ne=X[1]->geteblock(1)->getsize(2);
    pex=X[nexternal+1];
    pex->create (1);
    pex->createeblock (1, 0, ne);
    for (k=1;k<=nexternal;k++) {
        pxr=X[k];
        ninternal=pxr->getsize ();
        pxr->resize (ninternal+1);
        nc=pxr->getcblock(1)->getsize (2);
        pxr->createcblock (ninternal+1, 0, nc);
        pxr->createeblock (ninternal+1, 0, ne);
        for (i=1;i<=ninternal; i++) { /* fatora cada tripla D[i], C[i], E[i] */
            nd=pxr->getdblock(i)->getsize (2);
            me=pxr->geteblock(i)->getsize (1);
            ::QR (pxr->getdblock(i), pxr->getcblock(i), pxr->geteblock(i));
            pxr->getdblock (i)->resize (nd, nd);
            pxr->getcblock (ninternal+1)->moverows (pxr->getcblock (i), nd+1);
            pxr->geteblock (ninternal+1)->moverows (pxr->geteblock (i), nd+1);
        } // for i
        me=pxr->geteblock(ninternal+1)->getsize (1);
        ::QR (pxr->getcblock(ninternal+1), pxr->geteblock(ninternal+1), NULL);
        pxr->getcblock (ninternal+1)->resize (nc, nc);
        pex->geteblock (1)->moverows (pxr->geteblock (ninternal+1), nc+1);
    } // for k
    ::QR (pex->geteblock(1), NULL, NULL);
    pex->geteblock(1)->resize (ne, ne);
    return (OK);
}

int nrbafe::upi (int ke, int ki, vetor *qd, vetor *qc, vetor *qe)
{
    densa *pd, *pc, *pe, *pci, *pei, *pex;
    double *pdd, *pcc, *pee;
    int ninternal, nexternal;
    int nd, nc, ne;
    nexternal=n;
    ninternal=this->X[ke]->getsize ();
#ifdef _VERIFY_BOUNDS_
    if (ke>nexternal) {
        printf ("No %p(%d) xrbaf block.=n", ke, this);
        exit (0);
    }
    if (ki>ninternal) {
        printf ("No %p(%d)(%d) densa block.=n", ke, ki, this);
        exit (0);
    }
#endif
    pd=this->X[ke]->getdblock(ki);
    nd=pd->getsize (2);
    pc=this->X[ke]->getcblock(ki);

```

```

        nc=pc->getsize (2);
        pe=this->X[ke]->geteblock(ki);
        ne=pe->getsize (2);
#ifdef _VERIFY_BOUNDS_
        if (nd!=qd->getsize()) {
            printf ("%p.upi diagonal block update dimensions mismatch.\n", this);
            exit (0);
        }
        if (nc!=qc->getsize()) {
            printf ("%p.upi internal coupling block update dimensions mismatch.\n", this);
            exit (0);
        }
        if (ne!=qe->getsize()) {
            printf ("%p.upi external coupling block update dimensions mismatch.\n", this);
            exit (0);
        }
#endif
        pd->resize (pd->getsize(1)+1, nd);
        pd->shdown();
        pdd=pd->getline(1);
        memcpy (pdd+1, qd->getline()+1, nd*sizeof (double));
        pc->resize (pc->getsize(1)+1, nc);
        pc->shdown();
        pcc=pc->getline(1);
        memcpy (pcc+1, qc->getline()+1, nc*sizeof (double));
        pe->resize (pe->getsize(1)+1, ne);
        pe->shdown();
        pee=pe->getline(1);
        memcpy (pee+1, qe->getline()+1, ne*sizeof (double));
        ::QRh (pd, pc, pe);
        pd->resize (nd, nd);
        pci=X[ke]->getcblock (ninternal);
        pei=X[ke]->geteblock (ninternal);
        pci->moverows (pc, nd+1);
        pci->shdown();
        pei->moverows (pe, nd+1);
        pei->shdown();
        ::QRh (pci, pei, NULL);
        pci->resize (nc, nc);
        pex=X[nexternal]->geteblock(1);
        pex->moverows (pei, nc+1);
        pex->shdown();
        ::QRh (pex, NULL, NULL);
        pex->resize (ne, ne);
        return (OK);
    }
}

```

```

void nrbafe::uix (vetor *x)
// x=inv(U)*x; U is nrbafe, upper triangular
{
    int nexternal, ninternal, mx, m, n, i, j=0;
    double *px;
    vetor xce, xci, xdi,
        Uxce, Uxci;
    nexternal= this->n;
    mx=x->getsize(); // U and x MUST have compatible sizes.
    m=X[nexternal]->geteblock (1)->getsize(1);
    n=X[nexternal]->geteblock (1)->getsize(2);
    px=x->getline();
    xce.setd (px+mx-m, m); // pass sub-array pointer to the function
    vim (X[nexternal]->geteblock(1), &xce);
    mx=mx-m;
    for (i=nexternal-1;i>0;i--) {
        ninternal=X[i]->getsize();
        m=X[i]->getcblock(ninternal)->getsize(1);
        n=X[i]->getcblock(ninternal)->getsize(2);
        xci.setd(px+mx-m, m);
        if (xce.getsize()>0) {
            Uxce.destroy ();
            Uxce.copy (&xce);
            vam(X[i]->geteblock(ninternal), &Uxce);
            xci.minus(&Uxce);
        }
    }
}

```

```

    }
    vim (X[i]->getcblock(ninternal), &xci);
    mx-=m;
    for (j=ninternal-1;j>0;j--) {
        m=X[i]->getdblock(j)->getsize(1);
        n=X[i]->getdblock(j)->getsize(2);
        xdi.setd(px+mx-m, m);
        if (xci.getsize()>0) {
            Uxci.destroy ();
            Uxci.copy (&xci);
            vam(X[i]->getcblock(j), &Uxci);
            xdi.minus(&Uxci);
        }
        if (xce.getsize()>0) {
            Uxce.destroy ();
            Uxce.copy (&xce);
            vam (X[i]->geteblock(j), &Uxce);
            xdi.minus(&Uxce);
        }
        vim (X[i]->getdblock(j), &xdi);
        mx-=m;
    }
    Uxce.destroy();
    Uxci.destroy();
}

void nrbafe::utix (vetor *x)
{
    int nexternal, ninternal, i, j, m, n, mx0;
    vetor di, de, xdi, xde, xci, xce, Uxdi, Uxci;
    double *px;
    nexternal=this->n;
    mx0=0;
    de.create(X[nexternal]->geteblock(1)->getsize (2));
    de.fill (0.0);
    for (i=1;i<nexternal;i++) {
        ninternal=X[i]->getsize();
        di.resize(X[i]->getcblock(ninternal)->getsize (2));
        di.fill (0.0);
        px=x->getline();
        for (j=1;j<ninternal;j++) {
            m=X[i]->getdblock(j)->getsize(1);
            n=X[i]->getdblock(j)->getsize(2);
            xdi.setd(px+mx0, n);
            vtim (X[i]->getdblock(j), &xdi);
            if (X[i]->getcblock(j)->getsize (1)) {
                Uxdi.copy (&xdi);
                vtam(X[i]->getcblock(j), &Uxdi);
                di.plus(&Uxdi);
            }
            if (X[i]->geteblock(j)->getsize (1)) {
                Uxdi.copy (&xdi);
                vtam(X[i]->geteblock(j), &Uxdi);
                de.plus(&Uxdi);
            }
        }
        mx0+=n; // o tamanho amarra o processamento
    }
    m=X[i]->getcblock(ninternal)->getsize(1);
    n=X[i]->getcblock(ninternal)->getsize(2);
    xci.setd(px+mx0, n);
    xci.minus(&di);
    vtim (X[i]->getcblock(ninternal), &xci);
    if (X[i]->geteblock(ninternal)->getsize (1)) {
        Uxci.copy (&xci);
        vtam(X[i]->geteblock(ninternal), &Uxci);
        de.plus(&Uxci);
    }
    mx0+=n;
}

m=X[nexternal]->geteblock(1)->getsize(1);
n=X[nexternal]->geteblock(1)->getsize(2);
xce.setd(px+mx0, n);

```



```

    xce.minus(&de);
    vtim(X[nexternal]->geteblock(1), &xce);
    Uxdi.destroy();
    Uxci.destroy();
    di.destroy();
    de.destroy();
}

void nrbafe::uitet(int ke, int ki, vetor *bdi, vetor *bci, vetor *bce)
{
    int nexternal, ninternal;
    vetor Uqdi, Uqci;
    nexternal=this->n;
    ninternal=X[ke]->getsize();
    vtim(X[ke]->getdblock(ki), bdi);
    if (X[ke]->getcblock(ki)->getsize(1)) {
        Uqdi.copy(bdi);
        vtam(X[ke]->getcblock(ki), &Uqdi);
        bci->minus(&Uqdi);
    }
    vtim(X[ke]->getcblock(ninternal), bci);
    if (X[ke]->geteblock(ki)->getsize(1)) {
        Uqdi.copy(bdi);
        vtam(X[ke]->geteblock(ki), &Uqdi);
        bce->minus(&Uqdi);
    }
    if (X[ke]->geteblock(ninternal)->getsize(1)) {
        Uqci.copy(bci);
        vtam(X[ke]->geteblock(ninternal), &Uqci);
        bce->minus(&Uqci);
    }
    vtim(X[nexternal]->geteblock(1), bce);
    Uqci.destroy();
    Uqdi.destroy();
}

void nrbafe::emx(vetor *x)
{
    int nexternal, ninternal, i, j, n,
        nd, ndi, nc, nci, nce, md;
    vetor y, yi, xdi, xci, xce, Bx;
    double *px;

    nexternal=this->n;
    n=x->getsize();
    px=x->getline();
    nce=X[nexternal]->geteblock(1)->getsize(2);
    nci=n-ncce;
    ndi=n-ncce;
    xce.setd(px+n-ncce, nce);
    for (i=nexternal; i>0; i--) {
        ninternal=X[i]->getsize();
        nc=X[i]->getcblock(ninternal)->getsize(2);
        xci.setd(px+nci-nc, nc);
        ndi=nc;
        for (j=ninternal; j>0; j--) {
            md=X[i]->getdblock(j)->getsize(1);
            nd=X[i]->getdblock(j)->getsize(2);
            yi.resize(md);
            yi.fill(0.0);
            xdi.setd(px+ndi-nd, nd);
            if (X[i]->getdblock(j)->getsize(1)) {
                Bx.copy(&xdi);
                vam(X[i]->getdblock(j), &Bx);
                yi.plus(&Bx);
            }
            if (X[i]->getcblock(j)->getsize(1)) {
                Bx.copy(&xci);
                vam(X[i]->getcblock(j), &Bx);
                yi.plus(&Bx);
            }
            if (X[i]->geteblock(j)->getsize(1)) {
                Bx.copy(&xce);

```

```

        vam(X[i]->geteblock(j), &Bx);
        yi.plus(&Bx);
    }
    yi.vcat(&y);
    y.copy (&yi);
    ndi-=nd;
}
nci=ndi;
}
x->copy (&y);
y.destroy();
yi.destroy();
Bx.destroy();
}

void nrbafe::etmx(vetor *x)
{
    int nexternal, ninternal, i, j, n, md, nd, mx0;
    vetor xdi, Bx, y, ydi, yci, yce;
    double *px;
    mx0=0;
    n=x->getsize();
    px=x->getline();
    nexternal=this->n;
    yce.resize (X[nexternal]->geteblock(1)->getsize(2));
    yce.fill (0.0);
    for (i=1;i<=nexternal;i++) {
        ninternal=X[i]->getsize();
        yci.resize (X[i]->geteblock(ninternal)->getsize(2));
        yci.fill (0.0);
        for (j=1;j<=ninternal; j++) {
            md=X[i]->getdblock(j)->getsize(1);
            nd=X[i]->getdblock(j)->getsize(2);
            xdi.setd (px+mx0, md);
            mx0+=md;
            if (X[i]->getdblock(j)->getsize (1)) {
                ydi.copy (&xdi);
                vtam(X[i]->getdblock(j), &ydi);
            }
            if (X[i]->geteblock(j)->getsize (1)) {
                Bx.copy (&xdi);
                vtam(X[i]->geteblock(j), &Bx);
                yci.plus(&Bx);
            }
            if (X[i]->geteblock(j)->getsize (1)) {
                Bx.copy (&xdi);
                vtam(X[i]->geteblock(j), &Bx);
                yce.plus(&Bx);
            }
        }
        y.vcat (&ydi);
    }
    y.vcat(&yci);
}
y.vcat(&yce);
x->copy (&y);
y.destroy ();
Bx.destroy ();
ydi.destroy ();
yci.destroy ();
yce.destroy ();
}

double nrbafe::upo(int he, int hi, vetor *bdi, vetor *bci, vetor *bce)
{
    int nexternal, ninternal, m, nce, nci, ndi, m1, m2, i;
    double norms, *p, n2q;
    vetor qdi, qci, qce;
    nexternal=this->n;
    ninternal=X[he]->getsize();
    m=X[he]->geteblock(1)->getsize(1);
    nce=X[he]->geteblock(hi)->getsize(2);
    nci=X[he]->geteblock(hi)->getsize(2);
    ndi=X[he]->getdblock(hi)->getsize(2);

```

```

qdi.copy (bdi);
qci.copy (bci);
qce.copy (bce);
this->uitet(he, hi, &qdi, &qci, &qce);

norms=QRx(&qce,X[nexternal]->geteblock(1), NULL, NULL);
X[nexternal]->geteblock(1)->shup();
m1=X[nexternal]->geteblock(1)->getsize(1);
X[he]->geteblock(ninternal)->moverows (X[nexternal]->geteblock(1), m1);
m2=X[he]->geteblock(ninternal)->getsize(1);

X[he]->getcblock(ninternal)->resize(m2, nci);
p=X[he]->getcblock(ninternal)->getline(m2);
p++;
for (i=1;i<=nci;i++, p++) *p=0.0;
qci.resize(m2);
qci(m2)=norms;

norms=QRx(&qci, X[he]->getcblock(ninternal), X[he]->geteblock(ninternal), NULL);
X[he]->geteblock(ninternal)->shup();
m1=X[he]->geteblock(ninternal)->getsize(1);
X[he]->geteblock(hi)->moverows (X[he]->geteblock(ninternal), m1);

X[he]->getcblock(ninternal)->shup();
m1=X[he]->getcblock(ninternal)->getsize(1);
X[he]->getcblock(hi)->moverows (X[he]->getcblock(ninternal), m1);
m2=X[he]->getcblock(hi)->getsize(1);

X[he]->getdblock(hi)->resize(m2, ndi);
p=X[he]->getdblock(hi)->getline(m2);
p++;
for (i=1;i<=ndi;i++, p++) *p=0.0;
qdi.resize(m2);
qdi(m2)=norms;
norms=QRx(&qdi, X[he]->getdblock(hi), X[he]->getcblock(hi), X[he]->geteblock(hi));

qdi.destroy ();
qci.destroy ();
qce.destroy ();

p=X[he]->getdblock(hi)->getline(1);
qdi.setd (p, ndi);
bdi->cdot(norms);
bdi->minus(&qdi);
n2q= bdi->vnorm2();

p=X[he]->getcblock(hi)->getline(1);
qci.setd (p, nci);
bci->cdot(norms);
bci->minus(&qci);
n2q+= bci->vnorm2();

p=X[he]->geteblock(hi)->getline(1);
qce.setd (p, nce);
bce->cdot(norms);
bce->minus(&qce);
n2q+= bce->vnorm2();

m=X[he]->geteblock(hi)->getsize(1);
X[he]->getdblock(hi)->shup();
X[he]->getdblock(hi)->resize(m-1, ndi);
X[he]->getcblock(hi)->shup();
X[he]->getcblock(hi)->resize(m-1, nci);
X[he]->geteblock(hi)->shup();
X[he]->geteblock(hi)->resize(m-1, nce);
return (n2q);
}

int nrbafe::getsize (int dim)
/* dim=1 => numero de linhas
   dim!=1 => numero de colunas */
{

```

```

int i, j, nbe, nbi;
int nzeros, mzeros;
densa *D;
nzeros=0;
mzeros=0;
nbe=getsize();
for (i=1;i<=nbe;i++) {
    nbi=getxsize (i);
    for (j=1;j<=nbi;j++) {
        D=getdblock (i,j);
        if (D) nzeros+=D->getsize(2);
        mzeros+=geteblock (i,j)->getsize(1);
    }
    D=getcblock (i,nbi);
    if (D) nzeros+=D->getsize(2);
}
nzeros+=geteblock (nbe,nbi)->getsize(2);
if (dim==1) return (mzeros);
return (nzeros);
}

#ifdef _TESTE_NRBAF_
void main (int argc, char *argv[])
{
    FILE *fpIn, *fpOut;
    char name[80];
    nrbafe B;
    vetor c, qdi, qci, qce;
    double e;
    if (argc!=3) {
        puts ("Argument count mismatch.");
        puts ("Usage: nrbafe <inputfilename> <outputfilename>");
        exit (0);
    }
    fpIn=fopen (argv[1], "rb");
    if (!fpIn) {
        perror (argv[1]);
        exit (0);
    }
    fpOut=fopen (argv[2], "wb");
    if (!fpOut) {
        perror (argv[2]);
        exit (0);
    }
    B.cellrd (fpIn, name);
    puts (name);
    c.cellrd (fpIn, name);
    qdi.cellrd (fpIn, name);
    qci.cellrd (fpIn, name);
    qce.cellrd (fpIn, name);

    B.QR();
    // B.uitet (2, 2, &qdi, &qci, &qce);
    e=B.upo (2, 1, &qdi, &qci, &qce);

    // B.etmx (&c);
    B.cellwr (fpOut, "U");
    c.cellwr (fpOut, "x");
    qdi.cellwr (fpOut, "ydi");
    qci.cellwr (fpOut, "yci");
    qce.cellwr (fpOut, "yce");
    c.destroy ();
    B.destroy ();
    qdi.destroy ();
    qci.destroy ();
    qce.destroy ();
    fclose (fpIn);
    fclose (fpOut);
    printf ("erro=%le", e);
}
#endif

```

.....

.....

.....

.....

```

/*VETOR.CPP*/
#include <stdio.h>
"
#include <stdlib.h>
"
#include <ctype.h>
"
#include <string.h>
#include <math.h>
#include "global.hpp"
#include "vetor.hpp"

static int nAlloc=0, /* Contador de instancias alocadas. */
nInstance=0;        /* Numero de instancias criadas. */
/* A ideia com os contadores acima e' saber se temos mais memoria alocada
que instancias, ou se liberamos memoria mais de uma vez. */

vetor::vetor ()
{
    nInstance++;
    n=0;
    nmax=0;
    V=NULL;
}

vetor::~~vetor ()
{
    nInstance--;
}

void initclassvetor (void)
{
    nInstance=0;
    nAlloc=0;
}

int getvetornAlloc (void)
{
    return (nAlloc);
}

int getvetornInstance (void)
{
    return (nInstance);
}

vetor::create (int ni)
{
#ifdef _DEBUG_ENTRY_
    printf ("Criando %d+1 instancia.\n", nAlloc);
/* ja que tem que ser a primeira instrucao... */
#endif

    V=(double*) malloc (ni*sizeof (double));
#ifdef _VERIFY_ALLOC_
    if ((!V)&&(ni)) {
        printf ("Memoria insuficiente %d+1.\n", nAlloc);
        exit (0);
    }
#endif
    V--; /* sujeira para que o primeiro indice valido seja 1. */
    nAlloc++;
    n=ni;
    nmax=n;
#ifdef _DEBUG_END_
    printf ("%d instancia criada %p.\n", nAlloc, V); ' ' '
#endif
}

void vetor::destroy (void)
{

```

```

#ifdef _DEBUG_ENTRY_
    printf ("Removendo %d instancia %p.\n", nAlloc, V);
#endif
    if (V) {
        V++;
#ifdef _DEBUG_ALLOC_
        printf ("Liberando vetor %p.\n", V);
#endif
        free (V);
    }
    nAlloc--;
    n=0;
    nmax=0;
    V=NULL;
#ifdef _DEBUG_END_
    printf ("%d instancia removida.\n", nAlloc);
#endif
}

double &vetor::operator () (int i) .
{
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>n)) {
        printf ("%p(%d) fora dos limites do vetor.\n", V, i);
        exit (0);
    }
#endif
    return (V[i]);
}

void vetor::fprint (FILE *fp, const char *name)
{
    double *p;
    int i;
    fprintf (fp, "%s=[\n", name);
    p=V;
    p++;
    for (i=1;i<n;i++, p++) fprintf (fp, "%9le\t", *p);
    fprintf (fp, "%9le];\n", *p);
}

char *vetor::fread (FILE *fp, char *name)
{
    int c, in;
    long pos;
    int fisnum, j;
    char *pname;
    double *linha;
    fisnum=0;
    in=0;
    pname=name;
    do {
        c=fgetc (fp);
        if (isalnum(c)) *pname++=c;
    } while ((!feof (fp)) && (c!='('));
    *pname='\0';
    pos=ftell (fp);
    while ((c!=']')&&(!feof(fp))) {
        c=fgetc(fp);
        switch (c) {
            case '\t':
            case '\n':
            case ',':
            case ']':
            case ';':
            case ' ': if (fisnum) in++; fisnum=0; break;
            default: if (isdigit (c) || (c=='e') || (c=='E') || (c=='.')) || (c=='+') || (c=='-'))
                fisnum=1;
            else fisnum=0;
        }
    }
    if (fseek (fp, pos, SEEK_SET)) {
        printf ("Parse error.");
    }
}

```

```

        exit (0);
    }
    this->resize (in);
    linha=V;
    linha++;
    for (j=0;j<(in-1);j++, linha++) {
        if (fscanf (fp, " %lf ", linha)!=1) {
            printf ("syntax error.");
            exit (0);
        }
    }
    if (fscanf (fp, " %lf ]; ", linha)!=1) {
        printf ("syntax error.");
        exit (0);
    }
    return (name);
}

void vetor::resize (int nn)
{
    double *newline;      /* ponteiro para linha realocada. */
#ifdef _DEBUG_ENTRY_
    printf ("Redimensionando vetor %p\n", this);
#endif
    if (!V) this->create (nn);
    if (nn>nmax) {
#ifdef _DEBUG_ALLOC_
        printf ("Realocando vetor %p\n", V);
#endif
        newline=(double *) realloc (V+1, nn*sizeof (double));
#ifdef _VERIFY_ALLOC_
        if (!newline) {
            printf ("Memoria insuficiente %d+1.\n", nAlloc);
            exit (0);
        }
#endif
        V=newline-1; /* sujeira para que o primeiro indice valido seja 1. */
        nmax=nn;
    }
    n=nn;

#ifdef _DEBUG_END_
    printf ("%p Realocado.\n", V);
#endif
}

double *vetor::getline (void)
{
    return (V);
}

double **vetor::getpline (void)
{
    return (&V);
}

int vetor::getsize (void)
{
    return (n);
}

int vetor::getmaxsize (void)
{
    return (nmax);
}

void vetor::copy (vetor *a)
{
    double *av;
    this->resize (a->getsize()); /* nao reproduz exatamente o vetor a, ja que
                                   ele pode ter tamanho maximo diferente do
                                   tamanho atual, mas isto pode funcionar ate
                                   como um simples garbage collector manual... */
}

```



```

        av=a->getline();
        memcpy (V+1, av+1, n*sizeof(double));
    }

void vetor::matprt (FILE *fp, const char *name)
{
    int j;
    long type, mrows, ncols, imagf, namelen;
    double *p;

    mrows=1;
    ncols=n;          /* por causa do cast implicitito. */
    type=0;           /* Little Endian, IEEE double, densa. */
    imagf=0;          /* matriz real */
    namelen=strlen (name)+1;
    fwrite (&type, sizeof (long), 1, fp);
    fwrite (&mrows, sizeof (long), 1, fp);
    fwrite (&ncols, sizeof (long), 1, fp);
    fwrite (&imagf, sizeof (long), 1, fp);
    fwrite (&namelen, sizeof (long), 1, fp);
    fwrite (name, sizeof (char), namelen, fp);
    p=V+1;
    for (j=1;j<=n;j++, p++) fwrite (p, sizeof (double), 1, fp);
}

int vetor::update (double val, int i, int j)
{
    double *pj, *pi, *src, *dest;
    int len;
#ifdef _VERIFY_BOUNDS_
    if ((j<=0)|| (j>(n+1))|| (i<=0)|| (i>(n+1))) {
        printf ("%p(%d) ou %p(%d) fora dos limites do vetor.\n", V, j, V, i);
        exit (0);
    }
#endif
    len=0; src=NULL; dest=NULL;
    pi=V+i;
    pj=V+j;
    if (pj>pi) {
        src=pi;
        dest=pi+1;
        len=pj-pi;
    }
    else {
        if (pj<pi) {
            pi=pi-1;
            src=pj+1;
            dest=pj;
            len=pi-pj;
        }
    }
    memmove (dest, src, len*sizeof(double));
    *pi=val;
    return (0);
}

void vetor::minus (vetor *v)
{
    double *v1, *v2;
    int i;
    if (n!=v->getsize()) {
        printf ("%p minus: dimensions do not agree.", this->V);
        exit (0);
    }
    v1=this->V+1;
    v2=v->V+1;
    for (i=1;i<=n;i++, v1++, v2++) (*v1)-=(*v2);
}

double vetor::prod(vetor *v)
{
    int i;
    double r, *a, *b;

```

```

#ifdef _VERIFY_BOUNDS_
    if (n!=v->n) {
        printf ("%p prod: dimensi do not agree.", V);
        exit (0);
    }
#endif
    r=0; a=V+1; b=v->V+1;
    for (i=1;i<=n;i++, a++, b++) r+=((*a)*(*b));
    return (r);
}

void vetor::div(vetor *v)
{
    int i;
    double *a, *b;
#ifdef _VERIFY_BOUNDS_
    if (n!=v->n) {
        printf ("%p prod: dimensi do not agree.", V);
        exit (0);
    }
#endif
    a=V+1; b=v->V+1;
    for (i=1;i<=n;i++, a++, b++) (*a)/=(*b);
}

int vetor::max(void)
{
    int i, imax;
    double *a, max;
    if (!n) return (0);
    a=V+1; max=*a; imax=1;
    a++;
    for (i=2;i<=n;i++, a++) if ((*a)>max) { max=*a; imax=i; }
    return (imax);
}

int vetor::min(void)
{
    int i, imin;
    double *a, min;
    if (!n) return (0);
    a=V+1; min=*a; imin=1;
    a++;
    for (i=2;i<=n;i++, a++) if ((*a)<min) { min=*a; imin=i; }
    return (imin);
}

int vetor::first(void)
{
    int i;
    double *a;
    if (!n) return (0);
    a=V+1;
    for (i=1;i<=n;i++, a++) if ((*a)>0.0) return (i);
    return (0);
}

void vetor::cdot (double c)
{
    double *v1;
    int i;
    v1=this->V+1;
    for (i=1;i<=n;i++, v1++) (*v1)*=c;
}

void vetor::cplus (double c)
{
    double *v1;
    int i;
    v1=this->V+1;
    for (i=1;i<=n;i++, v1++) (*v1)+=c;
}

```

```

void vetor::fill (double a)
{
    double *v1;
    int i;
    v1=this->V+1;
    for (i=1;i<=n;i++, v1++) (*v1)=a;
}

void vetor::dot (vetor *v)
{
    double *v1, *v2;
    int i;
#ifdef _VERIFY_BOUNDS_
    if (n!=v->getsize()) {
        printf ("%p dot: dimensions do not agree.", this->V);
        exit (0);
    }
#endif
    v1=this->V+1;
    v2=v->V+1;
    for (i=1;i<=n;i++, v1++, v2++) (*v1)*=(*v2);
}

void vetor::plus (vetor *v)
{
    double *v1, *v2;
    int i;
#ifdef _VERIFY_BOUNDS_
    if (n!=v->getsize()) {
        printf ("%p plus: dimensions do not agree.", this->V);
        exit (0);
    }
#endif
    v1=this->V+1;
    v2=v->V+1;
    for (i=1;i<=n;i++, v1++, v2++) (*v1)+=(*v2);
}

void vetor::vcat (vetor *v)
{
    double *v1, *v2;
    int i, n0;
    n0=n;
    this->resize (n+v->n);
    v2=v->V+1;
    v1=this->V+1+n0;
    for (i=v->n;i>0;i--, v1++, v2++) (*v1)=(*v2);
}

double vetor::vnorm2 (void)
/* norma 2 do vetor */
{
    int i;
    double norm2, *p;
    norm2=0.0;
    p=V+1;
    for (i=1;i<=n;i++, p++) norm2+=(*p)*(*p);
    return (sqrt(norm2));
}

void vetor::cellwr (FILE *fp, const char *name)
{
    int j;
    long type, mrows, ncols;

    mrows=n;
    ncols=1; /* por causa do cast implicit. */
    type=T_MATRIX;
    fprintf (fp, "%s%c", name, '\0');
    fwrite (&type, sizeof (long), 1, fp);
    fwrite (&mrows, sizeof (long), 1, fp);
    fwrite (&ncols, sizeof (long), 1, fp);
    for (j=1;j<=n;j++) fwrite (&(V[j]), sizeof (double), 1, fp);
}

```

```

}

int vetor::cellrd (FILE *fp, char *name)
{
    int i, j, mm;
    long type, mrows, ncols;

    i=0;
    do {
        name[i]=fgetc(fp);
    } while (name[i++]!='\n');

    ::fread (&type, sizeof (long), 1, fp);
    ::fread (&mrows, sizeof (long), 1, fp);
    ::fread (&ncols, sizeof (long), 1, fp);
    if (type!=T_MATRIX) {
        printf ("vetor::cellrd: type mismatch.\n");
        exit (0);
    }
    if ((mrows>1)&&(ncols>1)) {
        printf ("vetor::cellrd: dimension mismatch (%d, %d).\n", mrows, ncols);
        exit (0);
    }
    mm=(mrows==1)?ncols:mrows;
    this->resize (mm);
    for (j=1;j<=mm;j++) ::fread (&(V[j]), sizeof (double), 1, fp);
    return (0);
}

```

```

/*IVETOR.CPP*/

/*
 * Classe vetor de indices para matrizes na forma NRBAF
 * Fabio Nakano - 10/02/99
 */

#include "ivetor.hpp"
#include "vetor.hpp"

// #define _TEST_IVETOR_

static int nAlloc=0, /* Contador de instancias alocadas. */
nInstance=0; /* Numero de instancias criadas. */
/* A ideia com os contadores acima e' saber se temos mais memoria alocada
que instancias, ou se liberamos memoria mais de uma vez. */

ivetor::ivetor ()
{
    nInstance++;
    n=0;
    nmax=0;
    t=NULL;
}

ivetor::~ivetor ()
{
    nInstance--;
}

void initclassivetor (void)
{
    nInstance=0;
    nAlloc=0;
}

int getivetornAlloc (void)
{
    return (nAlloc);
}

int getivetornInstance (void)
{
    return (nInstance);
}

ivetor::create (nrbafe *A)
/* A matriz nrbafe que gera este vetor de indices tem que ser a matriz que
define o PPL ou uma outra com a mesma quantidade de blocos internos e externos.
Desta forma, construir a base e a matriz de coeficientes de variaveis residuais
significa remover variaveis. Esta e a solucao que encontrei para nao precisar tratar
a insercao de novos blocos (internos ou externos). */
{
    int ni, nik, nbi, nbik, nbe, i, j, k;
    tripla *pt;
#ifdef _DEBUG_ENTRY_
    printf ("Criando %d+1 instancia.\n", nAlloc);
/* ja que tem que ser a primeira instrucao... */
#endif
    nbi=0;
    ni=0;
    nbe=A->getsize();
    for (i=1; i<=nbe; i++) {
        nbik=A->getblock(i)->getsize();
        nbi+=nbik;
        for (j=1; j<=nbik; j++) {
            ni+=A->getblock(i)->getblock(j)->getsize(1);
        }
    }

    xtag.create(nbe+1);
    tag.create(nbi+1);
}

```

```

        t=(tripla*) malloc ((ni+1)*sizeof (tripla));
#ifdef _VERIFY_ALLOC_
        if (!t) {
            printf ("Memoria insuficiente %d+1.\n", nAlloc);
            exit (0);
        }
#endif
        pt=t;
        t--; /* sujeira para que o primeiro indice valido seja 1. */
        nAlloc++;
        n=ni;
        nmax=ni+1;

/* posicionando os tags de acordo com a matriz. */
        nbi=0;
        ni=0;
        nbe=A->getsize();
        xtag.V[1]=(tripla **)tag.V;
        tag.V[1]=t+1;
        for (i=1;i<=nbe;i++) {
            nbik=A->getblock(i)->getsize();
            xtag.V[i+1]=((tripla **)xtag.V[i])+nbik;
            for (j=1;j<=nbik;j++) {
                nik=A->getblock(i)->getblock(j)->getsize(1);
                tag.V[nbi+j+1]=((tripla *)tag.V[nbi+j])+nik;
                for (k=1;k<=nik;k++, pt++) {
                    pt->be=i;
                    pt->bi=j;
                    pt->i=k;
                }
                ni+=nik;
            }
            nbi=nbi+nbik;
        }
        pt->be=nbe+1;
        pt->bi=1;
        pt->i=1;
//      xtag.V[nbe+1]=(tripla **>(&tag.V[nbi+1]));
//      tag.V[nbi+1]=&(t[n]);
#ifdef _DEBUG_END_
        printf ("%d instancia criada %p.\n", nAlloc, tag);
#endif
    }

void ivetor::destroy (void)
{
#ifdef _DEBUG_ENTRY_
    printf ("Removendo %d instancia %p.\n", nAlloc, t);
#endif
    t++;
#ifdef _DEBUG_ALLOC_
    printf ("Liberando vetor %p.\n", t);
#endif
    free (t);
    xtag.destroy ();
    tag.destroy ();
    nAlloc--;
    n=0;
    nmax=0;
    t=NULL;
#ifdef _DEBUG_END_
    printf ("%d instancia removida.\n", nAlloc);
#endif
}

int ivetor::getblocksize (int be, int bi)
{
//      return (((tripla *) (xtag[be][bi+1]))-((tripla *) (xtag[be][bi])));
    return (*((tripla **) (xtag.V[be]))+bi+1)-*((tripla **) (xtag.V[be])+bi));
}

int ivetor::getsize (void)
{

```

```

    return (n);
}

int ivetor::getmaxsize (void)
{
    return (nmax);
}

void ivetor::txtrd (FILE *fp)
{
    int nbe, nbi, i;
    fscanf (fp, "%d %d %d ", &nbe, &nbi, &n);
    if (n>nmax) {
        printf ("ivetor: Too many rows.");
        exit (0);
    }
    for (i=1;i<=n;i++) {
        if (fscanf (fp, "%d %d %d ", &t[i].be, &t[i].bi, &t[i].i)!=3) {
            printf ("ivetor: Bad input (missing rows).");
            exit (0);
        }
    }
    n--;
    arrangetags ();
}

void ivetor::txtwr (FILE *fp)
{
    int nbe, nbi, i;
    nbe=xtag.getsize();
    nbi=tag.getsize();
    fprintf (fp, "%d %d %d\n\n", nbe, nbi, n+1);
    for (i=1;i<=(n+1);i++) {
        fprintf (fp, "%d %d %d\n", t[i].be, t[i].bi, t[i].i);
    }
    /*
    fputs ("tags:\n", fp);
    for (i=1;i<=nbi;i++) {
        fprintf (fp, "%d %d %d\n", ((tripla *) (tag.V[i]))->be, ((tripla *) (tag.V[i]))->bi, ((tripla *) (tag.V[i]))->i);
    }
    fputs ("xtags:\n", fp);
    for (i=1;i<=nbe;i++) {
        fprintf (fp, "%d %d %d\n", ((tripla *) (((tripla **) {xtag.V[i]})[1]))->be, ((tripla *) (((tripla **) {xtag.V[i]})[1]))->bi, ((tripla *) (((tripla **) {xtag.V[i]})[1]))->i);
    }
    */
}

int ivetor::update1 (int be, int bi, int indice, int j)
/* retorna o indice sequencial do elemento que entra.
   o indice sequencial do elemento que sai e' j. */
{
    tripla *pj, *pi, *src, *dest;
    tripla **ptag;
    int len, r;
#ifdef _VERIFY_BOUNDS_
    if ((j<=0)|| (j>n)) {
        printf ("%p(%d) fora dos limites do vetor.\n", tag, j);
        exit (0);
    }
#endif
    len=0; src=NULL; dest=NULL;
    ptag=((tripla **) (xtag.V[be]))+bi;
    pi=((tripla *) (((tripla **) (xtag.V[be]))[bi])); r=pi-t;
    pj=&(t[j]);
    if (pj>pi) {
        src=pi;
        dest=pi+1;
        len=pj-pi;
        ptag++;
        while (*ptag<=pj) {
            (*ptag)++;
        }
    }
}

```

```

        ptag++;
    }
}
else {
    if (pj<pi) {
        pi=pi-1;
        src=pj+1;
        dest=pj;
        len=pi-pj;
        while (*ptag>pj) {
            (*ptag)--;
            ptag--;
        }
    }
    // else tag[pj->b]=pj;
}
memmove (dest, src, len*sizeof(tripla));
pi->be=be;
pi->bi=bi;
pi->i=indice;
return (r);
}

int ivetor::updaten (int be, int bi, int indice, int j)
/* retorna o indice sequencial do elemento que entra. */
{
    tripla *pj, *pi, *src, *dest;
    int len, r;
    tripla **ptag;
#ifdef _VERIFY_BOUNDS_
    if ((j<=0) || (j>n)) {
        printf ("%p(%d) fora dos limites do vetor.\n", tag, j);
        exit (0);
    }
#endif
    len=0; src=NULL; dest=NULL;
    ptag=((tripla **) (xtag.V[be]))+bi+1;
    pi=((tripla **) (xtag.V[be]))+bi+1; r=pi-t;
    pj=&(t[j]);
    if (pj>pi) {
        src=pi;
        dest=pi+1;
        len=pj-pi;
        ptag++;
        // while (*ptag<=pj) {
        //     (*ptag)++;
        //     ptag++;
        // }
    }
    else {
        if (pj<pi) {
            pi=pi-1;
            src=pj+1;
            dest=pj;
            len=pi-pj;
            ptag--;
            while (*ptag>pj) {
                (*ptag)--;
                ptag--;
            }
        }
        else (*ptag)++;
    }
    memmove (dest, src, len*sizeof(tripla));
    pi->be=be;
    pi->bi=bi;
    pi->i=indice;
    return (r);
}

void ivetor::arrangetags (void)
{
    int i;

```



```

    tripla **ptag1, **ptag2;
    tripla *pt1, *pt2;
    ((tripla **)(xtag.V[1]))[1]=t+1;
    for (i=1, pt1=t+1, pt2=t+2; i<(n+1); i++, pt1++, pt2++) {
        if ((pt1->bi!=pt2->bi) || (pt1->be!=pt2->be)) {
            ptag1=(tripla **) (xtag.V[pt1->be])+pt1->bi+1;
            ptag2=(tripla **) (xtag.V[pt2->be])+pt2->bi;
            *ptag2=pt2;
            while (ptag1<ptag2) {
                *ptag1=*ptag2;
                ptag1++;
            }
        }
    }
}

int ivetor::blockpos(int i)
{
    tripla *p, *p0;
#ifdef _VERIFY_BOUNDS_
    if ((i<=0) || (i>n)) {
        printf ("ivetor %p(%d) acesso fora dos limites do vetor.", tag, i);
        exit (0);
    }
#endif
    p=t+i;
    p0=((tripla *)(((tripla **) (xtag.V[t[i].be)))[t[i].bi]));
    return (p-p0+1);
}

int &ivetor::operator () (int i, int j, int k)
{
    return (((tripla *)(((tripla **) (xtag.V[i]))[j]))+k-1->i);
}

tripla &ivetor::operator () (int i)
{
    return (t[i]);
}

#ifdef _TEST_IVETOR_
void main (void)
{
    FILE *fpOut, *fpIn;
    nrba b;
    int j=30, be=2, bi=1, indice=17;
    ivetor b, c;
    vetor d;
    char name[30];

    fpIn=fopen ("auxm\\etmx.bin", "rb");
    B.cellrd (fpIn, name);
    fclose (fpIn);

    b.create (&B);
    fpIn=fopen ("auxm\\base.txt", "rt");
    b.txtrd (fpIn);
    fclose (fpIn);

    // b.updaten (be, bi, indice, j);
    fpOut=fopen ("auxm\\ttt.txt", "wt");
    b.txtwr (fpOut);
    fclose (fpOut);
}
#endif

```

```

/*VVETOR.CPP*/

"
#include "vvetor.hpp"
"

void vvetor::create (int ni)
{
    V=(void **) malloc (ni*sizeof (void *));
#ifdef _VERIFY_ALLOC_
    if ((!V)&&(ni)) {
        printf ("vvetor:Memoria insuficiente.\n");
        exit (0);
    }
#endif
    V--; /* sujeira para que o primeiro indice valido seja 1. */
    n=ni;
    nmax=n;
#ifdef _DEBUG_END_
    printf ("vvetor: instancia criada %p.\n", V);
#endif
}

void vvetor::destroy (void)
{
#ifdef _DEBUG_ENTRY_
    printf ("vvetor: Removendo instancia %p.\n", V);
#endif
    V++;
    if (n) {
#ifdef _DEBUG_ALLOC_
        printf ("vvetor: Liberando vetor %p.\n", V);
#endif
        free (V);
    }
    n=0;
    nmax=0;
    V=NULL;
#ifdef _DEBUG_END_
    printf ("vvetor: instancia removida.\n");
#endif
}

void * &vvetor::operator () (int i)
{
#ifdef _VERIFY_BOUNDS_
    if ((i<=0)|| (i>n)) {
        printf ("%p(%d) fora dos limites do vvetor.\n", V, i);
        exit (0);
    }
#endif
    return (V[i]);
}

void vvetor::resize (int nn)
{
    void **newline; /* ponteiro para linha realocada. */
#ifdef _DEBUG_ENTRY_
    printf ("vvetor: Redimensionando vetor %p\n", this);
#endif
    if (!V) this->create (nn);
    if (nn>nmax) {
#ifdef _DEBUG_ALLOC_
        printf ("vvetor: Realocando vetor %p\n", V);
#endif
        newline=(void **) realloc (V+1, nn*sizeof (void *));
#ifdef _VERIFY_ALLOC_
        if (!newline) {
            printf ("vvetor: Memoria insuficiente.\n");
            exit (0);
        }
    }
}

```

```

    }
#endif
    V=newline-1; /* sujeira para que o primeiro indice valido seja 1. */
    nmax=nn;
    }
    n=nn;

#ifdef _DEBUG_END_
    printf ("%p Realocado.\n", V);
#endif
}

```

```

/*SIMPLEX.CPP*/

"
#include <malloc.h>
"
#include <math.h>
"
#include "nrbafe.hpp"
"
#include "ivetor.hpp"
"
#include "vetor.hpp"
"

#define EPS 1e-15
"

#define _TESTE_COMPLICADO_
"

void heap_dump(int ipassos);

int monta (nrbafe *A, nrbafe *B, vetor *c, vetor *cb, ivetor *b)
/* Monta B e cb selecionando de A e c os elementos listados em b.
*/
{
    int i, j, k, l, nbe, nbi;
    int icb, offsetc;
    int m, ndi, nci, nce;
    double *psrc, *pdest;
    nrbafe *X;
    vetor src, dest;
    nbe=A->getsize();
    B->destroy ();
    B->resize (nbe);
    offsetc=0;
    icb=1;
    cb->resize (b->getsize());
    for (i=1;i<=nbe;i++) {
        nbi=A->getxsize (i);
        B->createblock(i, nbi);
        X=B->getblock (i);
        for (j=1;j<=nbi;j++) {
            m=b->getblocksize(i,j);
            ndi=A->getdblock (i,j)->getsize(2);
            nci=A->getcblock (i,j)->getsize(2);
            nce=A->geteblock (i,j)->getsize(2);
            X->createdblock (j, m, ndi);
            X->createcblock (j, m, nci);
            X->createeblock (j, m, nce);
            for (k=1;k<=m;k++, icb++) {
                l=(*b) (i,j,k);
                (*cb) (icb)=(*c) (offsetc+l);
                pdest=B->getdblock (i,j)->getline(k);
                psrc=A->getdblock (i,j)->getline(1);
                dest.setd (pdest, ndi);
                src.setd (psrc, ndi);
                dest.copy (&src);
                pdest=B->getcblock (i,j)->getline(k);
                psrc=A->getcblock (i,j)->getline(1);
                dest.setd (pdest, nci);
                src.setd (psrc, nci);
                dest.copy (&src);
                pdest=B->geteblock (i,j)->getline(k);
                psrc=A->geteblock (i,j)->getline(1);
                dest.setd (pdest, nce);
                src.setd (psrc, nce);
                dest.copy (&src);
            }
            offsetc+=A->geteblock(i,j)->getsize(1);
        }
    }
}

```

```

    }
    return (0);
}

int linha (nrbaif *A, int be, int bi, int indice, vetor *v)
/* em v a linha densa correspondente ao bloco "bloco" linha "indice" de A. */
{
    int i, j, nbe, nbi, nq;
    int nzeros;
    vetor z, q;
    double *Vq;
    nzeros=0;
    v->destroy ();
    nbe=A->getsize();
    for (i=1;i<be;i++) {
        nbi=A->getxsize (i);
        for (j=1;j<=nbi;j++)
            nzeros+=A->getdblock (i,j)->getsize(2);
        nzeros+=A->getcblock (i,nbi)->getsize(2);
    }
    for (j=1;j<bi;j++)
        nzeros+=A->getdblock (be,j)->getsize(2);
    z.resize (nzeros);
    z.fill (0.0);
    nq=A->getdblock (be,bi)->getsize(2);
    Vq=A->getdblock (be,bi)->getline(indice);
    q.setd (Vq, nq);
    z.vcat(&q);

    nbi=A->getxsize (be); //
    nzeros=0;
    for (j=bi+1;j<=nbi;j++)
        nzeros+=A->getdblock (be,j)->getsize(2);
    q.setd (NULL, 0);
    q.resize (nzeros);
    q.fill (0.0);
    z.vcat(&q);
    q.destroy ();
    nq=A->getcblock (be,bi)->getsize(2);
    Vq=A->getcblock (be,bi)->getline(indice);
    q.setd (Vq, nq);
    z.vcat(&q);
    nzeros=0;
    for (i=be+1;i<=nbe;i++) {
        nbi=A->getxsize (i);
        for (j=1;j<=nbi;j++)
            nzeros+=A->getdblock (i,j)->getsize(2);
        nzeros+=A->getcblock (i,nbi)->getsize(2);
    }
    q.setd (NULL, 0);
    q.resize (nzeros);
    q.fill (0.0);
    z.vcat(&q);
    q.destroy ();
    nq=A->geteblock (be,bi)->getsize(2);
    Vq=A->geteblock (be,bi)->getline(indice);
    q.setd (Vq, nq);
    z.vcat(&q);
    v->copy (&z);
    z.destroy();
    return (0);
}

void _check(int ipassos)
{
    heap_dump (ipassos);
    switch( _heapchk() ) {
        case _HEAPOK:
            printf( "OK - heap is good\n" );
            break;
        case _HEAPEMPTY:

```

```

        printf( "OK - heap is empty\n" );
        break;
    case _HEAPBADBEGIN:
        printf( "ERROR - heap is damaged. %d\n", ipassos );
        exit (0);
        break;
    case _HEAPBADNODE:
        printf( "ERROR - bad node in heap. %d\n", ipassos );
        exit (0);
        break;
    }
}

void heap_dump(int ipassos)
{
    struct _heapinfo h_info;
    int heap_status;

    h_info._pentry = NULL;
    for(;;) {
        heap_status = _heapwalk( &h_info );
        if( heap_status != _HEAPOK ) break;
        printf( " %s block at %Fp of size %4.4X\n",
            (h_info._useflag == _USEDENTRY ? "USED" :
"FREE"),
            h_info._pentry, h_info._size );
    }

    switch( heap_status ) {
    case _HEAPEND:
        printf( "OK - end of heap\n" );
        break;
    case _HEAPEMPTY:
        printf( "OK - heap is empty\n" );
        break;
    case _HEAPBADBEGIN:
        printf( "ERROR - heap is damaged. %d\n", ipassos );
        exit (0);
        break;
    case _HEAPBADPTR:
        printf( "ERROR - bad pointer to heap. %d\n", ipassos );
        exit (0);
        break;
    case _HEAPBADNODE:
        printf( "ERROR - bad node in heap. %d\n", ipassos );
        exit (0);
    }
}

int min (vetor *dtil, vetor *rtilj)
{
    int i, n, imin;
    double *pdtil, *prtilj, min;
    imin=0; min=1e308;
    n=dtil->getsize();
    pdtil=dtil->getline()+1;
    prtilj=rtilj->getline()+1;
    for (i=1;i<=n;i++, pdtil++, prtilj++) {
        if (*prtilj>0.0) {
            if ((*pdtil/(*prtilj))<min) {
                min=(*pdtil/(*prtilj));
                imin=i;
            }
        }
    }
    return (imin);
}

void swap (nrbafe *B, int ibe, int ibi, int tiraB, nrbafe *R, int jbe, int jbi, int tiraR,
    vetor *jd, vetor *jc, vetor *je, vetor *id, vetor *ic, vetor *ie)
/* remove de B a linha correspondente a tiraB do bloco ib e faz uma copia em id..ic;
   remove de R a linha correspondente a tiraR do bloco jb e faz uma copia em jd..jc;

```

```

insere em B, como primeira linha do bloco jb a linha tirada de R;
insere em R, como ultima linha do bloco ib a linha tirada de B.
*/
{
    double *bd, *bc, *be,
           *rd, *rc, *re,
           *aux, /* para copia dos vetores. */
           **pp;
    densa *D, *C, *E;
    int nd, nc, ne, m, i;

/* Tira da base. */
    D=B->getdblock (ibe, ibi);
    nd=D->getsize(2);
    m=D->getsize(1);
    pp=D->getpline(tiraB);
    bd=*pp;
    for (i=tiraB;i<m;i++, pp++) *pp=*(pp+1);
    *pp= ((double*) malloc (nd*sizeof(double)))-1;
    D->resize (m-1, nd);

    C=B->getcblock (ibe, ibi);
    nc=C->getsize(2);
    pp=C->getpline(tiraB);
    bc=*pp;
    for (i=tiraB;i<m;i++, pp++) *pp=*(pp+1);
    *pp= ((double*) malloc (nc*sizeof(double)))-1;
    C->resize (m-1, nc);

    E=B->geteblock (ibe, ibi);
    ne=E->getsize(2);
    pp=E->getpline(tiraB);
    be=*pp;
    for (i=tiraB;i<m;i++, pp++) *pp=*(pp+1);
    *pp= ((double*) malloc (ne*sizeof(double)))-1;
    E->resize (m-1, ne);

/* Faz copias: id=bd; ic=bc */
    id->resize (nd);
    aux=id->getline();
    memcpy (aux+1, bd+1, nd*sizeof(double));

    ic->resize (nc);
    aux=ic->getline();
    memcpy (aux+1, bc+1, nc*sizeof(double));

    ie->resize (ne);
    aux=ie->getline();
    memcpy (aux+1, be+1, ne*sizeof(double));
/* Tira de R. */
    D=R->getdblock (jbe, jbi);
    nd=D->getsize(2);
    m=D->getsize(1);
    pp=D->getpline(tiraR);
    rd=*pp;
    for (i=tiraR;i<m;i++, pp++) *pp=*(pp+1);
    *pp= ((double*) malloc (nd*sizeof(double)))-1;
    D->resize (m-1, nd);

    C=R->getcblock (jbe, jbi);
    nc=C->getsize(2);
    pp=C->getpline(tiraR);
    rc=*pp;
    for (i=tiraR;i<m;i++, pp++) *pp=*(pp+1);
    *pp= ((double*) malloc (nc*sizeof(double)))-1;
    C->resize (m-1, nc);

    E=R->geteblock (jbe, jbi);
    ne=E->getsize(2);
    pp=E->getpline(tiraR);
    re=*pp;
    for (i=tiraR;i<m;i++, pp++) *pp=*(pp+1);
    *pp= ((double*) malloc (ne*sizeof(double)))-1;

```

```

    E->resize (m-1, ne);
/* Faz copias: jd=rd; jc=rc */
    jd->resize (nd);
    aux=jd->getline();
    memcpy (aux+1, rd+1, nd*sizeof(double));

    jc->resize (nc);
    aux=jc->getline();
    memcpy (aux+1, rc+1, nc*sizeof(double));

    je->resize (ne);
    aux=je->getline();
    memcpy (aux+1, re+1, ne*sizeof(double));
/* Poe em B */
    D=B->getdblock (jbe, jbi);
    nd=D->getsize(2);
    m=D->getsize(1);
    D->resize(m+1, nd);
    D->shdown();
    pp=D->getpline(1);
    free (*pp+1);
    *pp=rd;

    C=B->getcblock (jbe, jbi);
    nc=C->getsize(2);
    m=C->getsize(1);
    C->resize(m+1, nc);
    C->shdown();
    pp=C->getpline(1);
    free (*pp+1);
    *pp=rc;

    E=B->geteblock (jbe, jbi);
    ne=E->getsize(2);
    m=E->getsize(1);
    E->resize(m+1, ne);
    E->shdown();
    pp=E->getpline(1);
    free (*pp+1);
    *pp=re;
/* Poe em R */
    D=R->getdblock (ibe, ibi);
    nd=D->getsize(2);
    m=D->getsize(1);
    D->resize(m+1, nd);
    pp=D->getpline(m+1);
    free (*pp+1);
    *pp=bd;

    C=R->getcblock (ibe, ibi);
    nc=C->getsize(2);
    m=C->getsize(1);
    C->resize(m+1, nc);
    pp=C->getpline(m+1);
    free (*pp+1);
    *pp=bc;

    E=R->geteblock (ibe, ibi);
    ne=E->getsize(2);
    m=E->getsize(1);
    E->resize(m+1, ne);
    pp=E->getpline(m+1);
    free (*pp+1);
    *pp=be;
}

#ifdef _TESTE_COMPLICADO_
void main (void)
{
    FILE *fpIn, *fpOut;
    char name[80];
    nrbafe A, B, R, U;
    ivetor b, r;

```



```

vetor c, d, cb, cr, z, dtil, rtilj;
vetor jd, jci, jce, id, ici, ice;
int i, ibe, ibi, ii,
    j, jbe, jbi, ji,
    indice, irinv;
int npassos, ipassos;
double vcb, vcr, n2q;
fpIn=fopen ("auxm\\simplex.bin", "rb");
if (!fpIn) {
    perror ("auxm\\simplex.bin");
    exit (0);
}
A.cellrd (fpIn, name);
c.cellrd (fpIn, name);
d.cellrd (fpIn, name);
fclose (fpIn);

fpIn=fopen ("auxm\\b.txt", "rt");
if (!fpIn) {
    perror ("auxm\\b.txt");
    exit (0);
}
b.create (&A);
b.txtrd (fpIn);
fclose (fpIn);

fpIn=fopen ("auxm\\r.txt", "rt");
if (!fpIn) {
    perror ("auxm\\r.txt");
    exit (0);
}
r.create (&A);
r.txtrd (fpIn);
fclose (fpIn);
//    monta (&A, &B, &c, &cb, &b);

/* aqui eu sei o tamanho do problema!! */
monta (&A, &B, &c, &cb, &b);
monta (&A, &R, &c, &cr, &r);
n2q=1;
irinv=-1;
//    U.copy (&B);
//    U.QR();

jce.create(10);
jci.create(10);
jd.create(10);
ice.create(10);
ici.create(10);
id.create(10);

npassos=5000;
ipassos=0;
while (npassos>ipassos) {
    ipassos++;

//    monta (&A, &B, &c, &cb, &b);
//    monta (&A, &R, &c, &cr, &r);

    fpOut=fopen ("base.out", "at");
    if (!fpOut) {
        perror ("base.out");
        exit (0);
    }
    fprintf (fpOut, "passo=%d\n", ipassos);
    fputs ("b=", fpOut);
    b.txtwr (fpOut);
    fputs ("];\nr=", fpOut);
    r.txtwr (fpOut);
    fputs ("];", fpOut);
    fclose (fpOut);

    if (fabs(n2q)>EPS) {

```

```

        U.copy (&B);
        U.QR();
        printf ("U(%d, %d)\n", U.getsize(1), U.getsize(2));
        printf ("Reinversao\n");
        irinv++;
    }
    z.copy (&cb);
    B.etmx(&z);
    U.utix(&z);
    U.uix(&z);
    R.emx(&z);
    z.minus (&cr);
    j=z.max();

//    j=z.first();
    jbe=r(j).be;
    jbi=r(j).bi;
    ji=r(j).i;
    vcr=cr(j);

    linha (&A, jbe, jbi, ji, &rtilj);
    dtil.copy (&d);
    U.utix (&dtil);
    U.uix (&dtil);
    B.emx (&dtil);

    if (z(j)<1e-12) {
        fpOut=fopen ("base.out", "at");
        if (!fpOut) {
            perror ("base.out");
            exit (0);
        }
        fprintf (fpOut, "solucao otima\n");
        fputs ("b=[", fpOut);
        b.txtwr (fpOut);
        fputs ("]", fpOut);
        dtil.fprint (fpOut, "xb");
        fclose (fpOut);
        fpOut=fopen ("otimo.bin", "wb");
        if (!fpOut) {
            perror ("otimo.bin");
            exit (0);
        }
        dtil.cellwr (fpOut, "xb");
        fclose (fpOut);
        break;
    }

    U.utix (&rtilj);
    U.uix (&rtilj);
    B.emx (&rtilj);

    i=min(&dtil, &rtilj);

    fpOut=fopen ("base.out", "at");
    if (!fpOut) {
        perror ("base.out");
        exit (0);
    }
    fprintf (fpOut, "j=%d\n", j);
    fprintf (fpOut, "i=%d\n", i);
    fclose (fpOut);

    if (i==0) {
        fpOut=fopen ("base.out", "at");
        if (!fpOut) {
            perror ("base.out");
            exit (0);
        }
        fprintf (fpOut, "ppl ilimitado\n");
        fputs ("b=[", fpOut);
        b.txtwr (fpOut);
    }

```

```

        fputs ("];", fpOut);
        dtil.fprint (fpOut, "xb");
        fclose (fpOut);
        break;
    }
    ibe=b(i).be;
    ibi=b(i).bi;
    ii=b(i).i;
    vcb=cb(i);

    swap (&B, ibe, ibi, b.blockpos(i), &R, jbe, jbi, r.blockpos (j),
          &jd, &jci, &jce, &id, &ici, &ice);
    indice=b.updatel (jbe, jbi, ji, i);
    cb.update (vcr, indice, i);
    indice=r.updaten (ibe, ibi, ii, j);
    cr.update (vcb, indice, j);
    n2q=U.upo (ibe, ibi, &id, &ici, &ice);
    U.upi (jbe, jbi, &jd, &jci, &jce);
}
A.destroy();
B.destroy();
R.destroy();
U.destroy();
b.destroy();
r.destroy();
c.destroy();
d.destroy();
cb.destroy();
cr.destroy();
z.destroy();
jd.destroy();
jci.destroy();
jce.destroy();
id.destroy();
ici.destroy();
ice.destroy();
rtilj.destroy();
dtil.destroy();

_check(ipassos);
printf ("irinv=%d\n", irinv);
}
#endif

#ifdef _TESTE_SIMPLES_
void main (void)
{
    FILE *fpIn, *fpOut;
    nrbafe A, B, U;
    vetor c, cb;
    ivetor b, r;
    int i;
    char name[80];
    vetor v;
    fpIn=fopen ("auxm\\simplex.bin", "rb");
    if (!fpIn) {
        perror ("auxm\\simplex.bin");
        exit (0);
    }
    A.cellrd (fpIn, name);
    c.cellrd (fpIn, name);
    fclose (fpIn);

    fpIn=fopen ("auxm\\b.txt", "rt");
    if (!fpIn) {
        perror ("auxm\\b.txt");
        exit (0);
    }
    b.create (&A);
    b.txtrd (fpIn);
    fclose (fpIn);

    fpIn=fopen ("auxm\\r.txt", "rt");

```

```

if (!fpIn) {
    perror ("auxm\\r.txt");
    exit (0);
}
r.create (&A);
r.txtrd (fpIn);
fclose (fpIn);
monta (&A, &B, &c, &cb, &b);
for (i=1;i<50;i++) {
    U.copy (&B);
    heap_dump(i);
    U.QR();
}
linha (&A, 2, 2, 3, &v);

fpOut=fopen ("auxm\\simplex.out", "wb");
B.cellwr (fpOut, "B");
v.cellwr (fpOut, "v");
c.cellwr (fpOut, "c");
cb.cellwr (fpOut, "cb");
fclose (fpOut);
}
#endif

```