UNIVERSIDADE ESTADUAL DE CAMPINAS FACULDADE DE ENGENHARIA QUÍMICA ÁREA DE CONCENTRAÇÃO SISTEMAS DE PROCESSOS QUÍMICOS E INFORMÁTICA

MODELOS DE ORDEM REDUZIDA E SIMULAÇÃO EM COLUNAS DE DESTILAÇÃO

Autor: Alexandre Rodrigues Simões Orientador: Prof. Dr. Mário de Jesus Mendes

Dissertação de Mestrado apresentada à Faculdade de Engenharia Química como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Química.

> Campinas – São Paulo Setembro de 2000

> > UNICAMP SIBLIOTECA CENTRA. SECÃO CIRCULANT

i



UNICANP

	L° C	HAMA	OA :	Constant of		<i></i>	÷ į
1 -	wheelman of		CC		2		1
-	sum alter	151	$\gamma\gamma\gamma$				ĺ †
-	I,	Share and the second	Ex.		-5 × 434% (+ 6-37	10 / / / = 4N	٠.
tion of the second	'OM9	IO BC	14	36	, U	Z	` s
	ROC.	16	2	92	\overline{z}	52	
	С			0	X	}	•••
P	REÇA	• <u>_R</u>	\$ 2	11	0	I	1
0	ATA	01	10	\mathcal{D}	10	A	
N	.* Ci	<i>Ф</i>	******	***********			
	900 A 100 A 100 A 100 A 100 A	NAMES OF TAXABLE	Contraction of the local division of the loc		PRESS ANA HOLE	-41-4 .	

CM-00153675-1

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Si51m	 Simões, Alexandre Rodrigues Modelos de ordem reduzida e simulação em colunas de destilação / Alexandre Rodrigues Simões Campinas, SP: [s.n.], 2000.
	Orientador: Mário de Jesus Mendes Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Química.
	1. Polinômios ortogonais. 2. Destilação. 3. Modelos matemáticos. 4. Simulação (Computadores). I. Mendes, Mário de Jesus. II. Universidade Estadual de Campinas. Faculdade de Engenharia Química. III. Título.

ii

Dissertação de Mestrado defendida por Alexandre Rodrigues Simões e aprovada em 19 de setembro de 2000 pela banca examinadora constituída pelos doutores:

Prof. Dr. Mário de Jesus Mendes Orientador

1 Der

Prof. Dr. Huiz Gimeno Latre

Prof. Dr. Roger Josef Zemp

UNICAMP SIBLIOTECA CENTRA...

iii

Esta versão corresponde a redação final da Dissertação de Mestrado em Engenharia Química defendida pelo Eng^o. Alexandre Rodrigues Simões e aprovada pela Comissão Julgadora em 19 de setembro de 2000.

Prof. Dr. Mário de Jesus Mendes Orientador

SECÃO CIRCULANT

Aos meus pais, Roberto e Lourdes, ao meu irmão Roberto, à Josiane e à Sebastiana, por acreditar e me incentivar.

AGRADECIMENTOS

Ao professor Doutor Mário de Jesus Mendes pela orientação, paciência e atenção dedicada.

Ao professor Doutor Roger Josef Zemp pela gentileza de ceder seu pacote de propriedades termodinâmicas.

À CAPES, pelo apoio financeiro.

À Rosa, da secretaria da FEQ.

A todos os professores e funcionários do DESQ e da FEQ.

A todos os colegas e amigos que me ajudaram nesse período, e em especial a Dorival, Marta, Alexandre, Luciana, Rodrigo, Marcone, Arlan e Frede.

Sumário	
---------	--

Nomenclatura	xiii
Resumo	xvii
1. Introdução	1
1.1 Motivação	2
1.2 Divisão do Trabalho	3
2. Revisão Bibliográfica	4
2.1 Objetivos Gerais do Trabalho	8
3. Equações às Diferenças e Colocação Ortogonal Discreta	9
3.1 Introdução	10
3.2 Expansão em Séries de Polinômios Ortogonais Discretos	11
3.3 Aproximação Polinomial: Colocação Ortogonal Discreta	13
3.3.1 O Polinômio Interpolador de Lagrange	17
3.2 Colocação Ortogonal Discreta e Diferenças Finitas	18
3.2.1 Definição das Diferenças Finitas	18
3.2.2 Algumas Propriedades de $\Delta e \nabla$	20
3.2.3 Aproximação das Diferenças de uma Função utilizando	
o Polinômio Interpolador de Lagrange	21
3.2.3.1 Cálculo das Diferenças "Forward"	21
3.2.3.2 Cálculo das Diferenças "Backward"	23
4. Modelos de Ordem Reduzida e Simulação em Colunas de Destilação	26
4.1 Introdução	27
4.2 Desenvolvimento do Modelo Completo	27
4.3 Desenvolvimento do Modelo de Ordem Reduzida utilizando	
Colocação Ortogonal Mista	31
4.4 Desenvolvimento do Modelo de Ordem Reduzida utilizando uma	
modificação da Colocação Ortogonal Mista	40
4.5 Aplicação dos Modelos de Ordem Reduzida à Simulação	
de Colunas de Destilação	46
4.5.1 Simulação de uma coluna depropenizadora em regime	
estacionário	46
5. Conclusões e Sugestões	63
5.1 Conclusões	64

	xii
5.2 Sugestões	65
Referências Bibliográficas	71
Anexo I - Polinômios Ortogonais Discretos e Cálculo de Diferenças Finitas	77
Anexo II – Programa Principal, Subrotina MA28 e dependências	103

Nomenclatura

Simbologias

unidades	
----------	--

b	Vazão molar do componente no produto de fundo	(mol/h)
В	Vazão total do produto de fundo	(mol/h)
d	Vazão molar do componente no destilado	(mol/h)
D	Vazão total do destilado	(mol/h)
f	Vazão molar do componente na alimentação	(mol/h)
F	Vazão total na alimentação	(mol/h)
h	Entalpia parcial molar do componente	(cal/h)
$h_n^{(\alpha,\beta)}(x,N)$	Polinômio de Hahn de grau n	
Н	Entalpia total	(cal/h)
K	Constante de equilíbrio	adim
1	Vazão molar do componente da fase líquida	mol/h
$l_k(s)$	Polinômio interpolador de Lagrange	
L	Vazão total de líquido	mol/h
Μ	Holdup de líquido no prato	adim
n	Número de pontos internos de colocação	
npcse	Número de pontos de colocação na secção de esgotamento	
npcsr	Número de pontos de colocação na secção de retificação	
N	Número total de pratos na coluna	

- i Componente
- j Índice do prato no modelo completo ou índice do ponto de colocação no modelo reduzido
- k Índice do ponto de colocação
- r Refervedor

Índice superior

- E Secção de esgotamento
- L Fase líquida
- R Secção de retificação
- V Fase vapor

RESUMO

Uma das maiores dificuldades encontradas no estudo dos processos de separação por estágios é a grande dimensão dos modelos do processo. Assim, a simulação desses processos envolve um elevado número de equações algébrico-diferenciais não-lineares ocasionando um elevado esforço computacional. Vários pesquisadores têm proposto estratégias que reduzam a dimensão dos modelos, estas técnicas recebem o nome de métodos de redução de ordem e os modelos por ela obtidos são denominados modelos de ordem reduzida.

Este trabalho tem como objetivos o desenvolvimento de modelos de ordem reduzida para sistemas de processos de separação por estágios usando a metodologia da colocação ortogonal discreta, a seleção da melhor estratégia de colocação, assim como a seleção do algoritmo mais adequado para a solução das equações do modelo reduzido visando selecionar aquela combinação de estrutura de modelo e de algoritmo que permita obter uma simulação adequada e custo computacional mínimo.

A aplicação do método da colocação ortogonal é estendida à solução de equações às diferenças, a partir do desenvolvimento de uma metodologia que unifica o uso do método da colocação ortogonal tanto para a solução de equações diferenciais quanto para a de equações às diferenças.

Os resultados obtidos nas simulações realizadas demonstram a viabilidade da aplicação da colocação ortogonal discreta na simulação de colunas de destilação em estado estacionário. O esquema de colocação ortogonal mista levou a uma melhor aproximação em relação à colocação ortogonal mista modificada, além de uma redução considerável no tempo computacional em comparação com o modelo completo.

ABSTRACT

One of the largest difficulties found in the analysis of staged separation processes is the great dimension of their models. Thus, the simulation of these processes involves a high number of algebraic-differential equations, causing a high computational effort. Several researchers have been proposing order reduction strategies, that is reduce the dimension of the models.

This work has as objectives the development of reduced order models for systems of staged separation processes through collocation strategies, the selection of the best collocation strategy, as well as the selection of the algorithm more adapted for the solution of the equations of the model seeking that combination of model structure and of algorithm that allows to obtain an appropriate simulation and minimum computational cost.

The application of the method of orthogonal collocation is extended to the solution of differences equations, starting from the development of a methodology that unifies the use of orthogonal collocation for the solution of differential equations as well as for the one of differences equations.

The results obtained in the simulations demonstrate the viability of the application of discrete orthogonal collocation in the simulation of distillation columns in stationary state. The discrete mixed orthogonal collocation led to a better approach than the modified mixed orthogonal collocation. Besides, a considerable reduction in the computational time in comparison to the complete model was obtained.

CAPÍTULO 1

INTRODUÇÃO

1. INTRODUÇÃO

1.1 Motivação

A destilação é um dos processos de separação por estágios mais estudados, visto que grande parte da indústria química utiliza esse processo. Uma das maiores dificuldades com modelos matemáticos de sistemas de separação por estágios é a grande dimensionalidade do modelo do processo, o que, entra outras coisas, aumenta as dificuldades computacionais para obter as condições ótima de operação da coluna. Assim existe a necessidade de desenvolver modelos de ordem reduzida mais eficientes.

O uso de modelos de ordem reduzida é fundamental em problemas de controle e otimização. A otimização de processos em tempo real, cujo objetivo é a determinação das condições ótimas de operação de uma planta ou unidade de processo em tempo real, requer o uso de modelos estáticos que representem o comportamento do processo com suficiente exatidão. Os modelos mais comumente usados na simulação de colunas de destilação são os modelos ditos completos de estágios (ideais ou com uma dada eficiência). No entanto, estes apresentam uma grande dimensão, o que torna o esforço computacional para a otimização em estado estacionário bastante elevado, principalmente quando a coluna possui muitos pratos ou mais de uma unidade está envolvida.

No controle preditivo com modelo, o controlador utiliza o conhecimento da dinâmica do processo (um modelo explícito) para encontrar a trajetória da variável controlada sobre um horizonte de controle que otimiza a função objetivo num horizonte de predição. O algoritmo de controle usa a otimização e a solução do sistema de equações algébrico-diferenciais do modelo dinâmico para determinar a trajetória ótima da variável controlada a cada instante de amostragem. Os modelos prato-a-prato de colunas de destilação geram um problema de programação não-linear contendo um grande número de variáveis e restrições cuja solução requer um grande esforço computacional para cada instante de amostragem. Isto pode criar um grande atraso entre a amostragem e a implementação das ações de controle e, em casos extremos, o tempo de cálculo pode ser maior que o de amostragem.

O controle ótimo no tempo é empregado quando deseja-se conduzir o sistema de um estado inicial para algum estado final, no menor tempo possível, por exemplo. Isto envolve

Capítulo 1 - Introdução

um problema de otimização cujas restrições incorporam o modelo dinâmico do processo (um sistema de equações algébrico-diferenciais). No caso particular de colunas de destilação, dada a elevada dimensão dos modelos dinâmicos completos, um grande esforço computacional é requerido para a obtenção da solução ótima.

1.2 Divisão do Trabalho

Apresentada a motivação deste trabalho, parte-se para a divisão do mesmo.

Inicialmente no capítulo 2 será feito uma revisão da literatura sobre modelos de ordem reduzida de colunas de destilação por estágios, bem como uma análise crítica sobre os modelos de ordem reduzida existentes e serão apresentados os objetivos gerais do presente trabalho.

No capítulo 3 será abordada a aplicação dos polinômios ortogonais de uma variável discreta ao cálculo das diferenças finitas, visando a solução de equações às diferenças.

Em seguida, no capítulo 4 será descrita a metodologia de redução de ordem aplicada a colunas de destilação. Para testar a eficiência dos métodos serão simulados alguns casos exemplo e os resultados comparados com os obtidos com o modelo completo.

Finalmente no capítulo 5 serão apresentadas as conclusões e sugestões para trabalhos futuros.

CAPÍTULO 2

REVISÃO BIBLIOGRÁFICA

2. Revisão Bibliográfica

Umas das maiores dificuldades com modelos matemáticos de sistemas de separação por estágios é a grande dimensionalidade do modelo do processo. Um modelo de estado estacionário prato-a-prato típico apresenta N(2NC+1) equações algébricas nãolineares , sendo NC o número de componentes e N o número de estágios da coluna, as quais, devido ainda a eventuais problemas de rigidez, podem apresentar grandes dificuldades na sua solução. Para tentar contornar estes problemas vários pesquisadores têm proposto estratégias que reduzam o número de equações sem perda de precisão em relação o modelo original. Estas técnicas recebem o nome de métodos de redução de ordem e os modelos por elas obtidos são denominados modelos de ordem reduzida.

Uma primeira categoria de modelos reduzidos, como os de Wahl e Harriot (1970) e Georgakis e Stoever (1982), utiliza modelos linearizados. Porém estes são de aplicação bastante restrita, pela própria natureza da linearização. Outra categoria é formada pelos chamados modelos compartimentais de España e Landau (1978) e Benallou *et al.* (1986), que baseiam-se no conceito de prato sensitivo. A grande desvantagem deste método, segundo Seferlis e Hrymak (1994a), é que o prato sensitivo não é definido explicitamente e sua localização tem um papel importante no tempo requerido para a simulação.

O uso da colocação ortogonal para solução de modelos de equações diferenciais é uma metodologia bem desenvolvida (Villadsen e Michelsen, 1978a). Recentemente vários artigos têm utilizado a colocação ortogonal para solução de modelos de equações às diferenças, típico de processos por estágios. Osborne (1971) transformou o modelo de equações às diferenças, para processos por estágios, em equações diferenciais parciais por aproximação das diferenças por diferenciais.

Wong e Luus (1980) desenvolveram um método de redução de ordem baseado na idéia que os perfis de composição possam ser representados como variáveis contínuas ao longo da coluna. Neste trabalho, à semelhança de Osborne (1971), os autores propuseram que as equações às diferenças do modelo original fossem transformadas em equações diferenciais parciais e estas resolvidas pelo método da colocação ortogonal, com os pontos de colocação localizados nas raízes do polinômio de Jacobi. Foi observada uma redução considerável no número de equações, no entanto o balanço de massa no estado estacionário não era preservado. Esta metodologia é razoável para colunas de recheio, porém demonstra-

Capítulo 2 - Revisão Bibliográfica

se inadequada para colunas de pratos, pois não preserva a natureza discreta das equações às diferenças do modelo original.

Cho e Joseph (1983) usaram duas abordagens para construir aproximações de ordem reduzida dos modelos prato-a-prato do processo de separação. Na primeira abordagem, similarmente à Osborne, a coluna de separação por estágio é aproximada como um sistema distribuído, com os perfis de composição e vazão tratados como variáveis contínuas ao longo do comprimento da coluna. As equações diferenciais parciais obtidas são então resolvidas pelo método da colocação ortogonal usando polinômios de Jacobi. Na segunda abordagem, a redução do modelo foi obtida pela transição direta para representação polinomial dos perfis de composição e vazão na coluna, utilizando interpolação de Lagrange e polinômios de Jacobi. Para uma simples coluna de absorção de gás, sem pratos de alimentação e sem pratos com retiradas laterais, verificou-se que o procedimento direto preserva a exatidão, enquanto que a abordagem por equações diferenciais parciais tende a dar erros altos.

Stewart *et al.* (1985) desenvolveram um método rápido para simulação de processos de separação multiestágio baseado também na transição direta para representação polinomial dos perfis de composição e entalpias na coluna. Entretanto, os autores basearam os polinômios da interpolação de Lagrange nas raízes dos polinômios de Hahn, uma família de polinômios ortogonais discretos.

Pinto e Biscaia (1988) discutiram os méritos de diferentes estratégias de redução de ordem para modelos de processo de separação por estágios. Os autores procuraram selecionar as melhores famílias de polinômios ortogonais discretos a serem usados na construção de modelos de ordem reduzida para sistema de separação por estágios. Para isso, eles escolheram um método dito de "colocação ortogonal por seções sem extrapolação", escolhendo em cada seção, o "prato extremo" (condensador ou refervedor) como ponto de colocação. Deste modo foram obtidos famílias de polinômios que não são nem do tipo Jacobi nem do tipo Hahn. Contudo, o método proposto tem o inconveniente de não considerar as descontinuidades que ocorrem justamente no condensador e no refervedor.

Um dos problemas das estratégias propostas por Stewart *et al.* (1985) e Pinto e Biscaia (1988) é que elas não levam em conta adequadamente os pratos de alimentação e os pratos com retiradas laterais. Uma outra crítica que poderia ser levantada diz respeito à complexidade da implementação do método da transição direta para representação polinomial dos perfis na coluna.

Seferlis e Hrymak (1994a,b) propuseram uma formulação de redução de ordem sendo a coluna dividida em dois elementos, cada um deles definido como um grupo de pratos situados entre dois pratos das correntes que entram ou saem da coluna. Uma coluna com um simples prato de alimentação e sem retiradas laterais, por exemplo, é separada em duas regiões. Aqueles pratos onde ocorrem descontinuidades do perfil na coluna são tratados como estágios discretos, enquanto que para as regiões os autores usaram a transição direta para representação polinomial dos perfis na coluna, com pontos de colocação internos como sendo as raízes dos polinômios de Hahn.

A colocação ortogonal discreta é um método eficiente para a redução da ordem do modelo de processos de separação por estágios (Serfelis e Hrymak, 1994). Contudo, o emprego da colocação ortogonal discreta para redução da ordem do modelo de processos de separação por estágios pode ser talvez melhor descrito pela opinião de Taylor e Lucia (1995) que " não existe muita publicação que utiliza esses métodos". Além das críticas formuladas acima, a situação é talvez devido ao fato que a metodologia da colocação ortogonal discreta é relativamente desconhecida pelos engenheiros, ao mesmo tempo, quando tenta-se usar polinômios ortogonais discretos ou aplicar colocação ortogonal discreta , o engenheiro é confrontado com o problema que a teoria dos polinômios ortogonais discretos está dispersa através de diferentes livros e artigos e que mesmo a notação difere dependendo do autor. Além disso, algumas questões mais práticas tem que ser respondidas, como a escolha da família do polinômios discretos ou o cálculo das raízes do polinômio ortogonal discreto.

2.1 Objetivos Gerais do Trabalho

O objetivo principal deste trabalho é o desenvolvimento de uma sistemática de construção de modelos de ordem reduzida para sistemas de separação por estágios,ou, mais especificamente, para colunas de destilação. Assim, serão analisados problemas em relação aos seguintes tópicos:

• Seleção da estratégia de colocação, a da colocação ortogonal mista e colocação ortogonal mista modificada.

 Seleção do algoritmo mais adequado para a solução das equações do modelo visando selecionar aquela combinação de estrutura do modelo e de algoritmo que permita obter uma simulação com precisão adequada e custo computacional mínimo.

• Avaliação do efeito de estratégia de redução e de simulação sobre o tempo computacional.

Dadas as características anteriormente mencionadas referentes à Teoria dos Polinômios Ortogonais Discretos, são apresentados no Anexo I os pontos básicos desta Teoria, seguida basicamente a notação adotada por Nikiforov *et al.* (1991).

CAPÍTULO 3

EQUAÇÕES ÀS DIFERENÇAS E COLOCAÇÃO ORTOGONAL DISCRETA

3. EQUAÇÕES ÀS DIFERENÇAS E COLOCAÇÃO ORTOGONAL DISCRETA

3.1. Introdução

Seja dada uma família de polinômios { $p_0(x), p_1(x), \dots, p_n(x), \dots$ }, onde $p_n(x)$ é um polinômio exatamente de grau n.

Sob certas condições, a função f(x) pode ser expandida numa série dos polinômios $p_n(x)$

$$f(x) = \sum_{j=0}^{\infty} c_j p_j(x) .$$
 (3.1)

Contudo, se a série (3.1) for interrompida no termo em $p_n(x)$ ela degenera numa aproximação de f(x) por um polinômio de grau n

$$f(x) \approx \sum_{j=0}^{n} c_j p_j(x) .$$
(3.2)

Os valores dos coeficientes c_j (j=0,1,...,n) na relação (3.2) podem ser calculados de tal maneira que esta relação seja exata para n+1 valores de x, $\{x = x_0, x_1, ..., x_n\}$, através do sistema de equações

$$f(x_i) = \sum_{j=0}^{n} c_j p_j(x_i) \quad , \quad (i = 0, 1, ..., n) \quad . \tag{3.3}$$

Para qualquer outro valor de $x \neq x_i$ (i=0,1,...,n) o resíduo da aproximação (3.2), que representaremos por $R_{n+1}(x)$, será então dado por (Jordan, 1965):

$$R_{n+1}(x) = (x - x_0)(x - x_1)...(x - x_n)\frac{f^{(n+1)}(\xi)}{(n+1)!}$$
(3.4)

onde ξ é um valor situado no intervalo que contém os pontos ($x, x_0, x_1, ..., x_n$), isto é, ξ será uma função de x (mas uma vez escolhido o valor de x em (3.4), $\frac{f^{(n+1)}(\xi)}{(n+1)!}$ será uma constante)!

Deste modo, a expansão:

$$f(x) = c_0 p_0(x) + c_1 p_1(x) + \dots + c_n p_n(x) + \frac{(x - x_0)(x - x_1)\dots(x - x_n)}{(n+1)!} f^{(n+1)}(\xi), \quad (3.5)$$

onde ξ uma função de x, é exata para qualquer valor de x!

A relação (3.4) é interessante, pois mostra que o resíduo $R_{n+1}(x)$ depende apenas dos valores ($x, x_0, x_1, ..., x_n$) e da derivada $f^{(n+1)}(x)$ de f(x), e não depende dos polinômios $p_n(x)$ usados na expansão.

O desenvolvimento feito mostra ainda que, se f(x) for uma função polinomial de grau exatamente *n* a expansão (3.2) será exata, ou seja, $R_{n+1}(x)$ será nulo para qualquer *x*.

3.2. Expansão em Séries de Polinômios Ortogonais Discretos

Consideremos novamente uma função f(x), e seja { $p_n(x)$ } uma família de polinômios ortogonais discretos definida pela condição de ortogonalidade

$$\sum_{i=0}^{N-1} \rho(x_i) p_n(x_i) p_m(x_i) = \delta_{mn} d_n^2 \qquad (3.6)$$

Suponhamos, inicialmente, que f(x) é um polinômio exatamente de grau n < N. Então, como vimos acima, a expansão polinomial

$$f(x) = \sum_{j=0}^{n} c_j \rho_j(x) , \qquad (3.7)$$

será exata para todos os valores de x. Neste caso, os valores dos coeficientes c_i podem ser obtidos multiplicando ambos os membros de (3.7) por $\rho(x)\rho_m(x)$, e somando para todos os x_i (i = 0, 1, ..., N-1)

$$\sum_{i=0}^{N-1} \rho(x_i) f(x_i) p_m(x_i) = \sum_{i=0}^{N-1} \left(\sum_{j=0}^n c_j p_j(x_i) \rho(x_i) p_m(x_i) \right),$$

ou

$$\sum_{i=0}^{N-1} \rho(x_i) f(x_i) p_m(x_i) = \sum_{j=0}^n c_j \sum_{i=0}^{N-1} \rho(x_i) p_j(x_i) p_m(x_i) \quad . \quad (3.8)$$

Devido, contudo, à condição de ortogonalidade (3.6), obtém-se imediatamente de (3.8):

$$c_{j} = \frac{\sum_{i=0}^{N-1} \rho(x_{i}) f(x_{i}) p_{j}(x_{i})}{d_{j}^{2}} \qquad (3.9)$$

No caso mais geral em que f(x) não é um polinômio de grau $\leq n$ a expansão (3.7) constitui apenas uma aproximação de f(x). Neste caso pode-se mostrar que os valores dos coeficientes c_i dados por (3.9) são ótimos pelo critério dos mínimos quadrados

$$S_n = \sum_{i=0}^{N-1} \rho(x_i) \left\{ f(x_i) - \sum_{j=0}^n c_j p_j(x_j) \right\}^2 \equiv Min \qquad (3.10)$$

Com efeito, as condições de mínimo

$$\frac{\partial S_n}{\partial c_k} = 2 \sum_{i=0}^{N-1} \rho(x_i) \left\{ f(x_i) - \sum_{j=0}^n c_j p_j(x_i) \right\} p_k(x_i) = 0 \quad ,$$

devido à condição de ortogonalidade (3.6), reduzem-se a

$$\sum_{i=0}^{N-1} \rho(x_i) f(x_i) p_j(x_i) = c_j \sum_{i=0}^{N-1} \rho(x_i) p_j^2(x_i)$$

ou

$$c_{j} = \frac{\sum_{i=0}^{N-1} \rho(x_{i}) f(x_{i}) p_{j}(x_{i})}{d_{j}^{2}} , \qquad (3.11)$$

expressão obviamente idêntica a (3.9).

Estas considerações permitem chegar a uma conclusão interessante (Jordan, 1965):

Para obter a melhor aproximação possível de uma função f(x), definida para os N valores de x {x=0,1,2,...,N-1}, por um polinômio de grau n de acordo com o princípio dos mínimos quadrados

$$S_n = \sum_{i=0}^{N-1} \left[f(x_i) - \sum_{j=0}^n c_j p_j(x_i) \right]^2 \equiv Min$$

basta expandir essa função numa série dos polinômios de Chebyshev discretos $\{p_j(x) = t_j(x, N)\}$ e terminar a série no termo em $t_n(x)$, sendo os c_j dados pela relação (3.76) correspondente.

3.3. Aproximação Polinomial : Colocação Ortogonal Discreta

Consideremos o seguinte problema:

Dada uma função f(x), contínua no intervalo [0, N-1], deseja-se aproximar essa função por um polinômio exatamente de grau n, $P_n(x)$

$$f(x) \approx P_n(x) \quad . \tag{3.12}$$

,

A qualidade da aproximação (3.12) é definida pela soma dos quadrados dos desvios S_N

$$S_N = \sum_{i=0}^{N-1} \left\{ f(x_i) - P_n(x_i) \right\}^2 \quad ., \tag{3.13}$$

onde os pontos x_i (i = 0, 1, ..., N - 1) são os <u>N pontos equidistantes</u>

$$x_{i+1} - x_i = 1$$

 $x_0 = 0; x_{N-1} = N - 1$

Como anteriormente, vamos definir o polinômio $P_n(x)$ por uma expansão em série na família de polinômios $\{p_n(x)\}$, truncada no termo em $p_n(x)$

$$P_n(x) = \sum_{j=0}^n c_j \, p_j(x) \, . \tag{3.14}$$

Como vimos, se os polinômios $p_j(x)$ forem os polinômios de Chebyshev discretos, $p_j(x) = t_j(x, N)$, e se os (n+1) coeficientes c_j em (3.14) forem calculados pelas relações

$$c_{j} = \frac{\left(f, t_{j}\right)}{h_{j}^{2}} = \frac{\sum_{i=0}^{N-1} f(x_{i}) t_{j}(x_{i})}{h_{j}^{2}} ,$$

a aproximação (3.12) será ótima, no sentido de que a soma dos quadrados dos desvios S_N será mínima.

Nas aplicações é contudo frequente escolher (n+1) pontos $u_j \in [0, N-1], (j = 0, 1, ..., n)$ e calcular os (n+1) coeficientes c_j de (3.14) a partir das (n+1) equações obtidas impondo a condição de que para $x = u_j$, (j = 0, 1, ..., n), a aproximação (3.12) seja exata, isto é, que se tenha

$$f(u_j) = P_n(u_j) = \sum_{k=0}^n c_k p_k(u_j) \quad (j = 0, 1, ..., n).$$
(3.15)

Põe-se então o problema de saber se existe um conjunto "ótimo" de pontos u_j (j = 0,1,...,n) para o qual a aproximação (3.12) pode ser considerada como "boa" no sentido da minimização da soma dos quadrados dos resíduos (3.13).

Para resolver este problema vamos considerar três possíveis situações.

1^a Situação: Colocação de Gauss. Os (n+1) pontos de colocação u_j (j = 0,1,...,n) estão todos situados no intervalo (0, N-1), isto é tem-se $u_j \in (0, N-1)$ (j = 0,1,...,n).

Neste caso, a soma dos quadrados dos resíduos é dada por

$$S_{N} = \sum_{i=0}^{N-1} \left\{ f(x_{i}) - P_{n}(x_{i}) \right\}^{2} = \sum_{i=0}^{N-1} \left\{ R_{n+1}(x_{i}) \right\}^{2} , \qquad (3.16)$$

onde, de acordo com (3.4) se tem

$$R_{n+1}(x) = \beta(x)(x - u_0)(x - u_1)...(x - u_n).$$
(3.17)

Mas, a soma (3.16) pode ser aproximada por uma fórmula de quadratura de Gauss,

$$\sum_{i=0}^{N-1} \{R_{n+1}(x_i)\}^2 \approx \sum_{j=0}^n \omega_j \{R_{n+1}(y_j)\}^2 , \qquad (3.18)$$

onde os y_j (j = 0,1,...,n) são os zeros do polinômio de grau (n+1) pertencente à família de polinômios ortogonais discretos definida pela condição de ortogonalidade

$$(p_n, p_m) = \sum_{i=0}^{N-1} p_n(x_i) p_m(x_i) = h_n^2 \delta_{mn},$$

ou seja, a família dos polinômios de Chebyshev discretos.

De (3.16-18) conclui-se imediatamente que, se <u>os (n+1) pontos de colocação</u> $u_j (j = 0,1,..,n)$ forem escolhidos como sendo os zeros do polinômio de Chebyshev $t_{n+1}(x, N)$, se terá

$$S_N = \sum_{i=0}^{N-1} \left\{ R_{n+1}(x_i) \right\}^2 \approx 0 \quad . \tag{3.19}$$

Este resultado mostra que nestas condições a aproximação (3.12) pode ,pelo menos ser considerada como "sub-ótima", de acordo com o princípio dos mínimos quadrados.

2^a Situação :Colocação de Radau. Nesta situação os (n+1) pontos de colocação u_j são escolhidos segundo os seguintes critérios:

-<u>n</u> desses pontos, digamos $u_1, u_2, ..., u_n$, são escolhidos no interior do intervalo (0, N-1);

-o ponto restante, u_0 , <u>é escolhido como sendo um dos extremos do intervalo</u>, isto é $u_0 = 0$ ou

 $u_0 = N - 1 .$

Supondo, por exemplo, que escolhemos $u_0 = N - 1$, então o resíduo da aproximação (3.12) será

$$R_{n+1}(x) = \beta(x)(x - u_1)(x - u_2)...(x - u_n)(x - x_{N-1}), \text{ onde } x_{N-1} = N - 1$$

e $u_j \in (0, N-1) (j = 1, 2, ..., n).$ (3.20)

Neste caso a soma dos quadrados dos resíduos (3.16) pode ser aproximada por uma fórmula de quadratura de Radau,

$$S_{N} = \sum_{i=0}^{N-1} \{R_{n+1}(x_{i})\} \approx \sum_{j=1}^{n} \omega_{j} \{R_{n+1}(y_{j})\}^{2} + \omega_{N-1} \{R_{n+1}(x=N-1)\}^{2}, \quad (3.21)$$

onde os y_j (j = 1,...,n) são os zeros do polinômio de grau <u>n</u>, $p_n(x)$, pertencente à família de polinômios ortogonais discretos definida pela condição de ortogonalidade

$$(p_n, p_m) = \sum_{i=0}^{N-1} (x_{N-1} - x_i) p_n(x_i) p_m(x_i) = h_n^2 \delta_{mn} . \qquad (3.22)$$

De (3.20-22) concluímos imediatamente que <u>se os n pontos de colocação</u> u_j (j = 1, ..., n)<u>internos ao intervalo (0, N-1) forem os zeros do polinômio ortogonal discreto de grau n</u> <u> $_{2}p_n(x)$, definido por (3.22)</u> se terá

$$S_N = \sum_{i=0}^{N-1} \{R_{n+1}(x_i)\}^2 \approx 0,$$

ou seja, a aproximação (3.12) é, nestas condições, pelo menos "sub-ótima" de acordo com o princípio dos mínimos quadrados.

3ª Situação: Colocação de Lobatto. Nesta última situação os (n+1) pontos de colocação u_j são escolhidos segundo os seguintes critérios:

-(n-1) destes pontos, digamos $u_1, u_2, ..., u_{n-1}$, são escolhidos no interior do intervalo (0, N - 1);

-os dois pontos restantes, $u_0 \in u_n$, são escolhidos como sendo iguais aos extremos do intervalo,

$$\operatorname{com} u_0 = 0 \ \mathrm{e} \ u_n = N - 1$$
, por exemplo.

Neste caso o resíduo da aproximação (3.12) será dado por

$$R_{n+1}(x) = \beta(x)(x-u_1)(x-u_2)\dots(x-u_{n-1})(x-x_0)(x-x_{N-1}), \quad [x_0 = 0, x_{N-1} = N-1] \quad (3.23)$$

e a soma dos quadrados dos resíduos pode ser aproximada por uma fórmula de quadratura de Lobatto

$$S_{N} = \sum_{i=0}^{N-1} \left\{ R_{n+1}(x_{i}) \right\}^{2} \approx \sum_{j=1}^{n-1} \omega_{j} \left\{ R_{n+1}(y_{j}) \right\}^{2} + \omega_{0} \left\{ R_{n+1}(x_{0}) \right\}^{2} + \omega_{N-1} \left\{ R_{n+1}(x_{N-1}) \right\}^{2}, (3.24)$$

onde os y_j (j = 1,...,n-1) são os zeros do polinômio de grau (n-1), $p_{n-1}(x)$, pertencente à família de polinômios ortogonais discretos definida pela condição de ortogonalidade

$$(p_n, p_m) = \sum_{i=0}^{N-1} (x_i - x_0) (x_{N-1} - x_i) p_n(x_i) p_m(x_i) = h_n^2 \delta_{mn} \quad . \tag{3.25}$$

De (3.23-25) concluímos imediatamente que, se os (n-1) pontos de colocação u_j (j = 1, 2, ..., n - 1) inter-nos ao intervalo (0, N - 1) forem os zeros do polinômio ortogonal discreto de grau (n-1), $p_{n-1}(x)$, definido por (3.25) se terá

$$S_N = \sum_{i=0}^{N-1} \{R_{n+1}(x_i)\}^2 \approx 0$$
,

ou seja, a aproximação (3.12) é, nestas condições, pelo menos "sub-ótima" de acordo com o princípio dos mínimos quadrados.

3.3.1. O Polinômio Interpolador de Lagrange

Uma vez definidos os <u>pontos de colocação</u> u_j (j = 0,1,...,n), o polinômio aproximador de grau n, $P_n(x)$, pode ser construído usando (3.14), com os coeficientes c_j calculados a partir de (3.15). Normalmente ,contudo, o polinômio aproximador $P_n(x)$ é construido usando o chamado <u>Polinômio Interpolador de Lagrange</u>:

-Uma vez definidos os pontos de colocação u_j (j = 0,1,...,n) o polinômio de grau n que *interpola* os (n+1) pontos $[u_j, f(u_j)]$ é dado por:

$$P_n(x) = \sum_{j=0}^n \ell_j(x) f(u_j) , \qquad (3.26)$$

onde os $\ell_j(x)$ são, por sua vez, polinômios de grau *n* definidos por:

$$\ell_j(x) = \frac{q_{n+1}(x)}{(x - u_j)q'_{n+1}(u_j)} , \qquad (3.27.a)$$

com:

$$q_{n+1}(x) = (x - u_0)(x - u_1)...(x - u_n)$$
 (3.27.b)

3.2. COLOCAÇÃO ORTOGONAL DISCRETA E DIFERENÇAS FINITAS

Uma das áreas de aplicação da Colocação Ortogonal Discreta é a do <u>Cálculo das Diferenças</u> <u>Finitas</u>, que trata especialmente de funções para as quais a variável independente x apenas toma determinados valores $x_0, x_1, ..., x_n$, isto é, a variável é descontínua.

3.2.1. Definição das Diferenças Finitas

A teoria do Cálculo das Diferenças Finitas foi desenvolvida para o caso de funções $\{f(x)\}$ definidas para valores discretos de x, $\{x_0, x_1, x_2, ..., x_n, ...\}$, quando estes valores são equidistantes, isto é, quando se tem

$$x_{i+l} - x_i = h , (3.28)$$

onde h é independente de i.

A teoria pode contudo, também ser aplicada a funções de variável contínua, através do conceito de diferenças finitas. Assim, dada a função contínua f(x), define-se a <u>1^a diferença</u> para a frente de f(x), representada por $\Delta f(x)$ por:

$$\Delta f(x) = f(x+h) - f(x)$$
 . (3.29)

onde h é constante independente do valor de x (Kreysig, 1993).

As fórmulas do Cálculo das Diferenças Finitas são extremamente simplificadas quando o incremento $h \, \acute{e}$ igual a 1 (h=1). É sempre possível reduzir a situação (3.29) ao caso h=1 pela mudança de variável

$$x = a + h\xi$$
,

de tal modo que, quando $\Delta x = h \in \Delta \xi = 1$. Então

$$f(x) = F(\xi)$$

e

$$\Delta F(\xi) = F(\xi+1) - F(\xi) \qquad (3.30)$$

Uma vez definida a diferença de ordem um, e supondo um valor de acréscimo h=1, nós podemos definir <u>diferenças de ordem superior</u> para a função f(x):

-Segunda diferença para a frente de
$$f(x)$$
, $\Delta^2 f(x)$

$$\Delta^2 f(x) = \Delta \left[\Delta f(x) \right] = \Delta f(x+1) - \Delta f(x) \qquad (3.31.a)$$

ou

$$\Delta^2 f(x) = f(x+2) - 2f(x+1) + f(x)$$
 (3.31.b)

- Enésima diferença para a frentede f(x), $\Delta^n f(x)$:

$$\Delta^{n} f(x) = \Delta \left[\Delta^{n-1} f(x) \right] \qquad (3.32)$$

Embora muitos autores não as considerem explicitamente (Jordan, 1965), é também frequente usar as chamadas <u>"backward differences</u>" ou <u>diferenças para trás da função</u> <u>contínua f(x) (Kreysig, 1993):</u>

-Primeira "backward difference" de f(x), $\nabla f(x)$

$$\nabla f(x) = f(x) - f(x-1)$$
 (3.33)

- Enésima "backward difference" de f(x), $\nabla^n f(x)$

$$\nabla^n f(x) = \nabla \left[\nabla^{n-1} f(x) \right]$$
 (3.34)

3.2.2. Algumas Propriedades de Δ e ∇

-Propriedade Comutativa em relação a Constantes

$$\Delta[cf(x)] = c\Delta f(x)$$

$$\nabla[cf(x)] = c\nabla f(x)$$
(3.35)

onde c é uma constante.

-Propriedade Distributiva:

$$\Delta[f_1(x) + f_2(x)] = \Delta f_1(x) + \Delta f_2(x) \qquad (3.36)$$

- Propriedade Comutativa dos Operadores:

$$\Delta \nabla = \nabla \Delta \qquad (3.37)$$

- Operadores Aplicados a Produtos:

$$\Delta[f_1(x)f_2(x)] = \Delta f_1(x).f_2(x) + \Delta f_2(x).f_1(x) + \Delta f_1(x).\Delta f_2(x)$$
(3.38.a)

$$\nabla [f_1(x).f_2(x)] = f_1(x).\nabla f_2(x) + f_2(x).\nabla f_1(x) - \nabla f_1(x)\nabla f_2(x)$$
(3.38.b)

3.2.3. Aproximação das Diferenças de uma Função utilizando o Polinômio Interpolador de Lagrange

Consideremos uma função f(x) e um polinômio interpolador de Lagrange que aproxima a função com base nos n+1 pontos de colocação u_i (j=0,1,...,n):

$$f(x) \cong \sum_{j=0}^{n} \ell_j(x) f(u_j) \qquad , \qquad (3.39)$$

onde, de acordo com (3.27a-b) se tem:

$$\ell_j(x) = \frac{p_{n+1}(x)}{(x - u_j)p'_{n+1}(u_j)} , \qquad (3.40.a)$$

e

$$p_{n+1}(x) = (x - u_0)(x - u_1)...(x - u_n)$$
 (3.40.b)

3.2.3.1. Cálculo das Diferenças "Forward"

De acordo com (3.35) e (3.36) tem-se:

$$\Delta f(x) \cong \sum_{i=0}^{n} \Delta \ell_i(x) f(u_i) \qquad (3.41)$$

Por outro lado, de (3.40.a) obtém-se:

$$(x - u_i) \ell_i(x) = \frac{p_{n+1}(x)}{p'_{n+1}(u_i)} \qquad (3.42)$$

Usando a diferença do produto, eq. (3.38.a), e atendendo a que se tem

$$\Delta(x - x_i) = (x + 1 - x_i) - (x - x_i) = 1,$$

obtém-se de (3.42):

$$\ell_i(x) + (x - u_i + 1) \Delta \ell_i(x) = \frac{\Delta p_{n+1}(x)}{p'_{n+1}(u_i)}$$

ou:

$$\ell_i(x) + (x - u_i + 1) \Delta \ell_i(x) = \frac{p_{n+1}(x + 1) - p_{n+1}(x)}{p'_{n+1}(u_i)}$$

ou

$$\Delta \ell_i(x) = \frac{p_{n+1}(x+1) - p_{n+1}(x)}{(x-u_i+1)p'_{n+1}(u_i)} - \frac{\ell_i(x)}{(x-u_i+1)} \qquad (3.43)$$

Nas aplicações está-se interessado em calcular os valores das diferenças nos pontos de colocação.

<u>1⁰ caso</u>: $\Delta \ell_i(u_i)$

Como se tem $p_{n+1}(u_i) = 0$, e como

$$\frac{p_{n+1}(x+1)}{(x-u_i+1)} = \frac{\prod_{k=0}^n (x+1-u_k)}{(x+1-u_i)} = \prod_{k=0,i}^n (x+1-u_k),$$

e

$$\ell_i(u_i) = 1$$

tem-se de (3.43):

$$\Delta \ell_i(u_i) = \frac{\prod_{k=0,i}^n (u_i - u_k + 1)}{p'_{n+1}(u_i)} - 1 \qquad (3.44)$$

$$\underline{2^{0} \operatorname{caso}}: \Delta \ell_{i}\left(u_{j}\right), j \neq i$$

Como se tem $p_{n+1}(u_j) = 0$, e como $\ell_i(u_j) = 0$, $j \neq i$, obtém-se de (3.43):

$$\Delta \ell_i(u_{j \neq i}) = \frac{\prod_{k=0,i}^n (u_j - u_k + 1)}{p'_{n+1}(u_i)}$$

Mas, como para k = j se tem , $u_i - u_k + l = l$, obtém-se finalmente:

$$\Delta \ell_i(u_{j\neq i}) = \frac{\prod_{k=0, (i,j)}^n (u_j - u_k + 1)}{p'_{n+1}(u_i)} \qquad (3.45)$$

As relações (3.43), (3.44) e (3.45) valem para o cálculo das 1^{as} diferenças "forward".

3.2.3.2. Cálculo das Diferenças "Backward"

De (3.39), usando (3.35) e (3.36) tem-se:

$$\nabla f(x) \approx \sum_{j=0}^{n} \nabla \ell_{i}(x) f(u_{i}).$$
(3.46)

1. . .

Para a primeira diferença "backward", $\nabla l_i(x)$ tem-se, partindo de (3.39-40ab) e por um método análogo ao usado atrás:

$$(x - u_i - 1)\nabla \ell_i(x) + \ell_i(x) = \frac{\nabla \rho_{n+1}(x)}{\rho'_{n+1}(u_i)}$$

ou:

$$(x - u_i - 1)\nabla \ell_i(x) + \ell_i(x) = \frac{p_{n+1}(x) - p_{n+1}(x - 1)}{p'_{n+1}(u_i)}$$

ou:

$$\nabla \ell_i(x) = \frac{p_{n+1}(x) - p_{n+1}(x-1)}{(x-u_i - 1)p'_{n+1}(u_i)} - \frac{\ell_i(x)}{x-u_i - 1} \qquad (3.47)$$

Para os pontos de colocação , $x = u_j (j = 0, 1, ..., n)$, tem-se então:

<u>3° caso</u>: $\nabla \ell_i(u_i)$

Como $p_{n+1}(u_i) = 0$ e $\ell_i(u_i) = 1$, tem-se de (3.47):

$$\nabla \ell_i(u_i) = 1 - \frac{\prod_{k=0,i}^n (u_i - u_k - 1)}{p'_{n+1}(u_i)} \qquad (3.48)$$

 $\underline{4^{\circ} \operatorname{caso}}: \nabla \ell_{i} \left(u_{j} \right) j \neq i$

Como $p_{n+1}(u_j) = 0$ e $\ell_i(u_i) = 0$, tem-se de (3.47):

$$\nabla \ell_i(u_{j \neq i}) = -\frac{\prod_{k=0,i}^n (u_j - u_k - 1)}{p'_{n+1}(u_i)} ,$$

e como para $k = j \in u_j - u_k - 1 = -1$, tem-se finalmente:

$$\nabla \ell_i(u_{j \neq i}) = -\frac{\prod_{k=0,(i,j)}^n (u_j - u_k - 1)}{p'_{n+1}(u_i)} \qquad (3.49)$$

Introduzindo os vetores das diferenças $\Delta f \in \nabla f$

$$\Delta \boldsymbol{f} = \left[\Delta f(\boldsymbol{u}_0), \Delta f(\boldsymbol{u}_1), \dots, \Delta f(\boldsymbol{u}_n) \right]^T , \qquad (3.50)$$

$$\nabla \boldsymbol{f} = \left[\nabla f(\boldsymbol{u}_0), \nabla f(\boldsymbol{u}_1), \dots, \nabla f(\boldsymbol{u}_n)\right]^T, \qquad (3.51)$$
pode-se então escrever de uma maneira condensada:

$$\Delta f = Af , \qquad (3.52)$$

$$\nabla f = \mathbf{A}' f \quad . \tag{3.53}$$

As matrizes $\mathbf{A} = (A_{ij})$ e $\mathbf{A}' = (A_{ij})$ são dadas por

$$A_{ij} = \Delta \ell_j(u_i) \quad ; A'_{ij} = \nabla \ell_j(u_i), \qquad (3.54)$$

e o vetor f por

$$f = [f(u_0), f(u_1), \dots, f(u_n)]^T .$$
(3.55)

A analogia entre estas relações e as usadas para a aproximação das derivadas na Colocação Ortogonal Clássica (Villadsen e Michelsen, 1978) é evidente.

CAPÍTULO 4

MODELOS DE ORDEM REDUZIDA E SIMULAÇÃO EM COLUNAS DE DESTILAÇÃO

4. MODELOS DE ORDEM REDUZIDA E SIMULAÇÃO EM COLUNAS DE DESTILAÇÃO

4.1 Introdução

Neste capítulo será demonstrada a aplicação da colocação ortogonal discreta ao desenvolvimento de modelos de ordem reduzida para sistemas de separação por estágios. Inicialmente será apresentado o desenvolvimento do modelo completo. Em seguida, será feito o desenvolvimento da metodologia de redução de ordem, utilizando duas estratégias de colocação. Através de alguns casos exemplo, será testada a eficiência do método, onde os perfis de composição e temperatura no estado estacionário serão comparadas com aqueles obtidos pelo modelo completo.

4.2 Desenvolvimento do Modelo Completo

O desenvolvimento do modelo completo será feito tomando-se como base uma coluna de destilação multicomponente com uma única alimentação e sem retiradas laterais, conforme ilustrado na Figura 4.1



Figura 4.1 - Coluna de Destilação Multicomponente

Serão adotadas as seguintes hipóteses simplificadoras:

- (a) Holdup do vapor desprezível;
- (b) Equilíbrio termodinâmico entre o líquido e o vapor que saem de cada prato;
- (c) Pressão constante ao longo da coluna;
- (d) O condensador e o refervedor serão tratados como estágios separados.

Excluindo-se o prato de alimentação, o refervedor e o condensador, tem-se para um prato j da coluna (ilustrado pela Figura 4.2) as seguintes equações de balanço de massa e energia:



Figura 4.2 – prato j da coluna de destilação

• Equações de Balanço de Massa por componente

$$\frac{d}{dt} \left(M_{j} \frac{l_{i,j}}{L_{j}} \right) = l_{i,j+1} - l_{i,j} + v_{i,j-1} - v_{i,j}$$

$$i = 1, \dots, NC$$

$$j = 2, \dots, N - 1, (j \neq N_{F})$$
(4.1)

• Equações de Balanço de Energia

$$\frac{d}{dt} \left(M_{j} \frac{H_{i,j}}{L_{j}} \right) = H_{j+1}^{L} - H_{j}^{L} + H_{j-1}^{V} - H_{j}^{V}$$
(4.2)
$$i = 1, ..., NC$$
$$j = 2, ..., N - 1, (j \neq N_{F})$$

Por apresentarem estágios com descontinuidades nas vazões de líquido e/ou vapor, o prato de alimentação, o refervedor e o condensador são modelados separadamente:

• Prato de Alimentação

$$\frac{d}{dt} \left(M_{N_F} \frac{l_{i,N_F}}{L_{N_F}} \right) = l_{i,N_F+1} - l_{i,N_F} + v_{i,N_F-1} - v_{i,N_F} + f_i$$
(4.3)
$$i = 1, \dots, NC$$

$$\frac{d}{dt} \left(M_{N_F} \frac{H_{N_F}^L}{L_{N_F}} \right) = H_{N_F+1}^L - H_{N_F}^L + H_{N_F-1}^V - H_{N_F}^V + FH_F$$
(4.4)

• <u>Refervedor</u>

$$\frac{d}{dt}\left(M_{1}\frac{b_{i}}{B}\right) = l_{i,2} - v_{i,1} - b_{i}$$

$$i = 1, \dots, NC$$

$$(4.5)$$

$$\frac{d}{dt}\left(M_{1}\frac{H_{1}^{L}}{B}\right) = H_{2}^{L} - H_{1}^{V} - H_{1}^{L}$$
(4.6)

• Condensador

$$\frac{d}{dt}\left(M_{N}\frac{d_{i}}{D}\right) = v_{i,N-1} - l_{i,N} - d_{i}$$

$$\tag{4.7}$$

$$i = 1, ..., NC$$

$$\frac{d}{dt}\left(M_{N}\frac{H_{N}^{L}}{D}\right) = H_{N-1}^{L} - H_{N}^{L} - H_{N}^{V}$$

$$\tag{4.8}$$

Além das relações de balanço de massa e energia tem-se ainda para cada prato j as seguintes equações de equilíbrio termodinâmico e de hidráulica do prato:

• Relação de Equilíbrio Termodinâmico

$$v_{i,j} = K_{i,j} l_{i,j} \frac{V_j}{L_j}$$

$$i = 1, \dots, NC$$

$$j = 1, \dots, N$$
(4.9)

• Equação Hidráulica do Prato

$$M_{j} = \Psi L_{j} \tag{4.10}$$
$$j = 1, \dots, N$$

Onde:

$$L_{j} = \sum_{i=1}^{NC} l_{i,j}$$
(4.11)

$$V_{j} = \sum_{i=1}^{NC} v_{i,j}$$
(4.12)

$$H_{j}^{L} = \sum_{i=1}^{NC} l_{i,j} h_{i,j}^{L}$$
(4.13)

$$H_{j}^{\nu} = \sum_{i=1}^{NC} v_{i,j} h_{i,j}^{\nu}$$
(4.14)

O índice *i* denota o componente e o índice *j* o número do prato

4.3 Desenvolvimento do Modelo de Ordem Reduzida utilizando Colocação Ortogonal Mista

O modelo de ordem reduzida será desenvolvido baseado nas mesmas considerações feitas para o modelo completo. A metodologia de ordem reduzida consiste em separar a coluna em duas secções (ou elementos), esgotamento e retificação, cada qual definida como um conjunto de pratos entre o prato de alimentação e um dos extremos da coluna (refervedor e condensador), conforme ilustrado na Figura 4.3.





Para cada prato dessa secção as equações de balanço de massa energia, de equilíbrio termodinâmico e de hidráulica do prato são as definidas por (4.1), (4.2), (4.3) e (4.4). O desenvolvimento do modelo de ordem reduzida será ilustrado tomando como base a secção de retificação. Será introduzida uma variável espacial *s*, contínua no intervalo $s \in [0, M^R - 1]$, e que assume valores inteiros em cada prato $s_j = \{0, 1, ..., M^R - 1\}$, onde M^R é o número de pratos da secção, como ilustrado na Figura 4.4.

Prato
$$j+1$$

Prato j
Prato j
Prato $j-1$
Localização $s_{j+1} = s_j + 1 = j+1$
Localização $s_{j+1} = s_j + 1 = j+1$
Localização $s_{j+1} = s_j + 1 = j+1$
Localização $s_j = j$
Localização $s_j = j$
Localização $s_{j-1} = j-1 = j-1$

Figura 4.4 – Ilustração de parte da secção de retificação da coluna

Com isso as equações (4.1), (4.2), (4.3) e (4.4) podem ser reescritas para cada prato s_i da seguinte maneira:

$$\frac{d}{dt}\left(M(s=j)\frac{l_i(s=j)}{L(s=j)}\right) = l_i(s=j+1) - l_i(s=j) + v_i(s=j-1) - v_i(s=j)$$
(4.15)

$$\frac{d}{dt}\left(M(s=j)\frac{H^{L}(s=j)}{L(s=j)}\right) = H^{L}(s=j+1) - H^{L}(s=j) + H^{V}(s=j-1) - H^{V}(s=j)$$

(4.16)

$$v_i(s=j) = K_i(s=j)l_i(s=j)\frac{V(s=j)}{L(s=j)}$$
(4.17)

$$M(s=j) = \Psi(L(s=j)$$
(4.18)

Utilizando as definições de primeiras diferenças forward e backward tem-se que:

$$\Delta l_i(s=j) = l_i(s=j+1) - l_i(s=j)$$
(4.19)

$$\nabla v_i(s=j) = v_i(s=j) - v_i(s=j-1)$$
(4.20)

$$\Delta H^{L}(s=j) = H^{L}(s=j+1) - H^{L}(s=j)$$
(4.21)

$$\nabla H^{\nu}(s=j) = H^{\nu}(s=j) - H^{\nu}(s=j-1)$$
(4.22)

e as equações (4.15) e (4.16) podem ser escritas na seguinte forma de diferenças finitas:

$$\frac{d}{dt}\left(M(s=j)\frac{l_i(s=j)}{L(s=j)}\right) = \Delta l_i(s=j) - \nabla v_i(s=j)$$
(4.23)

$$\frac{d}{dt}\left(M(s=j)\frac{H^{L}(s=j)}{L(s=j)}\right) = \Delta H^{L}(s=j) - \nabla H^{V}(s=j) \qquad (4.24)$$

As equações molares dos componentes, bem como as entalpias de líquido e vapor na secção serão aproximados por funções polinomiais da variável independente associada com a posição s. Como foi visto no Capítulo 3, um polinômio de grau n que aproxima subotimamente estas funções pode ser construído através do Polinômio Interpolador de Lagrange usando os pares de valores $(s_k, l_i(s_k)), (s_k, v_i(s_k)), (s_k, H^L(s_k)), (s_k, (H^V(s_k)), k = 0, 1, ..., n, s_k \in (0, M^R - 1), onde os s_k$ são as raízes do polinômio de Hahn $h_n^{(1,1)}(x-1, M^R - 2)$.

Será utilizado na construção do polinômio interpolador um esquema de colocação ortogonal mista, pois além dos *n* pontos de colocação internos $s_1,...,s_n$, os pontos $s_0 = 0$ e $s_{n+1} = M^R - 1$, correspondentes aos pratos extremos da secção, serão também utilizados como pontos de colocação. A Figura 4.5 ilustra a localização dos pontos de colocação na secção de retificação. Assim as funções de aproximação tomam a forma:

$$l_i(s) = \sum_{k=0}^{n+1} l_k(s) l_i(s_k)$$
(4.25)

$$v_i(s) = \sum_{k=0}^{n+1} I_k(s) v_i(s_k)$$
(4.26)

$$H^{L}(s) = \sum_{k=0}^{n+1} l_{k}(s) H^{L}(s_{k})$$
(4.27)

$$H^{\nu}(s) = \sum_{k=0}^{n+1} l_k(s) H^{\nu}(s_k)$$
(4.28)

i = 1, ..., NC

onde $l_k(s)$ são os membros do Polinômio Interpolador de Lagrange.



Figura 4.5 - Localização dos pontos de colocação na secção de retificação

Como foi visto no Capítulo 3, para uma função y(x) aproximada pelo Polinômio Interpolador de Lagrange:

$$y(x) \approx \sum_{k=0}^{n} l_k(x) y(x_k),$$
 (4.29)

e a aproximação das diferenças dessa mesma função é dada por:

$$\Delta y(x) \approx \sum_{k=0}^{n} \Delta l_{k}(x) y(x_{k}), \qquad (4.30)$$

Com isso as diferenças representadas pelas equações (4.19), (4.20), (4.21) e (4.22) podem ser aproximadas por:

$$\Delta l_i(s) = \sum_{k=0}^{n+1} \Delta l_k(s) l_i(s_k)$$
(4.31)

$$\nabla v_i(s) = \sum_{k=0}^{n+1} \nabla l_k(s) v_i(s_k)$$
(4.32)

$$\Delta H^{L}(s) = \sum_{k=0}^{n+1} \Delta l_{k}(s) H^{L}(s_{k})$$
(4.33)

$$\nabla H^{\nu}(s) = \sum_{k=0}^{n+1} \nabla l_k(s) H^{\nu}(s_k)$$
(4.34)

Substituindo-se as equações (4.25), (4.26), (4.27)e (4.28), juntamente com as (4.31), (4.32), (4.33) e (4.34) em (4.15), (4.16), (4.17) e (4.18), obtém-se, para um ponto interno de colocação s_i da secção de retificação, as seguintes equações:

$$\frac{d}{dt}\left(M(s_{j})\frac{l_{i}(s_{j})}{L(s_{j})}\right) = \sum_{k=0}^{n+1} \Delta l_{k}(s_{j})l_{i}(s_{k}) - \sum_{k=0}^{n+1} \nabla l_{k}(s_{j})v_{i}(s_{k}) \quad (4.35)$$
$$\frac{d}{dt}\left(M(s_{j})\frac{H^{L}(s_{j})}{L(s_{j})}\right) = \sum_{k=0}^{n+1} \Delta l_{k}(s_{j})H^{L}(s_{k}) - \sum_{k=0}^{n+1} \nabla l_{k}(s_{j})H^{V}(s_{k}) \quad (4.36)$$

Capítulo 4 - Modelos de Ordem Reduzida e Simulação em Colunas de Destilação

$$l_{i}(s_{j}) = K_{i}(s_{j})v_{i}(s_{j})\frac{V(s_{j})}{L(s_{j})}$$
(4.37)

$$M(s_j) = \Psi(L(s_j) \tag{4.38}$$

$$i = 1, ..., NC$$

 $j = 1, ..., n$

onde:

$$L(s_j) = \sum_{i=1}^{NC} l_i(s_j)$$
(4.39)

$$V(s_j) = \sum_{i=1}^{NC} v_i(s_j)$$
(4.40)

Nota – Nas equações (4.35) a (4.40), s_j representa o ponto j de colocação (j = 0,1,...,n+1) e não a posição do prato j.

As equações (4.35) e (4.36) não são válidas nos pontos extremos da secção, s_0 e s_{n+1} , pois como poder ser visto na Figura 4.5, os pontos s_{-1} e s_{n+2} não são pontos de interpolação da secção e portanto as diferenças *forward* e *backward* não podem ser calculadas para estes pontos. Em vez disso, os pontos s_{-1} e s_{n+2} serão considerados como estágios reais e as equações para estes serão as seguintes:

$$\frac{d}{dt}\left(M(s_0)\frac{l_i(s_0)}{L(s_0)}\right) = \sum_{k=0}^{n+1} \Delta l_k(s_0)l_i(s_k) + v_{i,f} - v_i(s_0)$$
(4.41)

$$\frac{d}{dt}\left(M(s_0)\frac{H^L(s_0)}{L(s_0)}\right) = \sum_{k=0}^{n+1} \Delta l_k(s_0)H^L(s_k) + H_f^V - H^V(s_0) \quad (4.42)$$

e

$$\frac{d}{dt} \left(M(s_{n+1}) \frac{l_i(s_{n+1})}{L(s_{n+1})} \right) = l_{i,c} - l_i(s_{n+1}) - \sum_{k=0}^{n+1} \nabla l_k(s_{n+1}) v_i(s_k) \quad (4.43)$$
$$\frac{d}{dt} \left(M(s_{n+1}) \frac{H^L(s_{n+1})}{L(s_{n+1})} \right) = H_c^L - H^L(s_{n+1}) - \sum_{k=0}^{n+1} \nabla l_k(s_{n+1}) H^V(s_k) (4.44)$$

Para evitar as descontinuidades nos perfis de composição, o refervedor, o condensador e o prato de alimentação serão tratados como estágios de equilíbrio discretos, cada qual apresentando seus próprios balanços de massa e energia. As Figuras (4.6), (4.7) e (4.8) ilustram as formulações destes estágios.



Figura 4.6 - Formulação do refervedor no modelo de ordem reduzida

<u>Refervedor</u>

$$\frac{d}{dt}\left(M_B \frac{b_i}{B}\right) = l_i(s_0^E) - b_i - v_{i,r}$$
(4.45)

$$\frac{d}{dt}\left(M_{B}\frac{H_{B}^{L}}{B}\right) = H^{L}(s_{0}^{E}) - H_{B}^{L} - H_{r}^{V}$$

$$(4.46)$$



Figura 4.7 – Formulação do prato de alimentação no modelo de ordem reduzida

Prato de Alimentação

$$\frac{d}{dt}\left(M_{F}\frac{l_{i,F}}{L_{F}}\right) = l_{i}(s_{0}^{E}) + v_{i}(s_{0}^{E}) - l_{i,F} - v_{i,F} + f_{i}$$
(4.47)

$$\frac{d}{dt}\left(M_{F}\frac{H_{F}^{L}}{L_{F}}\right) = H^{L}(s_{0}^{R}) - H^{V}(s_{n+1}^{E}) - H_{F}^{L} - H_{F}^{V} + H_{F}$$
(4.48)



Figura 4.8 - Formulação do condensador no modelo de ordem reduzida

Condensador

$$\frac{d}{dt}\left(M_D \frac{d_i}{D}\right) = v_i(s_{n+1}^R) - l_{i,c} - d_i$$
(4.49)

$$\frac{d}{dt}\left(M_D \frac{H_D^L}{D}\right) = H^V(s_{n+1}^R) - H_c^L - H_D^V$$
(4.50)

Os índices R e E são usados para indicar as secções de retificação e esgotamento da coluna, respectivamente.

As equações do modelo de ordem reduzida para a secção de esgotamento da coluna poderão ser obtidas utilizando um procedimento análogo ao qual foi usado para a retificação.

4.4 Desenvolvimento do Modelo de Ordem Reduzida utilizando uma modificação da Colocação Ortogonal Mista

A estratégia de colocação ortogonal mista modificada proposta neste modelo utiliza *n* pontos de colocação internos mais um ponto extremo para o líquido e mais um ponto extremo para o vapor (ilustrado pela Figura 4.9), sendo $s_0 = 0$ para o vapor e $s_{n+1} = M^R - 1$ para o líquido, considere a secção de retificação abaixo.



Figura 4.9 - Localização dos pontos de colocação na secção de retificação

As diferenças representadas pelas equações (4.19), (4.20), (4.21) e (4.22) podem ser aproximadas por:

$$\Delta l_i(s) = \sum_{k=1}^{n+1} \Delta l_k(s) l_i(s_k)$$
(4.51)

$$\nabla v_i(s) = \sum_{k=0}^n \nabla l_k(s) v_i(s_k)$$
(4.52)

$$\Delta H^{L}(s) = \sum_{k=1}^{n+1} \Delta l_{k}(s) H^{L}(s_{k})$$
(4.53)

$$\nabla H^{\nu}(s) = \sum_{k=0}^{n} \nabla l_k(s) H^{\nu}(s_k)$$
(4.54)

Onde s_k são as raízes do polinômio de Hahn $h_n^{(0,0)}(x, M^R)$. Para este esquema de colocação não é possível determinar os melhores pontos de colocação interior pois as fórmulas de aproximação (4.51) e (4.52) são assimétricas.

Substituindo-se as equações (4.25), (4.26), (4.27)e (4.28), juntamente com as (4.51), (4.52), (4.53) e (4.54) em (4.15), (4.16), (4.17) e (4.18), obtém-se, para um ponto interno de colocação s_i da secção de retificação, as seguintes equações:

$$\frac{d}{dt}\left(M(s_{j})\frac{l_{i}(s_{j})}{L(s_{j})}\right) = \sum_{k=1}^{n+1} \Delta l_{k}(s_{j})l_{i}(s_{k}) - \sum_{k=0}^{n} \nabla l_{k}(s_{j})v_{i}(s_{k}) \quad (4.55)$$

$$\frac{d}{dt}\left(M(s_{j})\frac{H^{L}(s_{j})}{L(s_{j})}\right) = \sum_{k=1}^{n+1} \Delta l_{k}(s_{j})H^{L}(s_{k}) - \sum_{k=0}^{n} \nabla l_{k}(s_{j})H^{V}(s_{k})(4.56)$$

$$l_{i}(s_{j}) = K_{i}(s_{j})v_{i}(s_{j})\frac{V(s_{j})}{L(s_{j})} \quad (4.57)$$

$$M(s_j) = \Psi(L(s_j) \tag{4.58}$$

$$i = 1,...,NC$$

 $j = 1,...,n$

onde:

Capítulo 4 - Modelos de Ordem Reduzida e Simulação em Colunas de Destilação

$$L(s_{j}) = \sum_{i=1}^{NC} l_{i}(s_{j})$$
(4.59)

$$V(s_j) = \sum_{i=1}^{NC} v_i(s_j)$$
(4.60)

As equações (4.55) e (4.56) não são válidas nos pontos extremos da secção, s_0 e s_{n+1} , pois como poder ser visto na Figura 4.9, os pontos s_{-1} e s_{n+2} não são pontos de interpolação da secção e portanto as diferenças *forward* e *backward* não podem ser calculadas para estes pontos. Assim, os pontos s_{-1} e s_{n+2} serão considerados como estágios reais e as equações para estes serão as seguintes:

$$\frac{d}{dt}\left(M(s_0)\frac{l_i(s_0)}{L(s_0)}\right) = v_{i,f} - v_i(s_0) - \sum_{k=0}^n \nabla l_k(s_0)v_i(s_k)$$
(4.61)

$$\frac{d}{dt}\left(M(s_0)\frac{H^L(s_0)}{L(s_0)}\right) = H_f^V - H^V(s_0) - \sum_{k=0}^n \nabla l_k(s_0)H^V(s_k) \quad (4.62)$$

e

$$\frac{d}{dt}\left(M(s_{n+1})\frac{l_i(s_{n+1})}{L(s_{n+1})}\right) = \sum_{k=1}^{n+1} \Delta l_k(s_{n+1})l_i(s_k) + l_{i,c} - l_i(s_{n+1}) \quad (4.63)$$
$$\frac{d}{dt}\left(M(s_{n+1})\frac{H^L(s_{n+1})}{L(s_{n+1})}\right) = \sum_{k=1}^{n+1} \Delta l_k(s_{n+1})H^L(s_k) + H_c^L - H^L(s_{n+1})(4.64)$$

Para evitar as descontinuidades nos perfis de composição, o refervedor, o condensador e o prato de alimentação serão tratados como estágios de equilíbrio discretos, cada qual apresentando seus próprios balanços de massa e energia. As Figuras (4.10), (4.11) e (4.12) ilustram as formulações destes estágios.



Figura 4.10 - Formulação do refervedor no modelo de ordem reduzida

Refervedor

$$\frac{d}{dt}\left(M_{B}\frac{b_{i}}{B}\right) = l_{i}(s_{1}^{E}) - b_{i} - v_{i,r}$$

$$\frac{d}{dt}\left(M_{B}\frac{H_{B}^{L}}{B}\right) = H^{L}(s_{1}^{E}) - H_{B}^{L} - H_{r}^{V}$$

$$(4.66)$$



Figura 4.11 – Formulação do prato de alimentação no modelo de ordem reduzida

Prato de Alimentação

$$\frac{d}{dt}\left(M_{F}\frac{l_{i,F}}{L_{F}}\right) = l_{i}(s_{1}^{E}) + v_{i}(s_{n}^{E}) - l_{i,F} - v_{i,F} + f_{i}$$
(4.67)

$$\frac{d}{dt}\left(M_{F}\frac{H_{F}^{L}}{L_{F}}\right) = H^{L}(s_{1}^{R}) + H^{V}(s_{n}^{E}) - H_{F}^{L} - H_{F}^{V} + H_{F}$$
(4.68)



Figura 4.12 - Formulação do condensador no modelo de ordem reduzida

Condensador

$$\frac{d}{dt}\left(M_{D}\frac{d_{i}}{D}\right) = v_{i}(s_{n}^{R}) - l_{i,c} - d_{i}$$

$$(4.69)$$

$$\frac{d}{dt}\left(M_{D}\frac{H_{D}^{L}}{D}\right) = H^{V}(s_{n}^{R}) - H_{c}^{L} - H_{D}^{V}$$

$$(4.70)$$

4.5 Aplicação dos Modelos de Ordem Reduzida à Simulação de Colunas de Destilação

Para testar a eficiência dos modelos de ordem reduzida desenvolvidos anteriormente, alguns casos exemplo foram estudados, utilizando em ambos modelagem de ordem reduzida e modelagem completa. Os perfis de composição e temperatura no estado estacionário foram comparados através de simulação computacional utilizando a linguagem FORTRAN.

4.5.1 Simulação de uma coluna depropenizadora em regime estacionário

A coluna utilizada no estudo será um separador de propeno-propano descrito por Serferlis e Hrymak (1994). Este tipo de coluna caracteriza-se por apresentar elevado número de estágios, devido à baixa volatilidade entre os dois componentes, e por elevadas razões de refluxo (o qual aumenta os custos de operação). As especificações e condições operacionais são apresentadas na Tabela (4.1).

Número de pratos	175				
Prato de alimentação	61				
Composição de alimentação	propeno: 0,8973				
	propano: 0,1027				
Vazão de alimentação (Mmol/d)	1,0734				
Temperatura da alimentação (°C)	46,11				
Pressão de operação (kPa)	1860,60				
Razão de refluxo	19,7				
Vazão de destilado (Mmol/d)	0,965				
Tipo de refervedor	equilíbrio				
Tipo de condensador	eauilíbrio				

Tal	pela	a 4	.1	******	Espec	ificaçõ	ões o	lo	separador	pro	openo	-pro	pano
-----	------	-----	----	--------	-------	---------	-------	----	-----------	-----	-------	------	------

Por tratar de uma mistura de hidrocarbonetos foi escolhido o modelo termodinâmico Peng-Robinson para representar o equilíbrio líquido-vapor, sendo utilizado o pacote de propriedades termodinâmicas desenvolvido por Zemp (1995). Este pacote possui oito subrotinas primárias : PP_INIT (inicialização das propriedades físicas), PP_BUBT (temperatura de bolha), PP_BUBP (pressão de bolha), PP_DEWT (temperatura de orvalho), PP_DEWP (pressão de orvalho), PP_PT (propriedades de fase (T,P)), PP_SEK (fator de separação,volatilidades relativas) e PP_FLASH (cálculo de flash).

Foram desenvolvidos dois modelos completos utilizando-se o código desenvolvido por Fredeslund *et al.* (1977), que utiliza o método de Naphtali e Sandholm (1971), com a substituição do modelo termodinâmico de UNIQUAC para Peng-Robinson. Utilizaram-se perfis lineares de composição, vazão e temperatura como estimativas iniciais.

Para o primeiro modelo completo o sistema de equações não-lineares foi resolvido pelo método de Newton e pelo algoritmo de Thomas e para o segundo modelo completo foi utilizada a subrotina C05PBF da biblioteca NAG, através do Método Híbrido de Powell (1970), com uma tolerância igual à raiz quadrada da precisão da máquina.

A Figura 4.13.a e 4.13.b mostra os perfis dos modelos completos. A concordância dos modelos completos é muito boa.



Figura 4.13.a – Comparação entre os perfis estacionários da fração molar de líquido obtidos pelos modelos completos



Figura 4.13.b – Comparação entre os perfis estacionários da temperatura obtidos pelos modelos completos

A Tabela 4.2 compara o número de equações, número de iterações e o tempo de CPU requerido para cada modelo completo, bem como o somatório do erro quadrático da fase líquida, gasosa e da temperatura, tomando-se como referência o modelo completo resolvido pelo Método de Newton e pelo algoritmo de Thomas.

Modelo Completo	número de equações	número de iterações	Somatório do erro quadrático da fase líquida	Somatório do erro quadrático da fase vapor	Somatório do erro quadrático da temperatura	Tempo de CPU [*] (s)
Newton e Thomas (referência)	875	13		-	-	25,73
Powell	875	19	1,76x10 ⁻⁵	1,60x10 ⁻⁵	5,63x10 ⁻⁴	127,20

Tabela	4.2 -	Resultad	os
--------	-------	----------	----

* SUN SPARC station 20



Figura 4.14 – Diagrama de blocos simplificado do programa de simulação em regime estacionário

Na simulação com o modelo de ordem reduzida utilizando colocação ortogonal mista o número de pontos internos de colocação foi variado de cinco a sete na secção de esgotamento e de seis a onze na secção de retificação. O sistema de equações não-lineares foi resolvido pelo Método de Newton e pela sub-rotina MA28 da biblioteca Harwell. A sub-rotina MA28 é um método para resolver equações lineares com matriz Jacobiana esparsa e não-simétrica, possui duas sub-rotinas primárias (MA28ad e MA28cd). A sub-rotina MA28ad utiliza o processo de fatoração LU dos blocos diagonais da matriz esparsa seguida pela sub-rotina MA28cd que resolve o sistema de equações algébricas.

A melhor concordância entre o modelo completo (Newton e Thomas) e o modelo reduzido foi obtida quando se utiliza sete e onze pontos de colocação respectivamente na secção de esgotamento e retificação. As Figuras 4.15.a, 4.15.b, 4.15.c, 4.15.d, 4.15.e, 4.15.f, 4.15.g, 4.15.h, 4.15.i, 4.15.j e a Tabela 4.3 comparam os perfis obtidos com o modelo completo e o modelo reduzido. Sendo npcse o número de pontos de colocação na secção de retificação.



Figura 4.15.a – Comparação entre os perfis estacionários da fração molar de líquido obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista) com npcse = 5 e npcsr = 6



Figura 4.15.b – Comparação entre os perfis estacionários da temperatura obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista) com npcse = 5 e npcsr= 6



Figura 4.15.c – Comparação entre os perfis estacionários da fração molar de líquido obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista) com npcse = 5 e npcsr= 7



Figura 4.15.d – Comparação entre os perfis estacionários da temperatura obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista) com npcse = 5 e npcsr = 7



Figura 4.15.e – Comparação entre os perfis estacionários da fração molar de líquido obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista) com npcse = 6 e npcsr = 7



Figura 4.15.f – Comparação entre os perfis estacionários da temperatura obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista) com npcse = 6 e npcsr = 7



Figura 4.15.g – Comparação entre os perfis estacionários da fração molar de líquido obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista) com npcse = 6 e npcsr = 8



Figura 4.15.h – Comparação entre os perfis estacionários da temperatura obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista) com npcse = 6 e npcsr = 8



Figura 4.15.i – Comparação entre os perfis estacionários da fração molar de líquido obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista) com npcse = 7 e npcsr = 11



Figura 4.15.j – Comparação entre os perfis estacionários da temperatura obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista) com npcse = 7 e npcsr = 11

Modelo Reduzido (Newton e MA28)	número de equações	número de iterações	Somatório do erro quadrático	Somatório do erro quadrático	Somatório do erro quadrático	Tempo de CPU [*] (s)
Polinômio de Hahn			da fase líquida	da fase vapor	da temperatura	
$h_n^{(1,1)}(x-1, N-2)$ npcse = 5 e npcsr = 6	90	14	1,38x10 ⁻¹	1,81x10 ⁻¹	3,36x10 ⁻¹	0,49
$h_n^{(1,1)}(x-1, N-2)$ npcse = 5 e npcsr = 7	95	13	4,05x10 ⁻²	1,09x10 ⁻²	1,55x10 ⁻²	0,53
$h_n^{(1,1)}(x-1, N-2)$ npcse = 6 e npcsr = 7	100	14	1,01x10 ⁻²	0,86x10 ⁻²	0,67x10 ⁻²	0,62
$h_n^{(1,1)}(x-1, N-2)$ npcse = 6 e npcsr = 8	105	17	6,72x10 ⁻⁴	5,81x10 ⁻⁴	3,47x10 ⁻³	0,77
$h_n^{(1,1)}(x-1, N-2)$ npcse =7 e npcsr =11	125	20	3,08x10 ⁻⁵	3,15x10 ⁻⁵	3,78x10 ⁻⁵	1,51

Tabela 4.3 - Resultados

* SUN SPARC station 20

Um outro método empregado para a resolução dos sistemas de equações nãolineares foi a sub-rotina F01BRF da biblioteca NAG (Método da Eliminação de Gauss). Nesta simulação o número de pontos foi variado de seis a oito na secção de esgotamento e de sete a onze na secção de retificação. A melhor concordância entre o modelo completo (Newton e Thomas) e o modelo reduzido foi obtida quando se utiliza sete pontos de colocação na secção de esgotamento e onze pontos na secção de retificação. As Figuras 4.15.k, 4.15.l, 4.15.m, 4.15.n e a Tabela 4.4, comparam os perfis obtidos pelo modelo completo e o modelo reduzido.



Figura 4.15.k – Comparação entre os perfis estacionários da temperatura obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista) com npcse = 6 e npcsr = 8



Figura 4.15.1 – Comparação entre os perfis estacionários da fração molar de líquido obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista) com npcse = 6 e npcsr = 8



Figura 4.15.m – Comparação entre os perfis estacionários da temperatura obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista) com npcse = 7 e npcsr = 11



Figura 4.15.n – Comparação entre os perfis estacionários da fração molar de líquido obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista) com npcse = 7 e npcsr = 11

Tabela 4.4 - Resultados

Modelo Reduzido (Gauss) Polinômio de Hahn	número de equações	número de iterações	Somatório do erro quadrático da fase líquida	Somatório do erro quadrático da fase vapor	Somatório do erro quadrático da temperatura	Tempo de CPU [*] (s)
$h_n^{(1,1)}(x-1, N-2)$ npcse = 6 e npcsr = 8	105	15	8,84x10 ⁻⁴	8 ,32x10 ⁻⁴	5,34x10 ⁻³	3,95
$h_n^{(1,1)}(x-1, N-2)$ npcse =7 e npcsr =11	125	21	3,87x10 ⁻⁵	3,83x10 ⁻⁵	4,44x10 ⁻⁵	5,07

A simulação do modelo de ordem reduzida utilizando uma modificação da colocação ortogonal mista apresentou bons resultados. Os pontos de colocação internos foram variados de sete a nove na secção de esgotamento e de nove a treze na secção de retificação. A melhor aproximação com o modelo completo foi obtida com nove e treze pontos de colocação respectivamente na secção de esgotamento e retificação O sistema de equações não-lineares foi resolvido pelo Método de Newton e pela sub-rotina MA28 da biblioteca Harwell. As Figuras 4.16.a, 4.16.b, 4.16.c, 4.16.d, 4.16.e, 4.16.f e a Tabela 4.5 comparam esses perfis com o modelo completo.



Figura 4.16.a – Comparação entre os perfis estacionários da fração molar de líquido obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista modificada) com npcse = 7 e npcsr = 9



Figura 4.16.b – Comparação entre os perfis estacionários da temperatura obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista modificada) com npcse = 7 e npcsr = 9



Figura 4.16.c – Comparação entre os perfis estacionários da fração molar de líquido obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista modificada) com npcse = 8 e npcsr = 10


Figura 4.16.d - Comparação entre os perfis estacionários da temperatura obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista modificada) com npcse = 8 e npcsr = 10



Figura 4.16.e - Comparação entre os perfis estacionários da fração molar de líquido obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista modificada) com npcse = 9 e npcsr = 13



Figura 4.16.f - Comparação entre os perfis estacionários da temperatura obtidos pelo modelo completo e o modelo de ordem reduzida (colocação ortogonal mista modificada) com npcse = 9 e npcsr = 13

Modelo Reduzido (Newton e MA28) Polinômio de Hahn	número de equações	número de iterações	Somatório do erro quadrático da fase líquida	Somatório do erro quadrático da fase vapor	Somatório do erro quadrático da temperatura	Tempo de CPU [*] (s)
$h^{(0,0)}(x,N)$	105	15	3,86x10 ⁻²	4,54x10 ⁻²	5,12x10 ⁻²	0,74
npcse =7 e npcsr =9			,	, ,		-
$h_n^{(0,0)}(x,N)$	115	18	6,56x10 ⁻³	5,99x10 ⁻³	1,97x10 ⁻³	0,83
npcse =8 e npcsr=10						
$h_n^{(0,0)}(x,N)$	125	20	4,23x10 ⁻⁴	3,96x10 ⁻⁴	3,39x10 ⁻³	1,56
npcse = 9 e npcsr=13						

Tabela 4.5 - Resultados

* SUN SPARC station 20

CAPÍTULO 5

CONCLUSÕES E SUGESTÕES

5. CONCLUSÕES E SUGESTÕES

UNICAMP BIBLIOTECA CENTRA. SEÇÃO CIRCULANT

69

5.1 Conclusões

Este trabalho tinha como objetivos o desenvolvimento de modelos de ordem reduzida para sistemas de processos de separação por estágios, a seleção da melhor estratégia de colocação, assim como a seleção do algoritmo mais adequado para a solução das equações do modelo visando selecionar aquela combinação de estrutura de modelo e de algoritmo que permitisse obter uma simulação adequada e custo computacional mínimo.

Os resultados das simulações realizadas demonstram a viabilidade da aplicação da colocação ortogonal discreta na simulação de colunas de destilação em estado estacionário.

A partir da teoria dos polinômios ortogonais discretos desenvolveu-se uma metodologia de redução de ordem, que, até certo ponto unifica a aplicação do método da colocação ortogonal tanto para a solução de equações diferenciais quanto para equações às diferenças.

A aplicação da metodologia de redução de ordem a colunas de destilação de pratos permite um equacionamento mais simples e mais compacto que os modelos de ordem reduzida encontrados na literatura.

As simulações em regime estacionário demonstraram que o esquema de colocação ortogonal mista com as raízes do polinômio de Hahn $h_n^{(1,1)}(x-1,N-2)$ leva a uma melhor aproximação em relação à colocação ortogonal mista modificada que utiliza as raízes do polinômio de Hahn $h_n^{(0,0)}(x,N)$.

Com relação ao tempo computacional, os modelos de ordem reduzida levaram a uma redução de até 97% em comparação com o modelo completo que utiliza o método de Newton e de Thomas e de até 99% para o modelo completo que utiliza o método de Powell.

Colocação ortogonal discreta é um método muito útil para a redução do modelo de processos de separação por estágios. Umas das dificuldades encontradas na utilização dos modelos de ordem reduzida que utilizam a colocação ortogonal é a ausência de um regra geral que possibilite a determinação do número de pontos de colocação ideal para a

obtenção da melhor aproximação dos perfis da coluna. Assim como o fato de que a notação utilizada na teoria dos polinômios ortogonais discretos difere muito de autor para autor.

5.2 Sugestões para trabalhos futuros

• Dividir a coluna em mais de um elemento por secção com o objetivo de contornar as descontinuidades nos perfis de vazão provocadas pela alimentação, refervedor e condensador. (Colocação por elementos finitos).

• Desenvolvimento de uma metodologia que permita determinar o número de pontos de colocação no modelo de ordem reduzida.

• Aplicação dos modelos de ordem reduzida desenvolvidos neste trabalho à problemas de otimização on-line, controle preditivo com modelo e controle ótimo do tempo.

• Estender a metodologia para o estado transiente a coluna de destilação.

REFERÊNCIAS BIBLIOGRÁFICAS

Referências Bibliográficas

- APOSTOL, T.M. *Mathematical Analysis*, 2. ed., Addison-Wesley Publishing Company, 1974.
- CHO, Y.S., JOSEPH, B. Reduced-Order Steady-State and Dynamic Models for Separation Processes – I. Development of the Model Reduction Procedure. *AIChE Journal*, v.29, n.2, p. 261 – 269, 1983a.
- CHO, Y.S., JOSEPH, B. Reduced-Order Steady-State and Dynamic Models for Separation Processes – II. Application to Nonlinear Multicomponent Systems. *AIChE Journal*, v.29, n.2, p. 270 – 276, 1983b.
- CHO, Y.S., JOSEPH, B. Reduced-Order Models for Separation Columns III. Application to Columns with Multiple Feeds and Sidestreams. *Computers and Chemical Engineering*, v.8, n.2, p. 81-90, 1984.
- DROZDOWICZ, B., MARTÍNEZ, E. Reduced Models for Separation Process in Real-Time Simulators. *Computers and Chemical Engineering*, v.12, n. 6, p. 547 – 560, 1988.
- ESPAÑA, A., LANDAU, I.D. Reduced Order Bilinear Models for Distillation Columns. Automatica, v.14, p. 345 – 355, 1978.
- FINLAYSON, B.A. Nonlinear Analysis in Chemical Engineering. New York: McGraw Hill International Book Company. Cap. 2, p.11 13. Ca:4, p.73 79, 1980.
- FREDESLUND, A., GMEHLING, J., RASMUSSEN, P. Vapor-Liquid Equilibria Using UNIFAC: A Group-Contribution Method, Amsterdan: Elsevier Scientific Publishing Company, 1977.
- GEOGARKIS, C., STOEVER, M.A. Time Domain Order Reduction of Tridiagonal Dynamics of Staged Processes – I. Uniform Lumping, Chemical Engineering Science, v.37, n. 5, p. 687-697, 1982.

- JORDAN, C. Calculus of Finite Differences, 3. ed. New York: Chelsea Publishing Company, 1965.
- KREYSZIG, E. Advanced Engineering Mathematics, New York: John Wiley, 1993.
- LEVIT, R.J. The Zeros of the Hahn Polynomials, SIAM Review, v.9, n.2, p. 191 203, 1967
- MENDES, M.J. Polinômios Ortogonais Discretos e Diferenças Finitas Campinas: Documento de Circulação Interna, DESQ/FEQ, UNICAMP, 1995.
- NAG (*The Numerical Algorithms Group Limited*), The NAG Fortran Library Manual, Mark 15, 1991.
- NAPHTALI, L.M., SANDHOLM, D.P. Multicomponent Separation Calculations by Linearization. *AIChE Journal*, v.17, n. 1, p. 148 153, 1971.
- NIKIFOROV, A.F., SUSLOV., S.K. UVAROV., V.B. Classical Orthogonal Polynomials of a Discrete Variable, Berlin: Springer Verlag, 1991.
- OSBORNE, A. The Calculations of Unstead State Multicomponent Distillation Using Partial Differential Equations. *AIChE Journal*, v.17, n. 3, p. 696 703, 1971
- PINTO, J.C., BISCAIA JR. E.C. Order Reduction Strategies for Models of Staged Separations Systems. Computers and Chemical Engineering, v.12, n. 8, p. 821-831, 1988.
- POWELL, M.J.D. *A Hybrid Method for Nonlinear Algebric Equation*. In: Numerical Methods For Nonlinear Algebric Equations. Rabinowitz, P. Gordon and Breach, 1970.
- RABINOWITZ, P. *Numerical Methods for Nonlinear Algebric Equations*. Gordon and Breach, 1970.

- RAVAGNANI, S.P. Modelos de Ordem Reduzida para Processos de Separação por Destilação Multicomponente em Colunas de Pratos. Aplicação a Sistemas Complexos.
 São Paulo: EP, USP, 1988. Tese (Doutorado) - Universidade de São Paulo.
- RUGGIERO, M.AG., LOPES, V.L.R. Cálculo Numérico Aspectos Teóricos e Computacionais. Rio de Janeiro: McGraw Hill, 1988.
- SERFELIS, P., HRYMAK, A.N. Optimization of Distillation Units Using Collocation Models. AIChE Journal, v.40, n. 5, p. 813 – 825, 1994a.
- SERFELIS, P., HRYMAK, A.N. Adaptative Collocation on Finite Elements Models for Optimization of Multistage Distillation Units. *Chemical Engineering Science*, v. 49, n. 9, p. 1369 – 1382, 1994b.
- SKOGESTAD, S., MORARI, M. Understanding the Dynamic Behavior of Distillation Columns, *Industrial & Engineering Chemistry Research*, v.27, n. 10, p. 1848 – 1862, 1988.
- SRIVASTAVA, R.K., JOSEPH, B. Reduced Order Models for Separation Columns -V. Selection of Collocation Points. *Computers and Chemical Engineering*, v.9, n. 6, p. 601 – 613, 1985.
- SRIVASTAVA, R.K., JOSEPH, B. Reduced Order Models for Separation Columns IV. Treatment of Column with Multiple Feeds and Sidestreams Via Spline Fitting. *Computers and Chemical Engineering*, v.11, n.2, p. 159 – 164, 1987a.
- SRIVASTAVA, R.K., JOSEPH, B. Reduced Order Models for Separation Columns VI. Columns with Steep and Flat Composition Profiles. *Computers and Chemical Engineering*, v. 11, n. 2, p. 165 – 176, 1987b.
- STEWART, W.E., LEVIEN, K.L., MORARI, M. Simulation of Fractionation by Orthogonal Collocation. *Chemical Engineering Science*, v. 40, n. 3, p. 409 – 421, 1985.

Referências Bibliográficas

- SWARTZ, C.L.E, STEWART, W.E. A Collocation Approach to Distillation Column Design. *AIChE Journal*, v.32, n. 11, p. 1832 1838, 1986.
- SZEGÖ, G. Orthogonal Polynomials, 4th. ed. New York: American Mathematical Society, 1975.
- TAYLOR, R., LUCIA, A. Modeling and Analysis of Multicomponent Separation Processes. In: AIChE Symposium Series, v.91, 1995.
- VILLADSEN, J., MICHELSEN, M.L. Solution of Boundary-Value Problems by Orthogonal Collocation. *Chemical Engineering Science*, v. 22, p. 1483 – 1501, 1967.
- VILLADSEN, J., MICHELSEN, M.L. Solution of Differential Equation Models by Polynomial Approximation, New Jersey: Prentice-Hall, 1978.
- WAHL, E.F., HARRIOT, P. Understanding and Prediction of the Dynamic Behaviour of Distillation Column., *Industrial & Engineering Chemistry Process Design* Development, v. 9, n. 3, p. 396-407, 1970.
- WONG, K.T., LUUS, R. Model Reduction of High-Order Multistage Systems by the Method of Orthogonal Collocation. *The Canadian Journal of Chemical Enginneering*, v. 58, p. 382 – 388, 1980.
- XAVIER, A.S.E. Modelagem de Colunas de Destilação em Estado Estacionário.
 Campinas: FEQ, UNICAMP, 1992. Tese (Mestrado) Faculdade de Engenharia Química, Universidade Estadual de Campinas, 1992.
- ZEMP, R.J. Software de Propriedades Termodinâmicas. Campinas: Comissão de Informática, DESQ/FEQ, UNICAMP, 1995.

ANEXO I

ANEXO I

I. POLINÔMIOS ORTOGONAIS DISCRETOS E CÁLCULO DE DIFERENÇAS FINITAS

I.1 INTRODUÇÃO

Os polinômios ortogonais constituem uma classe de funções de grande interesse na Análise Matemática (Szegö, 1975) e, mais particularmente, na Simulação de Processos (Villadsen e Michelsen, 1978). A teoria e aplicações dos chamados <u>polinômios ortogonais</u> <u>clássicos</u> (Jacobi, Hermite, Legendre) estão expostas nas referências citadas acima.

Neste trabalho é estudada a aplicação dos chamados <u>polinômios ortogonais de</u> <u>variável discreta</u> ao Cálculo das Diferenças Finitas, visando a solução de equações às diferenças, encontradas por exemplo nos modelos matemáticos de processos de separação por estágios.

Como a teoria destas famílias de polinômios ortogonais não está ainda sistematizada, não havendo sequer uma notação unificada, será primeiramente feita uma apresentação condensada desta teoria, visando sobretudo as aplicações em engenharia. Sempre que possível a notação seguida é a apresentada por Nikiforov *et al.* (1991).

I.1.1 Polinômios Ortogonais

Para uma função $\alpha(x)$, monotônica não-decrescente no intervalo [a,b], define-se o produto escalar de duas funções reais contínuas $f(x) \in g(x), x \in$ [a,b], pela seguinte integral de Stieltjes-Lebesgue (Apostol, 1974)

$$(f,g) = \int_{a}^{b} f(x)g(x)d\alpha(x), \qquad (1)$$

e a ortogonalidade de f(x) e g(x) em relação à distribuição $d\alpha(x)$ pela condição:

$$(f,g) = \int_{a}^{b} f(x)g(x)d\alpha(x) = 0, \qquad (2)$$

As funções f(x), g(x), para as quais o produto escalar (1) existe, podem ser consideradas como "vetores" de um espaço vetorial, cujo produto interno é definido por (1), sendo a norma associada a este produto interno, dada por:

$$\|f\| = (f, f)^{1/2} (3)$$

Um conjunto finito de funções $f_0(x)$, $f_1(x)$,..., $f_n(x)$ é dito linearmente independente se a equação :

$$\left\|\lambda_0 f_0 + \lambda_1 f_1 + \ldots + \lambda_n f_n\right\| = 0$$

só for válida para $\lambda_0 = \lambda_1 = \dots = \lambda_n = 0$.

Seja $d\alpha(x)$ uma distribuição no intervalo [a,b]; supondo-se que os momentos

$$c_n = \int_a^b x^n d\alpha(x)$$
 $n = 0, 1, ..., n_1, ... (4)$

existem, então, o conjunto das potências não-negativas de x,

$$\{1, x, x^2, x^3, \dots, x^n, \dots\}$$

é linearmente independente em [a,b] em relação a $d\alpha(x)$ (Szegö 1975). A partir deste conjunto, por um processo de ortogonalização Gram-Schmidt (Szegö,1975), é possível obter um conjunto de polinômios

$$p_0(x), p_1(x), \dots, p_n(x), \dots$$

determinado unicamente pelas condições:

(a) $p_n(x)$ é um polinômio precisamente de grau *n*, para o qual o coeficiente de x^n é positivo;

(b) O sistema { $p_n(x)$ } é ortogonal, isto é, tem-se:

$$\int_{a}^{b} p_{n}(x)p_{m}(x)d\alpha(x) = \delta_{nm}d_{n}^{2} , \quad n,m = 0, 1, n, ..., \quad (5)$$

onde

$$\delta_{nm} = \begin{cases} 1, & n = m \\ 0, & n \neq m \end{cases},$$

e d_n é a norma

$$d_n^2 = (p_n, p_n) = \|p_n\|^2 = \int_a^b p_n(x)p_n(x)d\alpha(x).$$
(6)

As condições a) e b) acima definem os polinômios $p_n(x)$ em questão.a menos de constantes multiplicativas (normalização) Vários tipos de normalização podem ser e são usados, tais como: definir o valor da norma d_n ; fixar o valor de $p_n(x)$ para x = a ou x = b; fixar o valor do coeficiente de x^n em $p_n(x)$, etc.

I.1.2 Propriedades Gerais dos Polinômios Ortogonais

Os polinômios $p_n(x)$, definidos pelas condições (a) e (b) acima, formam uma <u>família de</u> <u>polinômios ortogonais em relação ao intervalo [a,b] e à distribuição $d\alpha(x)$ </u>. As famílias de polinômios ortogonais obtidas apresentam um conjunto de propriedades gerais. De uma maneira geral, o polinômio $p_n(x)$, que é suposto ser exatamente de grau <u>n</u>, será representado, na sua forma expandida, por:

$$p_n(x) = \sum_{k=0}^{n} a_{n,k} x^k , \qquad (7)$$

onde $a_{n,n}$ é diferente de zero e positivo.

-<u>Qualquer polinômio</u> $q_n(x)$ de grau n pode ser representado por uma combinação linear dos polinômios ortogonais $p_k(x), k = 0, 1, ..., n$, isto é Anexo I

$$q_{n}(x) = \sum_{k=0}^{n} c_{k,n} p_{k}(x).$$
 (8)

$$\int_{a}^{b} a_{n}^{2}(x) d\alpha = \left\| a_{n}(x) \right\|^{2} = \sum_{k=0}^{n} \int_{a}^{b} c_{k,n} p_{k} p_{n} d\alpha = \begin{cases} 0, k \neq n \\ 1, k = 0 \end{cases}$$

Usando as relações de ortogonalidade (5) obtém-se:

$$c_{k,n} = \frac{1}{d_k^2} \int_a^b q_n(x) p_k(x) d\alpha(x).$$
(9)

- A relação de ortogonalidade (5) é equivalente a:

$$\int_{a}^{b} p_n(x) x^m d\alpha(x) = 0 \quad (m < n) .$$
 (10)

Com efeito, expandindo x^{m} em termos dos $p_{k}(x)$ conforme (8) ter-se-á:

$$x^{m}=\sum_{k=0}^{m}c_{k,m}\,p_{k}\left(x\right),$$

e a condição de ortogonalidade (5), para m<n , leva imediatamente a (10).

Uma consequência imediata de (10) é que o polinômio $p_n(x)$ é ortogonal em relação a qualquer polinômio de grau m, $q_m(x)$, sempre que for m < n.

Uma outra consequência imediata da relação de ortogonalidade (5) é:

$$\int_{a}^{b} x^{n} p_{n}(x) d\alpha(x) = \frac{1}{a_{n,n}} \int_{a}^{b} p_{n}(x) p_{n}(x) d\alpha(x) = \frac{(p_{n}, p_{n})}{a_{n,n}}.$$
 (11)

82

- <u>Relação de Recorrência</u> - Todas as famílias de polinômios ortogonais satisfazem a uma relação de recorrência do tipo:

$$x p_n(x) = \alpha_n p_{n+1}(x) + \beta_n p_n(x) + \gamma_n p_{n-1}(x) , \qquad (12)$$

onde α_n , $\beta_n \in \gamma_n$ são constantes.

Com efeito, consideremos o polinômio de grau n+1

$$q_{n+1}(x) = x p_n(x).$$

Expandindo $q_{n+1}(x)$ em termos dos $p_k(x)$ tem-se

$$q_{n+1}(x) = x p_n(x) = \sum_{k=0}^{n+1} c_{k,n+1} p_k(x), \qquad (13)$$

onde

$$c_{k,n+1} = \frac{1}{d_k^2} \int_{a}^{b} [x \, p_n(x)] p_k(x) \, d\alpha(x) = \frac{1}{d_k^2} \int_{a}^{b} p_n(x) [x \, p_k(x)] \, d\alpha(x). \quad (14)$$

Como $x p_k(x)$ é um polinômio de grau k+1, conclui-se de (14) que os coeficientes $c_{k,n+1}$ são nulos a menos que se tenha $k+1 \ge n$, ou seja, (13) reduz-se a:

$$x p_n(x) = c_{n+1,n+1} p_{n+1}(x) + c_{n,n+1} p_n(x) + c_{n-1,n+1} p_{n-1}(x).$$
(15)

Comparando (12) com (15) tem-se $\alpha_n = c_{n+1,n+1}$, $\beta_n = c_{n,n+1}$, $\gamma_n = c_{n-1,n+1}$.

Comparando-se os termos em x^{n+1} em ambos os membros de (12) tem-se imediatamente:

$$\alpha_n = \frac{a_{n,n}}{a_{n+1,n+1}},\tag{16}$$

onde os $a_{n,n}$ são os coeficientes dos termos de maior ordem da expansão de $p_n(x)$ dada por (7).

Anexo I

Por outro lado, de (14) obtém-se:

$$\beta_n = c_{n,n+1} = \frac{1}{d_n^2} \int_a^b x \, p_n(x) \, p_n(x) \, d\alpha(x) = \frac{(x \, p_n, p_n)}{d_n^2}.$$
 (17)

Alternativamente, por comparação dos termos em x^n em ambos os membros de (12) obtém-se:

$$a_{n,n-1} = \alpha_n a_{n+1,n} + \beta_n a_{n,n} ,$$

ou, com (16):

$$\beta_n = \frac{a_{n,n-1}}{a_{n,n}} - \frac{a_{n+1,n}}{a_{n+1,n+1}}.$$
 (18)

Finalmente tem-se:

$$\gamma_{n} = c_{n-1,n+1} = \frac{1}{d_{n-1}^{2}} \int_{a}^{b} p_{n}(x) [x p_{n-1}(x)] d\alpha(x),$$

relação que se reduz a

$$\gamma_n = \frac{a_{n-1,n-1}}{d_{n-1}^2} \int_a^b p_n(x) x^n d\alpha(x),$$

ou, com (11)

$$\gamma_n = \frac{a_{n-1,n-1}}{a_{n,n}} \frac{d_n^2}{d_{n-1}^2}.$$
 (19)

- Da relação de recorrência (12) deduz-se (Szego,1975) a chamada <u>Identidade de</u> <u>Christoffel-Darboux</u> :

$$\sum_{k=0}^{n} \frac{p_k(x)p_k(y)}{d_k^2} = \frac{a_{n,n}}{d_n^2 a_{n+1,n+1}} \frac{p_{n+1}(x)p_n(y) - p_n(x)p_{n+1}(y)}{(x-y)}$$
(20)

- Zeros dos Polinômios - Todos os zeros (raízes) x_i de $p_n(x)$ são reais e simples, e situados no intervalo (a,b).

Se a função $\alpha(x)$ em (5) for absolutamente contínua ,a condição de ortogonalidade reduz-se a:

$$\int_{a}^{b} p_{n}(x) p_{m}(x) w(x) dx, \qquad (21)$$

onde $w(x) = d\alpha/dx$, uma função não negativa, mensurável no sentido de Lebesgue. para a qual se tem $\int_{a}^{b} w(x)dx > 0$, é chamada de <u>função peso</u>.

Os <u>polinômios ortogonais clássicos</u> (Jacobi, Hermite, Legendre, etc) são definidos a partir de uma condição de ortogonalidade do tipo (21).

1.2 POLINÔMIOS ORTOGONAIS DISCRETOS

Seja $d\alpha(x)$ uma distribuição no intervalo [a,b]. Supondo-se que $\alpha(x)$ é <u>uma função degrau</u> <u>em [a,b]</u>, isto é, uma função constante por partes, apresentando um número finito de saltos (degraus) em [a,b], e se for N o número de tais degraus, então ter-se-á:

$$d\alpha(x) = \sum_{i=0}^{N-1} \rho(x_i) \delta(x - x_i), \qquad (22)$$

onde $\rho(x_i)$ é a amplitude do degrau de $\alpha(x)$ em $x = x_i$, (i = 0, 1, ..., N - 1). Nestas condições, a condição de ortogonalidade (5) toma a forma:

$$\sum_{i=0}^{N-1} p_n(x_i) p_m(x_i) \rho(x_i) = \delta_{nm} d_n^2.$$
 (23)

Os polinômios { $p_n(x)$ } que satisfazem a uma relação de ortogonalidade do tipo (23) são chamados de <u>polinômios ortogonais de uma variável discreta</u> (ou também de <u>polinômios</u> ortogonais discretos).

Note-se que estes polinômios $p_n(x)$ são ,na realidade, funções contínuas da variável contínua x; a designação "discreta" está associada às relações do tipo (23) e a outras propriedades destas famílias de polinômios.

Como a condição de ortogonalidade (23) não é mais do que um caso particular do integral de Stieltjes-Lebesgue (5), as famílias de polinômios ortogonais definidas por uma condição de ortogonalidade do tipo (23) gozam de todas as propriedades gerais dos polinômios ortogonais definidas no parágrafo **I.1**.

I.2.1 Os Polinômios de Hahn

Dentre os polinômios ortogonais de uma variável discreta, os mais conhecidos são os polinômios de **Hahn** (Nikiforov *et al.*,1991), para os quais na relação de ortogonalidade (23) se tem

$$x_{i+1} = x_i + 1$$
 , (24)

isto é, os pontos de descontinuidade de $\alpha(x)$ formam uma malha uniforme.

De acordo com Nikiforov *et al.*, os polinômios de Hahn, $p_n(x) = h_n^{(\alpha,\beta)}(x,N)$, são definidos pela condição de ortogonalidade

$$p_n(x,\alpha,\beta,N) = h_n^{(\alpha,\beta)}(x,N) \qquad , (25)$$

$$\sum_{x_i=0}^{N-1} h_n^{(\alpha,\beta)}(x_i,N) h_m^{(\alpha,\beta)}(x_i,N) \rho(x_i) = \delta_{nm} d_n^2 \qquad , (26)$$

sendo $\rho(x)$ a <u>função peso</u>

$$\rho(x) = \frac{\Gamma(N+\alpha-x)\Gamma(x+\beta+1)}{\Gamma(x+1)\Gamma(N-x)} \qquad . \tag{27}$$

onde $\Gamma(x)$ é a Função Gama, definida para x > 0 pela integral:

$$\Gamma(x) = \int_{0}^{\infty} e^{-t} t^{x-1} dt$$
(28)

<u>*n* e N são inteiros positivos</u>, sendo *n* a ordem (grau) do polinômio, com $n \le (N - 1)$. Os parâmetros $\alpha \in \beta$ são números reais, que satisfazem à condição:

$$\alpha, \beta > -1 \qquad (29)$$

A condição de ortogonalidade (26), juntamente com a função peso (27), define os polinômios $h_n^{(\alpha,\beta)}(x,N)$, a menos de uma constante multiplicativa.

-<u>Equação às Diferenças:</u> Os polinômios de Hahn, tal como definidos por Nikiforov *et al.*, são soluções da equação às diferenças

$$(x+\beta+1)(N-x-1)h_{n}(x+1)+x(N+\alpha-x)h_{n}(x-1) = = [x(N+\alpha-x)+(x+\beta+1)(N-x-1)-n(n+\alpha+\beta+1)]h_{n}(x),$$
(30)

- Zeros (Raízes) dos polinômios de Hahn: Em numerosas aplicações o conhecimento dos zeros dos polinômios $h_n^{(\alpha,\beta)}(x,N)$ é fundamental. É necessário pois desenvolver um método explícito e preciso para o cálculo destes zeros.

Consideremos a família de polinômios $h_n^{(\alpha,\beta)}(x,N)$ com a expansão em série de potência

$$h_n^{(\alpha,\beta)}(x,N) = \sum_{k=0}^n a_{n,k} x^k \qquad , \qquad (31)$$

Estes polinômios, assim como os outros polinômios ortogonais, satisfazem à relação de recorrência (12). Se normalizarmos os polinômios $h_n^{(\alpha,\beta)}(x,N)$ pela condição de se ter o coeficiente da potência x^n igual à unidade, isto é, se considerarmos a família de polinômios de Hahn $\{p_n(x)\}$

$$p_n(x) = \frac{h_n^{(\alpha,\beta)}(x,N)}{a_{n,n}} = x^n + b_{n,n-1}x^{n-1} + \dots + b_{n,0} , \qquad (31a)$$

é fácil de ver que os polinômios $p_n(x)$ definidos por (31a) satisfarão à condição de ortogonalidade

$$\sum_{x_i=0}^{N-1} p_n(x_i) p_m(x_i) \rho(x_i) = \frac{1}{a_{n,n}^2} \sum_{x_i=0}^{N-1} h_n^{(\alpha,\beta)}(x_i,N) h_m^{(\alpha,\beta)}(x_i,N) \rho(x_i) = \delta_{mn} \frac{d_n^2}{a_{n,n}^2} = h_n^2 \delta_{mn}, \quad (31b)$$

A Tabela 1 apresenta os dados característicos para os polinômios de Hahn, tal como definidos por (31a) e (31b).

<u>Tabela 1</u> - Dados para os polinômios de Hahn $h_n^{(\alpha,\beta)}(x,N)$ e de Chebyshev $t_n(x,N)$ (adaptado de Nikiforov,1991)

	$h_n^{(\alpha,\beta)}(x,N) (*)$	$t_n(x, N)$
(a, b)	(0, N)	(0, N)
<i>p</i> (<i>x</i>)	$\frac{\Gamma(N+\alpha-x)\Gamma(\beta+1+x)}{\Gamma(x+1)\Gamma(N-x)}, (\alpha,\beta>-1)$	1,0
a _{n,n} (* *)	1,0	1,0
h _n ²	$\frac{n!(\alpha+n)!(\beta+n)!(\alpha+\beta+N+n)!(\alpha+\beta+n)!}{(N-n-1)!(\alpha+\beta+2n+1)(\alpha+\beta+2n)^2!}$	$\frac{(n!)^4 (N+n)!}{(2n+1)(2n!)^2 (N-n-1)!}$
αη	1,0	1,0
β _n	$\frac{\left[(\alpha+\beta)(\beta+1)(N-1)+n(\alpha+\beta+n+1)(\alpha-\beta+2N-2)\right]}{(\alpha+\beta+2n)(\alpha+\beta+2n+2)}$	<u>N-1</u> 2
Ϋ́n	$\frac{n(\alpha+n)(\beta+n)(\alpha+\beta+N+n)(\alpha+\beta+n)(N-n)}{(\alpha+\beta+2n)^2(\alpha+\beta+2n+1)(\alpha+\beta+2n-1)}$	$\frac{n^2 (N^2 - n^2)}{4(2n+1)(2n-1)}$

(*) α,β inteiros

(**) $a_{n,n}$ é o coeficiente do termo em x^n de $h_n^{(\alpha,\beta)}(x,N)$

Anexo I

A relação de recorrência (12) para os polinômios $p_n(x)$ normalizados de acordo com (31a) pode então escrever-se sob a forma:

$$p_{n+1}(x) = (x - \beta_n)p_n(x) - \gamma_n p_{n-1}(x)$$
, (32)

onde β_n e γ_n são dados na Tabela 1.

Usando a relação (32) é possível calcular facilmente o valor de $p_n(x)$ para qualquer valor de x, assim como o valor da derivada $p'_n(x)$ e os zeros de $p_n(x)$, usando o método apresentado por Villadsen e Michelsen(1978). Os algoritmos para o cálculo das raízes de $p_n(x)$ e de $p'_n(x)$ são apresentados no <u>Anexo I</u>.

Para completar a introdução aos zeros dos polinômios de Hahn é interessante apresentar algumas propriedades gerais referentes ao seus zeros (Nikiforov, 1991).

1) Para $\alpha > -1 \ e \ \beta > -1 \ e \ n \le N-1$ os zeros dos polinômios de Hahn $h_n^{(\alpha,\beta)}(x,N)$ são todos reais e simples e estão localizados no intervalo aberto (0, N-1), sendo que para $x_i \in (0,N-1)$ existe no máximo um zero no intervalo fechado $[x_i,x_{i+1}]$.

2) Para n = N, ou seja para o polinômio de Hahn $h_N^{(\alpha,\beta)}(x,N)$, os zeros são os inteiros $\{0, 1, 2, ..., N-1\}$;

3) Para $n \ge N$ os polinômios de Hahn $h_n^{(\alpha,\beta)}(x,N)$ têm, além de outros, os zeros $\{0, 1, 2, ..., N-1\}$;

A função peso $\rho(x)$ definida por (3.7) é simétrica em relação às trocas $\alpha \leftrightarrow \beta$ e $x \leftrightarrow N-1-x$, isto é, tem-se:

$$\rho(\alpha,\beta,x,N) = \rho(\beta,\alpha,N-1-x,N)$$
 (33)

De (33) pode-se deduzir a relação de simetria para os polinômios de Hahn:

$$h_n^{(\alpha,\beta)}(x,N) = (-1)^n h_n^{(\beta,\alpha)}(N-1-x,N)$$
 (34)

A relação (34) mostra que, se x_i é um zero do polinômio $h_n^{(\alpha,\beta)}(x,N)$, então (N-1-x)é um zero do polinômio $h_n^{(\beta,\alpha)}(x,N)$.

Os polinômios de Hahn com $\alpha = \beta = 0$, isto é, os polinômios $h_n^{(0,0)}(x, N)$, são por vários autores chamados de **Polinômios de Chebyshev Discretos** e representados por

$$t_n(x,N) = h_n^{(0,0)}(x,N)$$
 (35)

A relação (34) para $\alpha = \beta = 0$ mostra que, para estes polinômios, <u>as raízes estão dispostas</u> <u>simetricamente no intervalo (0, N-1)</u>, isto é, para cada raiz x_i existe uma raiz $N-1-x_i$. Além disso, estes polinômios gozam de outra propriedade muito interessante, a saber, que para eles se tem:

$$\rho(x_i) = 1 \qquad (36)$$

Anexo I

I.3 FÓRMULAS DE QUADRATURA

No cálculo aproximado de integrais definidos e de somas de um grande número de termos recorre-se frequentemente a Fórmulas de Quadratura. O uso de fórmulas de quadratura para a aproximação de integrais definidos é apresentado por Villadsen e Michelsen (1978) e por Krylov e Stroud (1962). Aqui será feito o desenvolvimento de fórmulas de quadratura para a aproximação de somas de um elevado número de termos, baseadas nas propriedades dos polinômios ortogonais discretos, tal como sugerido por Nikiforov et al. (1991).

I.3.1 Fórmulas de Quadratura de Gauss

Consideremos uma soma S_N da forma

$$S_N = \sum_{i=0}^{N-1} \rho(x_i) f(x_i) , \qquad (37)$$

onde f(x) é uma função qualquer, contínua em $[x_0, x_{N-1}]$ e $\rho(x)$ uma função peso nãonegativa em $[x_0, x_{N-1}]$.

As fórmulas de quadratura mais simples e mais frequentemente usadas são as chamadas *Fórmulas de Quadratura de Gauss*, que aproximam a soma (37) por uma combinação linear de valores de f(x),

$$\sum_{i=0}^{N-1} \rho(x_i) f(x_i) \approx \sum_{j=1}^{n} \omega_j f(u_j) , \qquad (38)$$

onde os u_j (j = 1, ..., n) são determinados valores de x contidos no intervalo (x_0, x_{N-1}) .

<u>Propriedade 1</u>- Se os *n* pontos u_j contidos no intervalo (x_0, x_{N-1}) forem os zeros do polinômio $p_n(x)$ pertencente à família de polinômios ortogonais discretos definida pela condição de ortogonalidade

$$(p_n, p_m) = \sum_{i=0}^{N-1} \rho(x_i) p_n(x_i) p_m(x_i) = h_n^2 \delta_{mn} , \quad (39)$$

a aproximação (38) é exata para qualquer polinômio f(x) de grau $\leq 2n-1$.

-Consideremos, com efeito, o polinômio interpolador de grau n-1

$$\pi_{n-1}(x) = \sum_{j=1}^{n} \ell_j(x) f(u_j), \qquad (40.a)$$

com

$$\ell_j(x) = \frac{\rho_n(x)}{(x - u_j)\rho'_n(u_j)},\tag{40.b}$$

e

$$p_n(x) = (x - u_1)(x - u_2)...(x - u_n),$$
 (40.c)

onde $p_n(x)$ é o polinômio de grau *n* pertencente à família de polinômios ortogonais definida por (39) e os u_j (j = 1, ..., n) são os zeros de p_n .

Seja f(x) o polinômio de grau 2n-1

$$f(x) = \pi_{n-1}(x) + g_{n-1}(x) p_n(x), \qquad (41)$$

onde $g_{n-1}(x)$ é um polinômio arbitrário de grau *n-1*. De (41) obtém-se:

$$\sum_{i=0}^{N-1} \rho(x_i) f(x_i) = \sum_{i=0}^{N-1} \rho(x_i) \pi_{n-1}(x_i) + \sum_{i=0}^{N-1} \rho(x_i) g_{n-1}(x_i) \rho_n(x_i).$$
(42)

Mas, como $g_{n-1}(x)$ é um polinômio de grau n-1 < n, tem-se, atendendo às propriedades dos polinômios ortogonais

$$\sum_{i=0}^{N-1} \rho(x_i) g_{n-1}(x_i) \rho_n(x_i) = 0$$

Anexo I

e portanto

$$\sum_{i=0}^{N-1} \rho(x_i) f(x_i) = \sum_{i=0}^{N-1} \rho(x_i) \pi_{n-1}(x_i), \qquad (43)$$

ou, com (40)

$$\sum_{i=0}^{N-1} \rho(x_i) f(x_i) = \sum_{i=0}^{N-1} \rho(x_i) \left\{ \sum_{j=1}^{n} \frac{\rho_n(x_i)}{(x_i - u_j) p'_n(u_j)} f(u_j) \right\},$$

ou

$$\sum_{i=0}^{N-1} \rho(x_i) f(x_i) = \sum_{j=1}^{n} \frac{f(u_j)}{p'_n(u_j)} \left\{ \sum_{i=0}^{N-1} \rho(x_i) \frac{p_n(x_i)}{(x_i - u_j)} \right\},$$

ou, finalmente

$$\sum_{i=0}^{N-1} \rho(x_i) f(x_i) = \sum_{j=1}^n \omega_j f(u_j),$$
(44)

com

$$\omega_{j} = \frac{1}{p_{n}'(u_{j})} \sum_{i=0}^{N-1} \rho(x_{i}) \frac{p_{n}(x_{i})}{(x_{i} - u_{j})}.$$
(45)

- <u>Cálculo dos</u> ω_j :

Usando a Identidade de Christoffel-Darboux para a família de polinômios $\{p_n(x)\}$ tem-se

$$\frac{1}{h_n^2} \frac{\rho_{n+1}(x_i) \rho_n(u_j) - \rho_{n+1}(u_j) \rho_n(x_i)}{x_i - u_j} = \sum_{k=0}^n \frac{\rho_k(x_i) \rho_k(u_j)}{h_k^2},$$

ou, como $p_n(u_j)=0$

$$\frac{p_n(x_i)}{x_i - u_j} = -\frac{h_n^2}{p_{n+1}(u_j)} \sum_{k=0}^n \frac{p_k(x_i)p_k(u_j)}{h_k^2},$$

o que ,por substituição em (45) dá:

$$\omega_{j} = -\frac{h_{n}^{2}}{\rho_{n}'(u_{j})\rho_{n+1}(u_{j})} \sum_{k=0}^{n} \frac{\rho_{k}(u_{j})}{h_{k}^{2}} \left\{ \sum_{i=0}^{N-1} \rho(x_{i})\rho_{k}(x_{i}) \right\}.$$
(46)

Contudo, pelas propriedades dos polinômios ortogonais tem-se

$$\sum_{i=0}^{N-1} \rho(x_i) p_k(x_i) = \sum_{i=0}^{N-1} \rho(x_i) x_i^0 p_k(x_i) = \begin{cases} 0, \ k > 0 \\ h_k^2, k = 0 \end{cases}$$

pelo que, com $p_0(x) = 1$, se tem finalmente de (46):

$$\omega_j = -\frac{h_n^2}{\rho'_n(u_j)\rho_{n+1}(u_j)}.$$
(47)

A relação (47) pode ainda ser colocada numa forma alternativa. Atendendo-se à relação de recorrência

$$p_{n+1}(x) = (x - \beta_n)p_n(x) - \gamma_n p_{n-1}(x),$$

e a que u_j é um zero de $p_n(x)$, tem-se

$$p_{n+1}(u_j) = -\gamma_n p_{n-1}(u_j) = -\frac{h_n^2}{h_{n-1}^2} p_{n-1}(u_j),$$

e portanto ,substituindo-se em (47) tem-se:

$$\omega_{j} = \frac{h_{n-1}^{2}}{p_{n}'(u_{j})p_{n-1}(u_{j})}.$$
(48)

I.3.2 Fórmulas de Quadratura de Radau

As fórmulas de quadratura de Gauss usam para a aproximação da soma S_N uma combinação linear de valores de f(x) calculados para pontos contidos no intervalo (x_0, x_{N-1}) . As fórmulas de quadratura ditas de Radau usam na aproximação de S_N além dos pontos $u_j \in (x_0, x_{N-1}) j = 1, ..., n$, o valor de f(x) calculado num dos extremos do intervalo, $f(x_0) ou f(x_{N-1})$. Assim, uma fórmula de quadratura de Radau será, por exemplo:

$$S_{N} = \sum_{i=0}^{N-1} \rho(x_{i}) f(x_{i}) \approx \sum_{j=1}^{n} \omega_{j} f(u_{j}) + \omega_{N-1} f(x_{N-1}).$$
(49)

<u>Propriedade 2</u>-A aproximação (49) é exata para qualquer polinômio f(x) de grau $\leq 2n$ se os *n* pontos $u_j \in (x_0, x_{N-1})$ forem os zeros do polinômio $p_n(x)$ pertencente à família de polinômios ortogonais discretos definida pela condição de ortogonalidade

$$(p_n, p_m) = \sum_{i=0}^{N-1} \rho(x_i) (x_{N-1} - x_i) p_n(x_i) p_m(x_i) = h_n^2 \,\delta_{mn} \,. \tag{50}$$

A demonstração segue o mesmo caminho que a feita para o caso das fórmulas de quadratura de Gauss, partindo-se do polinômio interpolador de grau n, $\pi_n(x)$

$$\pi_n(x) = \sum_{j=1}^{n+1} \ell_j(x) f(u_j), \qquad (51.a)$$

onde $u_{n+1} = x_{N-1}$ e

$$\ell_j(x) = \frac{P_{n+1}(x)}{(x - u_j)P'_{n+1}(u_j)},$$
(51.b)

com

$$P_{n+1}(x) = (x - u_1)(x - u_2)...(x - u_n)(x - x_{N-1}) = (x - x_{N-1})p_n(x).$$
(51.c)

e construindo-se o polinômio arbitrário de grau 2n

$$f(x) = \pi_n(x) + g_{n-1}(x)P_{n+1}(x).$$
(52)

É fácil de ver que neste caso se tem

$$\sum_{i=0}^{N-1} \rho(x_i) f(x_i) = \sum_{j=1}^{n+1} \omega_j f(u_j),$$
(53)

com

$$\omega_{j} = \frac{1}{P_{n+1}'(u_{j})} \sum_{i=0}^{N-1} \rho(x_{i}) \frac{P_{n+1}(x_{i})}{x_{i} - u_{j}}, \qquad (54)$$

sendo $\omega_{n+1} = \omega_{N-1}$.

A analogia de (54) com (45) é evidente.

<u>Cálculo dos</u> ω_j

-<u>1⁰ caso</u>: $j \neq n + 1 (u_j \neq x_{N-1})$. Neste caso pode-se seguir o procedimento usado para o cálculo dos ω_j das fórmulas de quadratura de Gauss obtendo-se a partir de (54)

$$\omega_{j} = \frac{h_{n-1}^{2}}{\left(x_{N-1} - u_{j}\right)p_{n}'\left(u_{j}\right)p_{n-1}\left(u_{j}\right)}, (j \neq n+1).$$
(55)

- 2° caso: $j = n + 1 (u_j = x_{N-1})$. Neste caso tem-se:

$$P'_{n+1}(x_{N-1}) = p_n(x_{N-1}),$$

e

$$\frac{P_{n+1}(x_i)}{x_i - x_{N-1}} = p_n(x_i),$$

pelo que de (54) se obtem:

$$\omega_{n+1} = \omega_{N-1} = \frac{1}{\rho_n(x_{N-1})} \sum_{i=0}^{N-1} \rho(x_i) \rho_n(x_i).$$
 (56)

O caso correspondente à aproximação

$$S_N = \sum_{i=0}^{N-1} \rho(x_i) f(x_i) \approx \sum_{j=1}^n \omega_j f(u_j) + \omega_0 f(x_0), \qquad (57)$$

reduz-se a (49) com a mudança de variável

$$z = x_{N-1} + x_0 - x$$
.

Anexo I

Portanto neste caso os u_j serão os zeros do polinômio $p_{h}(x)$ definido pela condição de ortogonalidade

$$(p_n, p_m) = \sum_{i=0}^{N-1} \rho(x_i)(x_i - x_0) p_n(x_i) p_m(x_i) = h_n^2 \delta_{mn} .$$
 (58)

I.3.3 Fórmulas de Quadratura de Lobatto

Neste caso a soma S_N é aproximada por uma combinação linear de valores de f(x)incluindo os dois pontos extremos $x_0 \in x_{N-1}$, isto é, tem-se

$$S_{N} = \sum_{i=0}^{N-1} \rho(x_{i}) f(x_{i}) \approx \sum_{j=1}^{n} \omega_{j} f(u_{j}) + \omega_{0} f(x_{0}) + \omega_{N-1} f(x_{N-1}).$$
(59)

<u>Propriedade 3</u>-A aproximação (59) é exata para qualquer polinômio f(x) de grau $\leq 2n + 1$ se os *n* pontos $u_j \in (x_0, x_{N-1})$ forem os zeros do polinômio de grau $n p_n(x)$ pertencente à família de polinômios ortogonais discretos definida pela condição de ortogonalidade

$$(p_n, p_m) = \sum_{i=0}^{N-1} \rho(x_i)(x_i - x_0)(x_{N-1} - x_i)p_n(x_i)p_m(x_i) = h_n^2 \delta_{mn} .$$
(60)

A demonstração segue o mesmo caminho que anteriormente, partindo-se do polinômio interpolador de grau n+1

$$\pi_{n+1}(x) = \sum_{j=0}^{n+1} \ell_j(x) f(u_j), \qquad (61.a)$$

com

$$\ell_j(x) = \frac{P_{n+2}(x)}{(x - u_j)P'_{n+2}(u_j)},$$
(61.b)

e

$$P_{n+2}(x) = (x - x_0)(x - u_1)...(x - u_n)(x - x_{N-1}) = (x - x_0)(x - x_{N-1})p_n(x).$$
(61.c)

Construindo-se o polinômio arbitrário de grau 2n+1

$$f(x) = \pi_{n+1}(x) + g_{n-1}(x)P_{n+2}(x),$$

é fácil ver que se tem

$$S_{N} = \sum_{i=0}^{N-1} \rho(x_{i}) f(x_{i}) = \sum_{j=0}^{n+1} \omega_{j} f(u_{j}), \qquad (62)$$

com, de uma maneira geral

$$\omega_{j} = \frac{1}{P_{n+2}'(u_{j})} \sum_{i=0}^{N-1} \rho(x_{i}) \frac{P_{n+2}(x_{i})}{(x_{i} - u_{j})}.$$
(63)

<u>Cálculo dos</u> ω_i :

- $\underline{1^0 \text{ caso:}} \quad j \neq 0, n+1 \left(u_j \neq x_0, x_{N-1} \right)$. Neste caso pode-se seguir um procedimento análogo ao usado para o cálculo para as fórmulas de quadratura de Gauss, recorrendo à identidade de Christoffel-Darboux, obtendo-se a partir de (63):

$$\omega_{j} = \frac{h_{n-1}^{2}}{\left(u_{j} - x_{0}\right)\left(x_{N-1} - u_{j}\right)p_{n}'\left(u_{j}\right)p_{n-1}\left(u_{j}\right)}, j=1,2,...,n.$$
(64)

- 2° caso: $j = 0 (u_j = x_0)$. Neste caso tem-se diretamente de (3.63)

$$\omega_0 = \frac{1}{(x_{N-1} - x_0) \rho_n(x_0)} \sum_{i=0}^{N-1} \rho(x_i) (x_{N-1} - x_i) \rho_n(x_i) .$$
 (65)

-<u>3⁰ caso:</u> $j = n + 1 (u_j = x_{N-1})$. Neste caso tem-se também diretamente de (63)

$$\omega_{N-1} = \frac{1}{(x_{N-1} - x_0)\rho_n(x_{N-1})} \sum_{i=0}^{N-1} \rho(x_i)(x_i - x_0)\rho_n(x_i) .$$
 (66)

II. CÁLCULO DOS ZEROS DOS POLINÔMIOS ORTOGONAIS DISCRETOS

Para construir o algoritmo para o cálculo dos zeros dos <u>polinômios ortogonais</u> <u>discretos</u> $\{p_n(x)\}$ definidos por (3.25), isto é,dos polinômios de $h_n^{(\alpha,\beta)}(x,N)$ normalizados de tal modo que o coeficiente do termo em x^n é igual a 1, parte-se da relação de recorrência (3.12), que é aqui colocada da forma:

$$p_{j} = (x - g_{j})p_{j-1} - h_{j}p_{j-2}$$

(A1.1)

De (A1.1) obtém-se para as derivadas

$$p'_{j} = (x - g_{j})p'_{j-1} + p_{j-1} - h_{j}p'_{j-2}$$

(A1.2)

De acordo com a condição de normalização tem-se:

$$p_0 = 1$$
 , $p'_0 = 0$

(A1.3)

sendo que, para iniciar a iteração, se tem:

$$p_{-1} = p'_{-1} = qualquer valor.$$

Seguindo o algoritmo proposto por Villadsen e Michelsen (1978), para calcular as raízes de $\rho_n(x)$

procede-se do seguinte modo:

(a)Começando-se com x=0 e usando-se o método de Newton

$$x_{1}^{(i+1)} = x_{1}^{(i)} - \frac{p_{n}\left(x_{1}^{(i)}\right)}{p_{n}'\left(x_{1}^{(i)}\right)}$$

(A1.4)

obtém-se a raiz x_1 mais próxima de 0.

(b)Para calcular a segunda raiz x_2 , parte-se do polinômio quociente

$$q_{n-1}(x) = \frac{p_n(x)}{x - x_1}$$

(A1.5)

ou:

$$\ln q_{n-1}(x) = \ln p_n(x) - \ln (x - x_1)$$

ou:

$$\frac{d}{dx}\ln q_{n-1}(x) = \frac{d}{dx}\ln p_n(x) - \frac{d}{dx}\ln(x-x_1)$$

,

ou:

$$\frac{q'_{n-1}(x)}{q_n(x)} = \frac{p'_n(x)}{p_n(x)} - \frac{1}{(x-x_1)}$$

(A1.6)

Então tem-se:

$$x_{2}^{(i+1)} = x_{2}^{(i)} - \frac{q_{n-1}(x_{2}^{(i)})}{q_{n-1}'(x_{2}^{(i)})}$$

,

ou, atendendo-se a (A1.6):

$$x_{2}^{(i+1)} = x_{2}^{(i)} - \frac{\left[p_{n}(x_{2}^{(i)})/p_{n}'(x_{2}^{(i)})\right]}{1 - \left\{p_{n}(x_{2}^{(i)})/\left[p_{n}'(x_{2}^{(i)})(x_{2}^{(i)}-x_{1})\right]\right\}}$$

(A1.7)

A iteração (A1.7) é iniciada com um valor

$$x_2^{(1)} = x_1 + \varepsilon$$
, com, por ex. $\varepsilon \approx 10^{-4}$

(c) Da mesma maneira, uma vez calculadas as raizes $x_1, x_2, ..., x_k$ (k < n), para calcular a raiz x_{k+1} tem- se

$$x_{k+1}^{(i+1)} = x_{k+1}^{(i)} - \frac{p_n(x_{k+1}^{(i)}) / p'_n(x_{k+1}^{(i)})}{1 - \left[p_n(x_{k+1}^{(i)}) / p'_n(x_{k+1}^{(i)}) \right] \left\{ \sum_{j=1}^{k} \left[\frac{1}{(x_{k+1}^{(i)} - x_j)} \right] \right\}}$$

(A1.8)

Como se vê, o cálculo das raízes de $p_n(x)$ pressupõe cálculos sucessivos de $p_n(x)$ e de $p'_n(x)$ para diferentes valores de x, usando as relações de recorrência (A1.1) e (A1.2).

ANEXO II
Programa Principal

- C C C # PROGRAMA PARA A SIMULACAO DE UMA COLUNA DE DESTILAÇÃO
- MULTICOMPONENTE UTILIZANDO O METODO DA COLOCACAO #
- # ORTOGONAL MISTA (METODO DE NEWTON / MA28)
- Ċ C #
- # PACOTE TERMODINAMICO: PENG-ROBINSON
- С Cedido pelo Prof.Dr.Roger Zemp
- Ĉ # Autor: Alexandre Rodrigues Simões
- С

IMPLICIT NONE

INTEGER I, J, NK, NST, NCOMP, ICOND, ISL, II, K, NF, ICOMP(10), NPCSE, NPCSR, \$NPSE,NPSR,NXE,IS,NPF,NTPC,NXR,IPOL,IRN1,ICN1

REAL FEED(5),FL(50),FV(50),FLL(500,15), \$FVV(500,15),TK(500),ZF(10),YF(10),XF(10),EMU(50),DFE(15,15), \$DFR(15,15),DBE(15,15),DBR(15,15),X(500,15),Y(500,15),ROOTE(15), \$ROOTR(15),DIFE(15),DIFR(15),XINTPE(15),XINTPR(15),XX(500,15), \$YY(500,15),TTC(500),Texac(500),xexact(2,175,2),DEST,RFLX,P, \$TF,TT,TB,SL,PBAR,TTK,TBK,EMUS,FT,VAPF,LIQF,HVFLASH,HLFLASH,HF,BU, \$SLL,SVV,XY,XV,YV,TV,ERRLIQ,ERRVAP,ERRT,TFK CHARACTER*1 PHASE COMMON/COMP/NCOMP COMMON/ORTODAT/NPCSE,NPCSR,NTPC,NPF COMMON/DIF/DFE,DFR,DBE,DBR COMMON/COLDAT/NK_PBAR_FEED,RFLX,DEST,HF,BU,EMU,SL COMMON/DHAHN/ROOTE,ROOTR,DIFE,DIFR COMMON/MA/IRN1.ICN1 C C ******* **** Ĉ C * NOMENCLATURA × ******* ***** C C C * NCOMP => No. de componentes ź * ICOMP => No. de identificacao do componente no banco de c c dados TERMORPP.PPD * NST => No. de pratos teoricos da coluna * С * NF => No. do prato de alimentacao c c * ICOND => Tipo do condensador. ICOND = 1 (parcial) ICOND = 2 (total)C * ISL => Se ISL = 1 (existe saida lateral de liquido no Ċ condensador parcial) С DEST => Vazao de destilado (moles/h) С * RFLX => Razao de refluxo С * P => Pressao (atm) С * TT => Estimativa da Temperatura do condensador (oC) * TB => Estimativa da Temperatura do refervedor (oC) С С * SL => Vazao da saida lateral no condensador (moles/h) * NPCSE => No. de ptos. de colocacao da seccao de esgotamento * C С * NPCSR => No. de ptos. de colocacao da seccao de retificacao * С * NPSE => No. de pratos da seccao de esgotamento ¢ * NPSR => No. de pratos da seccao de retificacao С * NPF => No. do pto. de colocação referente a alimentação С * NTPC => No. total de ptos. de colocação * * С * EMUS => Eficiencia de Murphree С * TF => Temperatura da alimentacao (oC) C C * FEED(i) => Vazao de alimentacao do componente i (mol/h) * ****** С OPEN(UNIT=6, FILE = 'multred.in', STATUS = 'OLD') OPEN(UNIT=7, FILE = 'COLOC1.IN', STATUS = 'OLD') OPEN(UNIT=9, FILE = 'EXIT.OUT', STATUS = 'UNKNOWN') С READ(7,*) IPOL READ(7,*) IRN1,ICN1 С LEITURA DO NUMERO DE COMPONENTES C Ç READ(6,*) NCOMP NK = NCOMP

```
С
С
   LEITURA DOS COMPONENTES
С
        DO I = 1,NK
         READ(6,*) ICOMP(I)
        END DO
С
С
   CHAMADA DA SUBROTINA DE INICIALIZAÇÃO
С
        CALL PP_INIT(ICOMP,'TERMORPP.PPD',2,NK)
С
С
   LEITURA DA ESPECIFICACOES DA COLUNA
С
        READ(6,*) NST,NF,ICOND,ISL
        READ(6,*) DEST,RFLX,P,TT,TB
        IF(ISL.EQ.1) THEN
         READ(6,*) SL
        ELSE
        SL = 0.D0
        END IF
С
        PBAR = P * 1.013
        TTK = TT+273.15
        TBK = TB+273.15
C
С
   LEITURA DOS DADOS DA COLOCACAO ORTOGONAL
С
        READ(7,*) NPCSE,NPCSR
        READ(7,*) NPSE,NPSR
С
        NPF = NPCSE+4
        NTPC = NPCSE+NPCSR+7
С
   LEITURA DA EFICIENCIA DE MURPHREE
С
С
        READ(6,*) EMUS
        DOI = 2, NTPC
        EMU(I) = EMUS
        END DO
        EMU(1) = 1
        IF (ICOND.EQ.2) EMU(NTPC)=1E-08
С
С
   LEITURA DAS ESPECIFICACOES DA ALIMENTACAO
С
        READ(6,*) TF,(FEED(I),I=1,NK)
С
        TFK = TF+273.15
С
        FT = 0
        DO J = 1,NK
        FT = FT + FEED(J)
        END DO
С
        DOJ = 1, NK
        ZF(J) = FEED(J)/FT
        END DO
С
С
   CHAMAR A SUBROTINA PARA CALCULAR A CONDICAO DA ALIMENTACAO
С
        CALL PP_FLASH(ZF,PBAR,TFK,LIQF,VAPF,HLFLASH,HVFLASH,XF,YF,PHASE)
С
С
   PRIMEIRA APROXIMACAO DOS FLUXOS TOTAIS
С
        FV(NTPC) = DEST
        FL(NTPC) = DEST*RFLX
        FV(NTPC-1) = FL(NTPC)+DEST+SL
        DO II = 3,NTPC
         I = NTPC+2-II
         FL(I) = FL(I+1)
         IF(I.EQ.NPF) FL(I)=FL(I)+FT*LIQF
         FV(I-1) = FV(I)
         IF (I.EQ.NPF) FV(I-1)=FV(I-1)-FT*VAPF
```

```
END DO
        FL(1) = FL(2) - FV(1)
        BU = FT - DEST - SL
С
        WRITE(9,*)'IMPRESSAO DOS RESULTADOS'
        WRITE(9,*)
        WRITE(9,*)
        WRITE(9,18)
 18 FORMAT(/,'COMPOSICAO DE ALIMENTACAO ',//)
        WRITE(9,23) (J,ZF(J),J=1,NK)
 23 FORMAT(I3,F12.3)
С
        WRITE(9,16) NST,NF,RFLX,P
 16 FORMAT(//,'NUMERO DE PRATOS ',19/,
  $NUMERO DO PRATO DE ALIMENTACAO ',19/,
  $'RAZAO DE REFLUXO ',F6.3/,
  $'PRESSAO DE OPERACAO (atm) ',F5.2//)
С
        WRITE(9,*) FV(NTPC),FL(NTPC),FVV,FLL
   ESTIMATIVA INICIAL DO PERFIL DE TEMPERATURA (LINEAR)
С
С
        DO I = 1,NTPC
         TK(I) = TBK+(I-1)*(TTK-TBK)/NTPC
        END DO
С
С
   ESTIMATIVA INICIAL DAS VAZOES MOLARES (VALOR FIXO)
Ċ
        DO I = 1,NTPC
         DOJ = 1,NK
         FLL(I,J) = FEED(J)/FT*FL(I)
         FVV(I,J) = FEED(J)/FT*FV(I)
         END DO
        END DO
С
С
   CHAMAR A SUBROTINA DE COLOCACAO ORTOGONAL
C
        CALL COL_ORTO(NPSE,NPSR,NPCSE,NPCSR,DFE,DFR,DBE,DBR,IPOL)
С
   CHAMAR A SUBROTINA PARA CALCULAR O SISTEMA DE EQUACOES
C
Ċ
        CALL CALC(FLL,FVV,TK)
С
   CALCULAR AS FRACOES MOLARES
С
С
        DO I = 1,NTPC
         SLL = 0
         SVV = 0
         DO J = 1,NK
         SLL = SLL + FLL(I,J)
         SVV = SVV + FVV(I,J)
         END DO
         FL(I) = SLL
         FV(I) = SVV
         END DO
С
         DOI = 1.NTPC
         DOJ = 1,NK
         X(I,J) = FLL(I,J)/FL(I)
         Y(I,J) = FVV(I,J)/FV(I)
         END DO
         END DO
С
    IMPRESSAO DOS RESULTADOS NOS PONTOS DE COLOCACAO
С
С
         WRITE(9,*)
         WRITE(9,*) ' SOLUCAO NOS PONTOS DE COLOCACAO '
         WRITE(9,*) '-
         WRITE(9,*)
         WRITE(9,*)
         WRITE(9,*) ' Composicao da fase liquida '
С
         WRITE(9,*)
         DOI = 1,NTPC
         WRITE(9,*) I,(X(I,J),J=1,NK)
```

<u>Anexo II</u>

```
END DO
С
         WRITE(9,*)
         WRITE(9,*) ' Composicao da fase vapor '
         WRITE(9,*)
С
         DO I = 1.NTPC
         WRITE(9,*) I,(Y(I,J),J=1,NK)
         END DO
С
         WRITE(9,*)
         WRITE(9,*) 'Perfil de Temperatura ( oC )'
         WRITE(9,*)
С
         DO I = 1,NTPC
         WRITE(9,*) I,(TK(I)-273.15)
         END DO
С
         DOJ = 1,NK
         XX(1,J) = X(1,J)
         YY(1,J) = Y(1,J)
         END DO
         TTC(1) = TK(1)-273.15
С
c
c
   CALCULAR OS RESULTADOS PRATO A PRATO UTILIZANDO INTERPOLAÇÃO
         NXE = NPCSE+2
         DO IS = 1,NPSE
         XY = (IS-1)/1
         CALL INTPR(15,NXE,XY,ROOTE,DIFE,XINTPE)
         DOI = 1,NK
          XV = 0
          YV = 0
          \overline{T}V = 0
          DO J = 1,NXE
          XV = XV + XINTPE(J) * X(J+1,I)
          YV = YV + XINTPE(J) * Y(J+1,I)
          TV = TV+XINTPE(J)*TK(J+1)
          END DO
          XX(IS+1,I) = XV
          YY(IS+1,I) = YV
          TTC(IS+1) = TV-273.15
         END DO
         END DO
С
         DOJ = 1.NK
         XX(NF,J) = X(NPF,J)
         YY(NF,J) = Y(NPF,J)
         END DO
         TTC(NF) = TK(NPF)-273.15
С
         NXR = NPCSR+2
         DO IS = 1,NPSR
         XY = (IS-1)/1
         CALL INTPR(15,NXR,XY,ROOTR,DIFR,XINTPR)
         DO I = 1,NK
          XV = 0
          YV = 0
          TV = 0
          DO J = 1,NXR
          XV = XV + XINTPR(J) * X(J + NPF, I)
          YV = YV + XINTPR(J) * Y(J + NPF,I)
          TV = TV+XINTPR(J)*TK(J+NPF)
          END DO
          XX(NF+IS,I) = XV
          YY(NF+IS,I) = YV
          TTC(NF+IS) = TV-273.15
         END DO
         END DO
С
         DOJ = 1,NK
         XX(NST,J) = X(NTPC,J)
         YY(NST,J) = Y(NTPC,J)
```

```
END DO
           TTC(NST) = TK(NTPC)-273.15
С
č
     IMPRESAO DOS RESULTADOS PRATO A PRATO
           WRITE(9,*)
           WRITE(9,*)' SOLUCAO PRATO-A-PRATO '
           WRITE(9,*)'---
           WRITE(9,*)
          WRITE(9,*) ' Composicao da fase liquida '
           WRITE(9,*)
С
          DOI=1,NST
           WRITE(9,*) I,(XX(I,J),J=1,NK)
          END DO
С
          WRITE(9,*)
          WRITE(9,*) ' Composicao da fase vapor '
          WRITE(9,*)
С
          DOI = 1.NST
           WRITE(9,*) I,(YY(I,J),J=1,NK)
          END DO
С
          WRITE(9,*)
          WRITE(9,*) 'Perfil de Temperatura ( oC )'
          WRITE(9,*)
С
          DO I = 1,NST
           WRITE(9,*) I,TTC(I)
          END DO
          WRITE(9,*) FV(NTPC),FL(NTPC),FVV,FLL
С
C***** Transferencía dos resultados para os seus arquivos ********** C
          open (unit=91, file='resx1ma.dat', status='unknown')
          open (unit=92, file='resx2ma.dat', status='unknown')
          open (unit=93, file='resy1ma.dat', status='unknown')
          open (unit=94, file='resy2ma.dat', status='unknown')
          open (unit=95, file='restema.dat', status='unknown')
С
          do i=1.nst
           write (91,*) xx(i,1)
           write (92,*) xx(i,2)
          write (92,*) xx(i,2)
write (93,*) yy(i,1)
write (94,*) yy(i,2)
write (95,*) TTC(i)
          end do
С
          close (unit=91)
          close (unit=92)
          close (unit=93)
          close (unit=94)
          close (unit=95)
C
          errliq=0.0
          errvap=0.0
          errt=0.0
          open (unit=11, file = 'multexac.in', status = 'old')
          do k=1,2
           do i=1.nst
           do j=1,nk
            read(11,*) xexact(k,i,j)
            if (k.eq.1) then
            errliq=errliq+(xexact(k,i,j)-xx(i,j))**2
            else
            errvap=errvap+(xexact(k,i,j)-yy(i,j))**2
            end if
           end do
           end do
```

UNICAMP BLIOTECA CENTRA

```
end do
        do i=1.nst
         read(11,*) Texac(i)
         errt=errt+(Texac(i)-TTC(i))**2
        end do
С
        write(9,*)'erro quadratico na fase liquida=',errlíq
        write(9,*)'erro quadratico na fase vapor=',errvap
        write(9,*)'erro quadratico da temperatura=',errt
С
        close(unit=11)
С
   CALCULO DAS CARGAS TERMICAS DO REFERVEDOR E CONDENSADOR
C
С
   CALL ENT(NST,TK,PBAR,FLL,FVV,HLL,HVV,HL,HV,CL,CV)
c
   QC=HV(NST-1)-HL(NST)-HV(NST)
¢
   QR=HV(1)+HL(1)-HL(2)
с
с
   WRITE(6,725) QR
   WRITE(6,726) QC
с
c 725 FORMAT(//,' O CALOR FORNECIDO AO REFERVEDOR E',E12.4;KJ/H',/)
c 726 FORMAT(//,' O CALOR RETIRADO NO CONDENSADOR E',E12.4,'KJ/H',/)
с
        STOP
        END
   C
С
   ┼┼╎╏╡╪╕┼╏┤╏╞╕┑╞╞┽┝╞┥╡╞┤╏┑╎╏╡╏┼┇┽╗╇╗╄┑┝┱┿┿╋╤╋┿╋╪╋┱╋╪╋╗┝╴
С
             SUBROTINA COL_ORTO
   ÷
                                               +
   + Calcula as diferencas forward e backward para as seccoes de +
С
С
   +
             esgotamento e retificacao
                                          -
С
             ┿┿<del>┇╴┇╺┇╍┋╸┋╸╡╸╡╶┨╶┋╶╬╺┋╍╡╍╪╍┋┅╛╼┨┍╋╺┧╍┋╼┇╸┇╸┨╺┨</del>
                                         <u>╺╆╶╆╍╆┉╪┉╪┉╪┉╪╌╪╶╪╴╪╶┧╴╊╶╂╶╉╌┿╍╪╍╉┈╋┈╊┉╋╍╄┉╋╌╄┉╉╴┾╌┨╌┧┈╀╍</u>┿
с
        SUBROUTINE COL_ORTO(NPSE,NPSR,NPCSE,NPCSR,DFE,DFR,DBE,DBR,IPOL)
        REAL XNE(2),XNR(2),DIFE(15),DIFR(15),ROOTE(15),ROOTR(15),
  $DFE(15,15),DFR(15,15),DBE(15,15),DBR(15,15)
        INTEGER I, J, NTE, NTR, IPOL
        COMMON/DHAHN/ROOTE,ROOTR,DIFE,DIFR
c
С
   CALCULO DOS PTOS. DE COLOCACAO PARA A SECCAO DE ESGOTAMENTO
с
        XNE(1) = 0.0
        XNE(2) = (NPSE-1)/1.0
        IF (IPOL.EO.1) THEN
         CALL HAHN(15,NPSE,NPCSE,1,1,1.0,1.0,DIFE,ROOTE,XNE)
        ELSE IF (IPOL.EQ.2) THEN
         CALL ORTPOL(NPSE,NPCSE,1,1,DIFE,ROOTE,XNE)
        ELSE
        STOP
        END IF
С
   CALCULO DOS PTOS. DE COLOCACAO PARA A SECCAO DE RETIFICACAO
C
        XNR(1) = 0.0
        XNR(2) = (NPSR-1)/1.0
        IF (IPOL.EQ.1) THEN
        CALL HAHN(15,NPSR,NPCSR,1,1,1.0,1.0,DIFR,ROOTR,XNR)
        ELSE IF (IPOL.EQ.2) THEN
        CALL ORTPOL(NPSR,NPCSR,1,1,DIFR,ROOTR,XNR)
        ELSE
         STOP
        END IF
С
C
C
   CALCULO DAS DIFERENCAS FORWARD E BACKWARD PARA A SECCAO
           DE ESGOTAMENTO
С
        NTE = NPCSE+2
         DO I = 1,NTE
         DO J = 1,NTE
          XX = ROOTE(J)
          CALL DILFOR(15,NTE,I,DIFE,ROOTE,XX,VCE)
```

DFE(I,J) = VCE

```
CALL DILBAC(15,NTE,I,DIFE,ROOTE,XX,VBE)
          DBE(I,J) = VBE
          END DO
         END DO
С
C
C
C
   CALCULO DAS DIFERENCAS FORWARD E BACKWARD PARA A SECCAO
           DE RETIFICACAO
        NTR = NPCSR+2
        DO I = 1,NTR
         DO J = 1,NTR
         XY = ROOTR(J)
          CALL DILFOR(15,NTR,I,DIFR,ROOTR,XY,VCR)
         DFR(I,J) = VCR
          CALL DILBAC(15,NTR,I,DIFR,ROOTR,XY,VBR)
         DBR(I,J) = VBR
         END DO
         END DO
с
        RETURN
        END
С
C
C
C
              ╾╬╌╉╶┧╶╅╍╫╌╏╴╕╶╡╶╎╴╕╴┇╴╪╍╬┅┫┅╬╴╡╌╬┅╉╴╅┅╬┥╬┉╋╌╬╍╋╼╣┝┥╡┝╢╌╗╴╅╴┥┥┥╢╴╢╴╋╴╋╴╋╸╋╸┨╴┨╴╉╴┨╴╋╴┨╴╋
   agungung
              SUBROTINA CALC
                                             +
   4
   Ĉ
        SUBROUTINE CALC(FLL, FVV, TK)
        REAL FLL(500,15),FVV(500,15),X(200),
  $FEED(5),TK(500),EMU(50)
         INTEGER N.IT.IFLAG
         COMMON/ORTODAT/NPCSE,NPCSR,NTPC,NPF
         COMMON/COLDAT/NK_PBAR_FEED,RFLX,DEST,HF,BU,EMU,SL
¢
        N = NTPC^{*}(2^{*}NK^{+1})
¢
         it=1
         iflag=1
¢
   transformacao das variaveis
c
с
         DOJ = 1,NK
         DOI = 1, NTPC
         X((J-1)*NTPC+I) = FLL(I,J)
          X((J-1)*NTPC+NK*NTPC+NTPC+I) = FVV(I,J)
         END DO
         END DO
с
         DO I=1,NTPC
         X(NK*NTPC+I) = TK(I)
         END DO
с
   chamar a subrotina newton
с
         T1 = HIGH_RES_CLOCK@(.TRUE.)
с
         call newton(N,X,iflag,it)
         T2 = HIGH_RES_CLOCK@(.FALSE.)
¢
c
   impresao do tempo de cpu e outras informacoes
¢
с
         it=it-1
         tcpu = T2-T1
ç
С
         write(6,*) 'Tempo de CPU(seg) = ',tcpu
         write(9,*)'numero de iteracoes=',it
         write(9,*)'iflag=',iflag
         if (iflag.lt.0) then
         write(9,*)'no success'
         end if
c
   transformacao das variaveis
С
с
         DOJ = 1,NK
         DO1 = 1,NTPC
          FLL(I,J) = X((J-1)*NTPC+I)
```

```
FVV(I,J) = X((J-1)*NTPC+NK*NTPC+NTPC+I)
         END DO
         END DO
¢
         DOI=1,NTPC
         TK(I) = X(NK*NTPC+I)
         END DO
с
         RETURN
         END
c
С
    ╉╋╉┼┲┿╄┋╃┲┿╪┼╋┾╪┼╇┾╪┿╋╆┧╄╉┼╉╤╪╪╞╞╕┽╎┟╞╒┾┿╓╒┝╕╉╕╋╕┽╏╄╪┾╡┿╓╎╎╕╕┟╄╕╕
Č
            FIM DA SUBROTINA CALC
    4
                                                  +
С
    с
   <del>╡╋┇╗┝╗┿┉╗┿╕╘╘╞╏╏╎┇┇┇┇╡╡┥╡┿┋┿┋┢╹╕┝╽╹╡╏╞┇┇┇┇┇┢┇╘┇╘╻</del>╋╦╇<del>╝┇┇┇┇┇</del>
¢
              SUBROTINA NEWTON
   +
с
                                               +
с
   с
         subroutine newton(n,xv,iflag,it)
         COMMON/MA/IRN1,ICN1
         REAL xv(n),delta_xv(200)
с
         do while(it.le.19)
¢
       calcular delta_xv, resolvendo o problema linear Ax=b_
С
    sendo A=fjac, (rhs=)b=fvec e x=delta_xv
с
с
         call ma28(n,xv,delta_xv,iflag)
с
       calcular o passo newton_
с
c
         do i=1,n
          xv(i)=xv(i)-delta_xv(i)
         end do
с
c
       aumentar o numero de iteracoes_
¢
         it=it+1
с
         end do
¢
         return
         end
с
               <del>╡┇┪┇┧┇╎╏╔┇┥┇╎┇┇┇┇╎┇╏┇┇╡╎┇┇┥┇┇╏┇╏╏┇╏╎╡┥╿┇╇</del>╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋
с
             FIM DA SUBROTINA NEWTON
                                                     ÷
с
        ╃╅╅┾┼┟╅╆┿╋╪╅┽╛┟╗┿╞┼╅┾╁╅┾╁╅┼╆┼╆┼╆┼┼┾╛┽┼╡┥╘╛╓┼╡╢┟╗╆╄╬╋╋╋╞╞┊╄╢┦╶╛╆┾╋
¢
   +++
   <del>┥╞┢┪┢╗┫╪╝┥┚╏╏╞╗╗╎┇╪╪╪┥╞┥┇╔┇╖╝╚┇╹┊╔╞┇╏╏╏╏╏╏╏╏╏╏╏╏╏╏╏╏╏╏╏╏╏╏╹┊╹┊╹┊╹╵┊╹╵╵╵╵╵</del>
с
              SUBROTINA MA28
                                              +
c
   -
   <u>╅╞┊┥╪┼┿╕╪╪╹┊╷╓┿╄╔╬┶╔╅┼┾┾╞┊╎┾╓┆╡┤┾┊┤╎╞┊╎┼┽┾┾╋╋╋╋╋╋╋╋╋╋╋╋</u>╋╋╋<mark>╞</mark>╞╟┉
c
с
         subroutine ma28(n,xv,rhs,iflag)
         INTEGER MTYPE, LICN, LIRN, IRN(IRN1), ICN(ICN1), IKEEP(IRN1,5),
  $IW(IRN1,8)
         REAL xv(n),a(ICN1),rhs(n),w(IRN1),U
         common/coloc/ne1,ne2,nr1,nr2
         COMMON/MA/IRN1,ICN1
С
c
      declaracao das variaveis, sendo:___
С
   mtype integer, para resolver a equacao direta (mtype=1)
с
¢
       ou a su matrix transposta (mtype.ne.1)
   licn integer, tamanho dos arrays A e icn
с
   lirn integer, tamanho do array irn
с
       real, para controlar se "bias towards numeric or
с
   B
       sparsity pivoting. u=1.0 gives partial pivoting
¢
       while u=0.0 does not check multipliers at all."
c
       \{u > 1 \rightarrow u = 1, u < 0 \rightarrow u = 0\}
с
С
   a(2*nz) vetor com os valores nao-zeros do Jacobiano
c
   irn(nz) vetro com as posicoes "row" dos elementos
С
        nao-zeros na matriz Jacobiana
с
```

```
с
   icn(2*nz) vetor com as posicoes "column" dos elementos
       nao-zeros da matriz Jacobiano
c
   rhs(n) vetor com os valores fvec das equacoes ao entrar
с
       em ma28 e com o resultado delta xv na saida
¢
   ikeep(nz,5)
с
¢
   iw(nz,8)
¢
   w(nz)
С
с
        nz=IRN1
        nzd=2*nz
        mtype=1
        licn=nzd
        lim=nz
        u=1.0
с
     calcular o vetor da funcao e a matrix do jacobiano
C
\mathbf{c}
        call fcn(n,xv,rhs,a,nz,nzd,irn,icn)
с
     Decomposicao LU da matríx A(n*n,1)____
¢
c
        call ma28ad(n,nz,a,licn,irn,lirn,icn,u,ikeep,iw,w,iflag)
¢
     Resolver o problema linear Ax=b
¢
¢
  com rhs=b na entrada da subrotina e x=rhs na saida
¢
        call ma28cd(n,a,licn,icn,ikeep,rhs,w,mtype)
с
        return
        end
¢
¢
   c
  +
           FIM DA SUBROTINA MA28
                                            +
  с
с
  ┽╴╴╪╴╪╴╋╋┿╅╗╪╪╪╴╪╺╪╪┝╧╋┾┝╝╪╖┙╪╵┊┊┿╪╪╋╪╪╋╪╪╋╪╄┨╘┨┿╋╋╓╖┿┿╪╫╖╖┥╎┊┊╖╋┿┿╋╋╋╋╋╋╋╋╋
С
             SUBROTINA FCN
¢
                                       +
  +
   c
с
        SUBROUTINE FCN(N,X,FVEC,aa,nze,nzed,irn,icn)
        REAL X(N),FVEC(N),FLL(500,15),FVV(500,15),XFK(500,15),
  $DFE(15,15),DFR(15,15),DBE(15,15),DBR(15,15),FEED(5),EMU(50),
  $TK(500),DFK(500,15,15),FL(50),FV(50),
  $HLL(500,15),HVV(500,15),HL(500),HV(500),CL(500),CV(500),aa(nzed),
  $$1,$2,$3,$4,$5,$6,$7,$8,$9,$10
        INTEGER IRN(IRN1),ICN(ICN1)
        COMMON/ORTODAT/NPCSE, NPCSR, NTPC, NPF
        COMMON/DIF/DFE,DFR,DBE,DBR
        COMMON/COLDAT/NK,PBAR,FEED,RFLX,DEST,HF,BU,EMU,SL
        COMMON/MA/IRN1.ICN1
С
¢
        DO J = 1,NK.
        DO I = 1,NTPC
        FLL(I,J) = X((J-1)*NTPC+I)
        FVV(I,J) = X((J-1)*NTPC+NK*NTPC+NTPC+I)
        END DO
        END DO
ç
        DO I = 1,NTPC
        TK(I) = X(NK*NTPC+I)
        END DO
с
        DO I = 1,NTPC
        SLL=0
        SVV=0
        DOJ = 1,NK
        SLL = SLL + FLL(I,J)
        SVV = SVV + FVV(IJ)
        END DO
```

```
FL(1) = SLL
        FV(I) = SVV
       END DO
с
с
       CALL KFAC(NTPC, TK, PBAR, FLL, FVV, XFK, DFK)
       CALL ENT(NTPC,TK,PBAR,FLL,FVV,HLL,HVV,HL,HV,CL,CV)
с
С
   FUNCOES DISCREPANCIA DE BALANCO DE MASSA
С
Ĉ
\mathbf{C}
   REFERVEDOR
с
       DOJ = 1.NK
        FVEC(J) = FLL(2,J)-FLL(1,J)-FVV(1,J)
       END DO
с
Ċ
   PRIMEIRO PTO. EXTREMO DA SECCAO DE ESGOTAMENTO
с
       DO J = 1,NK
        S1 = 0
        DO II = 1,NPCSE+2
        S1 = S1 + DFE(II, 1) + FLL(II+1, J)
        END DO
        FVEC(NK+J) = S1+FVV(1,J)-FVV(2,J)
        KTT = NK+J
       END DO
c
C
   PTOS. INTERNOS DA SECCAO DE ESGOTAMENTO
С
       NN = 0
       DO K = 2,(NPCSE+1)
        DO J = 1, NK
        S2 = 0
        S3 = 0
        DO JJ = 1,NPCSE+2
         S2 = S2 + DFE(JJ,K) + FLL(JJ+1,J)
         S3 = S3 + DBE(JJ,K) * FVV(JJ+1,J)
        END DO
        NN = NN+1
        FVEC(2*NK+NN) = S2-S3
        END DO
       END DO
       KK = NN+2*NK
¢
С
   SEGUNDO PTO. EXTREMO DA SECCAO DE ESGOTAMENTO
¢
       DOJ = 1.NK
        S4 = 0
        DO II = 1,NPCSE+2
        S4 = S4+DBE(II,NPCSE+2)*FVV(II+1,J)
        END DO
        FVEC(KK+J) = FLL(NPF,J)-FLL(NPF-1,J)-S4
        IJ = KK+J
       END DO
с
   PRATO DE ALIMENTACAO
С
¢
       DOJ = 1.NK
        FVEC(IJ+J) = FLL(NPF+1,J)+FVV(NPF-1,J)-FLL(NPF,J)-FVV(NPF,J)
  $
        +FEED(J)
        JI = IJ+J
       END DO
С
   PRIMEIRO PTO. EXTREMO DA SECCAO DE RETIFICACAO
С
C
       DO J = 1,NK
        S5 = 0
        DO II = 1,NPCSR+2
        S5 = S5+DFR(II,1)*FLL(NPF+II,J)
        END DO
```

```
FVEC(JI+J) = S5+FVV(NPF,J)-FVV(NPF+1,J)
         KW = JI + J
        END DO
c
С
   PTOS. INTERNOS DA SECCAO DE RETIFICACAO
с
        NW = 0
        DO K = 2,(NPCSR+1)
        DO J = 1.NK
         S6 = 0
         S7 = 0
         DO JJ = 1,NPCSR+2
          S6 = S6+DFR(JJ,K)*FLL(JJ+NPF,J)
         S7 = S7+DBR(JJ,K)*FVV(JJ+NPF,J)
         END DO
         NW = NW+1
         FVEC(NW+KW) = S6-S7
         END DO
        END DO
        KH = NW + KW
c
C
   SEGUNDO PTO. EXTREMO DA SECCAO DE RETIFICACAO
с
        DO J = 1,NK
        S8 = 0
        DO JJ = 1,NPCSR+2
         S8 = S8+DBR(JJ,NPCSR+2)*FVV(JJ+NPF,J)
         END DO
        FVEC(KH+J) = FLL(NTPC,J)-FLL(NTPC-1,J)-S8
        KX = KH+J
        END DO
¢
С
   CONDENSADOR PARCIAL
c
        DOJ = 1,NK
        FVEC(KX+J) = FVV(NTPC-1,J)-FVV(NTPC,J)-FLL(NTPC,J)*(1.+SL/FL(N
  $
         TPC))
         KS = KX+J
        END DO
   FUNCOES DISCREPANCIA DE RELACAO DE EQUILIBRIO
С
С
        KA = 0
        DO I = 1,NTPC
        DO J = 1,NK
         KA=KA+1
         FVEC(KA+KS) = EMU(I)*XFK(I,J)*FV(I)*FLL(I,J)/FL(I)-FVV(I,J)
         IF(I.GT.1) FVEC(KA+KS)=FVEC(KA+KS)+(1.-EMU(I))*FVV(I-1,J)*
  $
         FV(I)/FV(I-1)
         END DO
        END DO
С
č
c
   FUNCOES DISCREPANCIA DE BALANCO DE ENERGIA
C
   REFERVEDOR
        S1 = 0
        DOJ = 1,NK
        S1 = S1 + FLL(1,J)
        END DO
с
        FVEC(2*NK*NTPC+1) = BU-S1
¢
   10 PTO, EXTREMO DA SECCAO DE ESGOTAMENTO
С
¢
        S2 = 0
        DO I = 1,NPCSE+2
        S2 = S2 + DFE(I,1) + HL(I+1)
        END DO
с
        FVEC(2*NK*NTPC+2) = S2+HV(1)-HV(2)
с
C
   PTOS. INTERNOS DA SECCAO DE ESGOTAMENTO
с
```

```
DO J = 2,NPCSE+1
        S3 = 0
        S4 = 0
        DO I = 1,NPCSE+2
         S3 = S3 + DFE(I,J) + HL(I+1)
         S4 = S4 + DBE(I,J) + HV(I+1)
        END DO
        FVEC(2*NK*NTPC+1+J) = S3-S4
        END DO
с
С
   20 PTO. EXTREMO DA SECCAO DE ESGOTAMENTO
c
        S5 = 0
        DO I = 1,NPCSE+2
        S5 = S5+DBE(I,NPCSE+2)*HV(I+1)
        END DO
с
        FVEC(2*NK*NTPC+NPF-1) = HL(NPF)-HL(NPF-1)-S5
с
С
   PRATO DE ALIMENTACAO
¢
        FVEC(2*NK*NTPC+NPF) = HL(NPF+1)+HV(NPF-1)-HL(NPF)-HV(NPF)+HF
¢
С
   10 PTO. EXTREMO DA SECCAO DE RETIFICACAO
с
        S6 = 0
        DOI=1,NPCSR+2
        S6 = S6+DFR(I,1)+HL(NPF+I)
        END DO
с
        FVEC(2*NK*NTPC+NPF+1) = S6+HV(NPF)-HV(NPF+1)
с
С
   PTOS. INTERNOS DA SECCAO DE RETIFICACAO
с
        DO J = 2,NPCSR+1
        S7 = 0
        S8 = 0
        DO I = 1,NPCSR+2
         S7 = S7 + DFR(LJ) + HL(NPF+I)
         S8 = S8+DBR(I,J)*HV(NPF+I)
        END DO
        FVEC(2*NK*NTPC+NPF+J) = S7-S8
        END DO
с
С
   20 PTO, EXTREMO DA SECCAO DE RETIFICACAO
с
        S9 = 0
        DO1 = 1,NPCSR+2
        S9 = S9+DBR(I,NPCSR+2)*HV(NPF+I)
        END DO
с
        FVEC(2*NK*NTPC+NTPC-1) = HL(NTPC)-HL(NTPC-1)-S9
¢
С
   CONDENSADOR
c
        S10 = 0
        DOJ = 1.NK
        S10 = S10 + FLL(NTPC,J)
        END DO
с
        FVEC(NTPC*(2*NK+1)) = DEST*RFLX-S10
с
с
С
   JACOBIANO DAS EQUACOES DO REFERVEDOR
¢
        k=1
        NX = 0
        DOJ = 1.NK
        aa(k) = -1
        im(k) = J
        icn(k) = (J-1)*NTPC+1
        aa(k+1) = 1
```

```
irn(k+1) = J
          icn(k+1) = (J-1)*NTPC+2
          aa(k+2) = -1
          irn(k+2) = J
          icn(k+2) = NTPC*(NK+(J-1))+NTPC+1
         k=k+3
         NX = NX+1
         END DO
c
C
   JACOBIANO DO 1º PTO. EXTREMO DA SECCAO DE ESGOTAMENTO
с
         DOl = 1, NK
         DO J = 1,NPCSE+2
          aa(k) = DFE(J,1)
          irn(k) = NX+I
          icn(k) = J+(I-1)*NTPC+1
          k=k+1
         END DO
         aa(k) = 1
         irn(k) = NX+I
         icn(k) = NTPC*(NK+(I-1))+NTPC+1
         aa(k+1) = -1
         irn(k+1) = NX+I
         icn(k+1) = NTPC*(NK+(I-1))+NTPC+2
         k=k+2
         NW = NX + I
         END DO
С
   JACOBIANO DOS PTOS. INTERNOS DA SECCAO DE ESGOTAMENTO
С
с
         NZ = 0
         DO L = 2,NPCSE+1
         DOI = 1.NK
          NZ = NZ+1
          DO J = 1.NPCSE+2
          aa(k) = DFE(J,L)
          irn(k) = NZ+NW
          icn(k) = J+(I-1)*NTPC+1
          aa(k+1) = -DBE(J,L)
          irn(k+1) = NZ+NW
          icn(k+1) = J+NTPC*(NK+(I-1))+NTPC+1
          k=k+2
          END DO
         END DO
         END DO
         NY = NZ+NW
с
   JACOBIANO DO 2º PTO. EXTREMO DA SECCAO DE ESGOTAMENTO
C
¢
         DOI = 1, NK
         aa(k) = -1
         irn(k) = NY+I
         icn(k) = (I-1)*NTPC+NPF-1
         aa(k+1) = 1
         im(k+1) = NY+I
         icn(k+1) = (I-1)*NTPC+NPF
         k=k+2
         DO J = 1,NPCSE+2
          aa(k) = -DBE(J,NPCSE+2)
          irn(k) = NY+I
          icn(k) = J+NTPC*(NK+(I-1))+NTPC+1
          k=k+1
         END DO
         NG = NY+I
         END DO
¢
Ċ
   JACOBIANO DO PRATO DE ALIMENTACAO
с
         DOI = 1, NK
         aa(k) = -1
         irm(k) = NG+I
icm(k) = (I-1)*NTPC+NPF
          aa(k+1) = 1
```

<u>Anexo II</u>

```
irn(k+1) = NG+I
         icn(k+1) = (I-1)*NTPC+NPF+1
         aa(k+2) = 1
         im(k+2) = NG+1
         icn(k+2) = NTPC*((I-1)+NK)+NTPC+NPF-1
         aa(k+3) = -1
         irn(k+3) = NG+I
         icn(k+3) = NTPC*((I-1)+NK)+NTPC+NPF
         k≕k+4
         NU = NG+I
         END DO
с
С
   JACOBIANO DO 10 PTO. EXTREMO DA SECCAO DE RETIFICAÇÃO
с
         DO I = 1,NK
         DO J = 1,NPCSR+2
         aa(k) = DFR(J,1)
          irn(k) = I + NU
         icn(k) = (I-1)*NTPC+NPF+J
         k≕k+1
         END DO
         aa(k) = 1
         im(k) = I + NU
         icn(k) = NTPC*((I-1)+NK)+NTPC+NPF
         aa(k+1) = -1
         irn(k+1) = I+NU
         icn(k+1) = NTPC*((I-1)+NK)+NTPC+NPF+1
         k=k+2
         NH = I + NU
         END DO
с
   JACOBIANO DOS PTOS. INTERNOS DA SECCAO DE RETIFICACAO
С
с
         NV = 0
         DO L = 2,NPCSR+1
         DOI = 1,NK
         NV = NV+1
         DO J = 1,NPCSR+2
          aa(k) = DFR(J,L)
          irn(k) = NV + NH
          icn(k) = J+(I-1)*NTPC+NPF
          aa(k+1) = -DBR(J,L)
          irn(k+1) = NV+NH
          icn(k+1) = J+NTPC*((I-1)+NK)+NTPC+NPF
          k=k+2
         END DO
         END DO
         END DO
        NT = NV+NH
с
С
   JACOBIANO DO 2º PTO, EXTREMO DA SECCAO DE RETIFICAÇÃO
с
         DO I = 1,NK
         aa(k) = -1
         im(k) = NT+I
         icn(k) = NTPC^{*}((I-1)+1)-1
         aa(k+1) = 1
         im(k+1) = NT+I
         icn(k+1) = NTPC*((I-1)+1)
         k≕k+2
         DOJ = 1.NPCSR+2
         aa(k) = -DBR(J,NPCSR+2)
         im(k) = NT+I
         icn(k) = J+NTPC*((I-1)+NK)+NTPC+NPF
         k=k+1
         END DO
         NQ = NT+I
         END DO
¢
С
   JACOBIANO DO CONDENSADOR
с
         DO 11 = 1,NK
         DOJ = 1,NK
```

¢

```
DOL = 1,NTPC
           IF(L.EQ.NTPC) THEN
           aa(k) = SL*FLL(NTPC,II)/FL(NTPC)**2
            IF (J.EQ.II) THEN
                   aa(k) = aa(k) - (1.+SL/FL(NTPC))
            END IF
            irn(k) = NQ+II
            icn(k) = L+(J-1)*NTPC
           k≕k+]
           END IF
           END DO
          END DO
          END DO
с
         DOI=1,NK
          aa(k) = 1
          im(k) = NQ+I
          icn(k) = NTPC*((I-1)+NK+1)+NTPC-1
          aa(k+1) = -1
          irn(k+1) = NQ+I
          icn(k+1) = NTPC*((I-1)+NK+1)+NTPC
          k=k+2
          NR = NQ+I
          END DO
С
С
   JACOBIANO DAS RELACOES DE EQUILIBRIO
С
         DOI=1,NTPC
          DO II = 1,NK
          DO J = 1,NK
           DO L = 1,NTPC
           IF(L.EQ.I) THEN
            IF(J.EQ.II) THEN
                   aa(k) = FV(I)*EMU(I)/FL(I)*(XFK(I,II)+FLL(I,II)*
  $
           DFK(I,II,II)-XFK(I,II)*FLL(I,II)/FL(I))
                   im(k) = NK*(NTPC+(I-1))+II
                   icn(k) = L+(J-1)*NTPC
                   aa(k+1) = EMU(I)*FLL(I,II)/FL(I)*(XFK(I,II)+FV(I)*
  $
            DFK(1,II,J+NK+1))-1
                   im(k+1) = NK*(NTPC+(I-1))+II
                   icn(k+1) = L+NTPC*((J-1)+NK)+NTPC
                   IF(LGT.1) THEN
                   aa(k+1) = aa(k+1)+(1-EMU(I))*FVV(I-1,II)/FV(I-1)
                   END IF
            ELSE
                   aa(k) = FV(I)*EMU(I)/FL(I)*(FLL(I,II)*DFK(I,II,II)-
  $
           XFK(I,II)*FLL(I,II)/FL(I))
                   irn(k) = NK*(NTPC+(I-1))+II
                   icn(k) = L+(J-1)*NTPC
                   aa(k+1) = EMU(I)*FLL(I,II)/FL(I)*(XFK(I,II)+FV(I)*
  $
            DFK(I,II,J+NK+1))
                   irn(k+1) = NK*(NTPC+(I-1))+II
                   icn(k+1) = L+NTPC*((J-1)+NK)+NTPC
                   IF(I.GT.1) THEN
                   aa(k+1) = aa(k+1)+(1-EMU(I))*FVV(I-1,II)/FV(I-1)
                   END IF
            END IF
            k=k+2
           END IF
С
           IF(L,LT,I) THEN
            IF(L.EQ.(I-1)) THEN
                   IF(J.EQ.II) THEN
                    aa(k) = (1-EMU(I))*FV(I)/FV(I-1)
                    irn(k) = NK*(NTPC+(I-1))+II
                    icn(k) = L+NTPC*((J-1)+NK)+NTPC
                    IF(I.GT.1) THEN
                    aa(k) = aa(k)*(1-FVV(l-1,II)/FV(l-1))
                    END IF
                   ELSE
                    aa(k) = (1-EMU(I))*FV(I)/FV(I-1)
                    irn(k) = NK*(NTPC+(I-1))+II
```

```
icn(k) = L+NTPC*((J-1)+NK)+NTPC
                   IF(LGT.1) THEN
                   aa(k) = aa(k)*(-FVV(I-1,II)/FV(I-1))
                   END IF
                  END IF
                  k=k+1
           END IF
           END IF
с
           END DO
          END DO
          END DO
         END DO
Ċ
         DO I = 1,NTPC
         DO II = 1,NK
          DOL = 1,NTPC
           IF(L.EQ.I) THEN
           aa(k) = EMU(I)*FV(I)*FLL(I,II)/FL(I)*DFK(I,II,NK+1)
           irn(k) = NK*(NTPC+(I-1))+II
           icn(k) = NK*NTPC+L
           k=k+1
          END IF
          END DO
         END DO
         END DO
С
С
    JACOBIANO DAS EQUACOES DE BALANCO DE ENERGIA
C
    EM RELACAO A VAZAO MOLAR DE LIQUIDO
С
c
С
    REFERVEDOR
с
         DOJ = 1,NK
         aa(k) = -1
         irn(k) = 2*NK*NTPC+1
         icn(k) = (J-1)*NTPC+1
         k=k+1
         END DO
С
С
    1º PTO. EXTREMO DA SECCAO DE ESGOTAMENTO
¢
         DOI = 2, (NPCSE+3)
         DO J = 1,NK
         aa(k) = DFE(I-1,1)*HLL(I,J)
         im(k) = 2*NK*NTPC+2
         icn(k) = (J-1)*NTPC+I
         k=k+1
         END DO
         END DO
c
С
   PTOS. INTERNOS DA SECCAO DE ESGOTAMENTO
с
         DOI = 2, NPCSE+1
         DO J = 1,NK
         DOL = 2, (NPCSE+3)
          aa(k) = DFE(L-1,I)*HLL(L,J)
          irn(k) = 2*NK*NTPC+1+I
          icn(k) = (J-1)*NTPC+L
          k≔k+1
         END DO
         END DO
        END DO
¢
С
   20 PTO. EXTREMO DA SECCAO DE ESGOTAMENTO
с
        DOJ = 1,NK
         aa(k) = -HLL(NPF-1,J)
         irn(k)=2*NK*NTPC+NPF-1
         icn(k) = (J-1)*NTPC+NPF-1
         aa(k+1) = HLL(NPF,J)
         irn(k+1) = 2*NK*NTPC+NPF-1
         icn(k+1) = (J-1)*NTPC+NPF
```

120

```
k=k+2
         END DO
с
С
    PRATO DE ALIMENTACAO
с
         DO J = 1,NK
         aa(k) = -HLL(NPF,J)
          irn(k) = 2*NK*NTPC+NPF
          icn(k) = (J-1)*NTPC+NPF
          aa(k+1) = HLL(NPF+1,J)
          irn(k+1) = 2*NK*NTPC+NPF
         icn(k+1) = (J-1)*NTPC+NPF+1
         k=k+2
         END DO
с
Ĉ
    10 PTO. EXTREMO DA SECCAO DE RETIFICACAO
с
         DO I = 2,(NPCSR+3)
         DOJ = 1,NK
          aa(k) = DFR(I-1,1)*HLL(NPF+I-1,J)
          im(k)=2*NTPC*NK+NPF+1
          icn(k) = (J-1)*NTPC+NPF-1+I
          k=k+1
         END DO
         END DO
¢
С
   PTOS. INTERNOS DA SECCAO DE RETIFICACAO
с
         DOI = 2, (NPCSR+1)
         DO J = 1,NK
          DOL = 2,(NPCSR+3)
          aa(k) = DFR(L-1,I)*HLL(NPF-1+L,J)
          irn(k) = 2*NK*NTPC+NPF+I
          icn(k) = (J-1)*NTPC+NPF-1+L
          k=k+1
         END DO
         END DO
         END DO
с
Ċ
   20 PTO. EXTREMO DA SECCAO DE RETIFICACAO
с
         DOJ = 1,NK
         aa(k) = -HLL(NTPC-1,J)
         irn(k) = NTPC^{*}(2^{*}NK+1)-1
         icn(k) = NTPC^*((J-1)+1)-1
         aa(k+1) = HLL(NTPC,J)
         irn(k+1) = NTPC*(2*NK+1)-1
         icn(k+1) = NTPC*((J-1)+1)
         k==k+2
         END DO
с
С
   CONDENSADOR
¢
         DOJ = 1,NK
         aa(k) = -1
         irn(k) = NTPC*(2*NK+1)
         icn(k) = NTPC^{*}((J-1)+1)
         k=k+1
         END DO
c
С
   EM RELACAO A TEMPERATURA
¢
С
   REFERVEDOR
¢
   (jacobiano = zero)
¢
С
   10 PTO. EXTREMO DA SECCAO DE ESGOTAMENTO
¢
         aa(k) = CV(1)
         im(k) = 2*NK*NTPC+2
         icn(k) = NK*NTPC+1
         aa(k+1) = DFE(1,1)*CL(2)-CV(2)
         irn(k+1) = 2*NK*NTPC+2
         icn(k+1) = NK*NTPC+2
```

```
k=k+2
с
         DO1 = 3,(NPCSE+3)
         aa(k) = DFE((1-1), 1)*CL(1)
         irn(k) = 2*NK*NTPC+2
         icn(k) = NK*NTPC+I
         k=k+1
         END DO
с
С
   PTOS. INTERNOS DA SECCAO DE ESGOTAMENTO
¢
         DOI = 2, (NPCSE+1)
         DO J = 2,(NPCSE+3)
         aa(k) = DFE(J-1,I)*CL(J)-DBE(J-1,I)*CV(J)
         irn(k) = 2*NK*NTPC+1+I
          icn(k) = NK*NTPC+J
         k=k+1
         END DO
         END DO
с
С
   20 PTO. EXTREMO DA SECCAO DE ESGOTAMENTO
c
         DOI = 2, (NPCSE+3)
         aa(k) = -DBE(I-1,NPCSE+2)*CV(I)
         im(k) = 2*NK*NTPC+NPCSE+3
         icn(k) = NK*NTPC+I
         IF(I.EQ.(NPCSE+3)) THEN
         aa(k) = aa(k) - CL(NPF-1)
END IF
         k≕k+1
        END DO
c
        aa(k) = CL(NPF)
        irn(k) = 2*NK*NTPC+NPCSE+3
        icn(k) = NK*NTPC+NPF
        k=k+1
c
С
   PRATO DE ALIMENTACAO
с
         aa(k) = CV(NPF-1)
         irn(k) = 2*NK*NTPC+NPF
         icn(k) = NK*NTPC+NPF-1
         aa(k+1) = -CL(NPF)-CV(NPF)
         irn(k+1) = 2*NK*NTPC+NPF
         icn(k+1) = NK*NTPC+NPF
         aa(k+2) = CL(NPF+1)
         im(k+2) = 2*NK*NTPC+NPF
         icn(k+2) = NK*NTPC+NPF+1
         k=k+3
с
С
   10 PTO. EXTREMO DA SECCAO DE RETIFICACAO
с
         aa(k) = CV(NPF)
         irn(k) = 2*NK*NTPC+NPF+1
         icn(k) = NK*NTPC+NPF
         k=k+1
с
        DOJ = 1,NPCSR+2
         aa(k) = DFR(J,1)*CL(NPF+J)
         im(k) = 2*NK*NTPC+NPF+1
         icn(k) = NK*NTPC+NPF+J
         IF(J.ÉQ.1) THEN
         aa(k) = aa(k) - CV(NPF+1)
         END IF
         k≕k+1
        END DO
с
С
   PTOS. INTERNOS DA SECCAO DE RETIFICACAO
¢
        DO I = NPF+2,NTPC-2
         DO J = 1,NPCSR+2
         aa(k) = DFR(J,I-NPF)*CL(NPF+J)-DBR(J,I-NPF)*CV(NPF+J)
         im(k) = 2*NK*NTPC+I
```

```
icn(k) = NK*NTPC+NPF+J
         k≕k+1
         END DO
         END DO
¢
С
   20 PTO, EXTREMO DA SECCAO DE RETIFICACAO
с
         DOJ = 1,NPCSR+2
         aa(k) = -DBR(J,NPCSR+2)*CV(NPF+J)
         irn(k) = 2*NK*NTPC+NTPC-1
         icn(k) = NK*NTPC+NPF+J
         IF(J.EQ.(NPCSR+2)) THEN
         aa(k) = aa(k) - CL(NTPC-1)
         END IF
         k=k+1
         END DO
с
         aa(k) = CL(NTPC)
        irn(k) = NTPC*(2*NK+1)-1
         icn(k) = NK*NTPC+NTPC
        k=k+1
с
С
   CONDENSADOR
   (Jacobiano = zero)
с
с
С
   EM RELACAO A VAZAO MOLAR DE VAPOR
С
   REFERVEDOR
С
   (Jacobiano = zero)
С
с
   10 PTO. EXTREMO DA SECCAO DE ESGOTAMENTO
С
¢
         DOJ = 1,NK
         aa(k) = HVV(1,J)
         im(k) = 2*NK*NTPC+2
         icn(k) = NTPC^{*}((J-1)+NK+1)+1
         aa(k+1) = -HVV(2,J)
         irn(k+1) = 2*NK*NTPC+2
         icn(k+1) = NTPC*((J-1)+NK+1)+2
         k=k+2
         END DO
¢
С
   PTOS. INTERNOS DA SECCAO DE ESGOTAMENTO
с
         DOI = 2, (NPCSE+1)
         DOJ = 1.NK
         DOL = 2, (NPCSE+3)
          aa(k) = -DBE(L-1,I)*HVV(L,J)
          irm(k) = 2*NK*NTPC+1+I
          ien(k) = NTPC*((J-1)+NK+1)+L
          k=k+1
         END DO
         END DO
         END DO
¢
   20 PTO. EXTREMO DA SECCAO DE ESGOTAMENTO
С
с
         DO1 = 2, (NPCSE+3)
         DOJ = 1, NK
          aa(k) = -DBE(I-1,NPCSE+2)*HVV(I,J)
          irn(k) = 2*NK*NTPC+NPF-1
          icn(k) = NTPC^*((J-1)+NK+1)+I
          k=k+1
         END DO
         END DO
с
C
    PRATO DE ALIMENTACAO
с
         DO J = 1,NK
         aa(k) = HVV(NPF-1,J)
         im(k) = 2*NK*NTPC+NPF
         icn(k) = NTPC*((J-1)+NK+1)+NPF-1
          aa(k+1) = -HVV(NPF,J)
```

```
irn(k+1) = 2*NK*NTPC+NPF
          icn(k+1) = NTPC*((J-1)+NK+1)+NPF
          k=k+2
         END DO
c
С
    10 PTO. EXTREMO DA SECCAO DE RETIFICAÇÃO
¢
         DO J = 1,NK
         aa(k) = HVV(NPF,J)
          irn(k) = 2*NK*NTPC+NPF+1
          icn(k) = NTPC*((J-1)+NK+1)+NPF
          aa(k+1) = -HVV(NPF+1,J)
          im(k+1) = 2*NK*NTPC+NPF+1
          icn(k+1) = NTPC*((J-1)+NK+1)+NPF+1
         k=k+2
         END DO
с
С
   PTOS. INTERNOS DA SECCAO DE RETIFICAÇÃO
с
         DOI = 2,(NPCSR+1)
         DOJ = 1.NK
          DO L = 2,(NPCSR+3)
          aa(k) = -DBR(L-1,I)*HVV(NPF-1+L,J)
          irn(k) = 2*NK*NTPC+NPF+I
          icn(k) = NTPC*((J-1)+NK+1)+NPF-1+L
          k=k+1
          END DO
         END DO
         END DO
¢
С
   20 PTO. EXTREMO DA SECCAO DE RETIFICAÇÃO
с
         DOI = 2, (NPCSR+3)
         DO J = 1,NK
          aa(k) = -DBR(I-1,NPCSR+2)*HVV(NPF-1+I,J)
          im(k) = NTPC^{*}(2^{*}NK+1)-1
          icn(k) = NTPC*((J-1)+NK+1)+NPF-1+I
          k=k+1
         END DO
         END DO
c
С
   CONDENSADOR
   (Jacobiano = zero)
С
С
         RETURN
         END
с
с
           <del>╶┥┩╛╞╒┾┩╪┼┇╞┇┇╪╪╞╞╞┇┇╎┇╎┊┼┊╞┋┼┇╛╏╡┥┽╛╗┦╡┇╎┥╎┆╕┼┇╏╡╻╞┇╏╏┊┇╿</del>
   +++++
с
              SUBROTINA HAHN
   +
   + Calcula os pontos de colocacao( raizes do polinomio de Hahn ) +
¢
с
   +
        e a derivada primeira nestes mesmos pontos.
                                                   4
   с
с
         subroutine hahn(nd,np,n,n0,n1,al,be,dif1,root,xne)
         real dif1(nd),dif2(10),root(nd),xne(2)
¢
c evaluation of roots of hahn polynomials and of derivatives of inter
c polating polynomial
c
c evaluation of coefficients in recursion formulas. These coefficients
c are stored transitorily in dif1 and dif2
¢
         znp=np
         ab=al+be
         ad=be-al
         ap=al*be
         dif1(1)=(ad/(ab+2.)+1.)*(znp-1.)/2.
         dif2(1)=0.
         if(n.lt.2) goto 15
         do 10 i=2.n
         zi=i-1
```

```
z=ab+2.*zi
           difl(i)=((znp-1.)*ab*(be+1.)-zi*ab*ad-zi*(zi+1.)*ad+2.*zi*(ab+
   j zi+1.)*(znp-1.))/z/(z+2.)
с
           if(i.ne.2) goto 11
           dif2(i)=(znp-1.)*(ab+ap+zi)*(znp+z-1.)/z/z/(z+1.)
           goto 10
  ll z=z*z
           y=zi*(ab+zi)
           y=y*(ap+y)*(znp-zi)*(ab+znp+zi)
           dif2(i)=y/z/(z-1.)
  10 continue
с
c root determination by newton method
¢
  15 x≕0.
           do 20 i=1,n
 25 xd=0.
           xn=1.
           xd1=0.
          xn1=0.
           do 30 j=1,n
           xp=(dif1(j)-x)*xn-dif2(j)*xd
          xp1=(dif1(j)-x)*xn1-dif2(j)*xd1-xn
          xd=xn
          xd1=xn1
          xn=xp
 30 xn1=xp1
          zc=1
          z=xn/xn1
          if(i.eq.1) goto 21
          do 22 j=2,i
 22 zc=zc-z/(x-root(j-1))
 21 z=z/zc
          X=x-z
          if(abs(z).gt.1e-05) goto 25
          root(i)=x
          x=x+.0001
 20 continue
с
c add eventual interpolation points at the extremes
¢
          nt=n+n0+n1
          if(n0.eq.0) goto 35
          do 31 i=1,n
          j=n+1-i
 31 root(j+1)=root(j)
          root(1)=xne(1)
 35 if(n1.eq.1) root(nt)=xne(2)
с
c evaluate first derivatives of polynomial to be stored in dif1
¢
          do 40 i=1,nt
          x=root(i)
          difl(i)=1.
          do 40 j=1,nt
          if (j.eq.i) goto 40
          y=x-root(j)
          dif1(i)=y*dif1(i)
 40 continue
          return
          end
   ╺┿┲╃┿╦┽┼╎╞╪╃╪┅╬╤╅┪╕┨┿╪┿┾┫╋╌┿┫┨┨╋┿╪╅┼╝╧┼┾┾╎╧┼┼┟╋╪┼╏┨┨╏┇╪┿╟╝┩
¢
               FIM DA SUBROTINA HAHN
¢
   +
                                                       +
          <del>╡╎╎┇┇┍┉╋╪╒┉╽╏┊┇╍┢┥┇┇┇╎╖┥╒</del>┾<del>╞┥╡╡┥╡╵╸</del>
    +++
¢
¢
с
   SUBROTINA DILFOR
   +
                                                   +
¢
           Calcula as primeiras diferencas forward
                                                     +
с
   +
   <del>╶╡┥┇┋┿╋╋┪╋╡┟╎┊<sup>┿</sup>╡╞╎╞┿╹╹╡╪┝┊╞╵╞╘╘╘┿╡</del><del>┛╹╡╽┇┥╡╏╏┊╡┇┇┇┇╗╗╒╗╗</del>┾<del>┇</del>
¢
¢
          subroutine dilfor(nd,nt,i,dif1,root,x,vect)
```

125

```
REAL dif1(nd),root(nd)
с
c subroutine evaluates first forward differences of polynomial li at
c all nodes, which are stored in vect
с
       v1=1.
       v2=1.
       do 20 k=1,nt
       if(k.eq.i) goto 20
       vi=vi*(x-root(k)+1.)
       v2=v2*(x-root(k))
 20 continue
       vect=(v1-v2)/dif1(i)
       return
       end
¢
  +
            SUBROTINA DILBAC
с
                                     4
¢
  +
        Calcula as primeiras diferencas backward
                                     +
  ¢
       subroutine dilbac(nd,nt,i,dif1,root,x,vb)
       REAL dif1(nd),root(nd)
с
c subroutine evaluates first backward differences of polynomials
c li at the collocation nodes
с
       v1=1.
       v2=1.
       do 20 k=1,nt
       if(k.eq.i) goto 20
       v1=v1*(x-root(k))
       v2=v2*(x-root(k)-1.)
 20 continue
с
       vb=(v1-v2)/dif1(i)
       return
       end
Ċ
с
  <del>╺╪╡╪╧╗┿╪┲┿╪╋┿╪╪┼┊┼┊┼┊╡╡╡╡╎┊╡╎╡╎╡╎╡╞╞</del>╞╧╋<del>╘╞╕╞╛╋╪┿╇╚╓┇╕┥╄╋╘╎┆╡╡┇┇┇</del>╞
         FIM DA SUBROTINA DILBAC
                                        -
  ÷
с
  ++++++
       c
  ┿┶┽┼╉┡┚┼╡╊╅╊╓╝╎╏╎╏╡┝┡┲╞╅┝╊╊╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋╋
¢
  +
         SUBROTINA ENT
                                  +
с
¢
  ¢
  Subroutine ENT(NST, T, P, FLL, FVV, HLL, HVV, HL, HV, CL, CV)
  REAL HLL(500,15),HVV(500,15),FLL(500,15),FVV(500,15),T(500),
```

\$HL(500), HV(500),CL(500), CV(500), XX(10), YY(10), \$FUGCL(10), FUGCV(10),HVAPX(10), HLIQX(10),ZCOMP(10),P, \$HLIQP,HVAPP,SLIQ,SVAP,HLIQ,HVAP,XT,YT,FLIQ,FVAP,TP,ISMIX INTEGER NCOMP,NK,I,NST,J,II

```
COMMON /COMP/ NCOMP
```

NK = NCOMP

 $\begin{array}{l} \text{Do 70 I = 1, NST} \\ \text{HL(I) = 0.0} \\ \text{HV(I) = 0.0} \\ \text{CV(I) = 0.0} \\ \text{CV(I) = 0.0} \\ \text{XT = 0.0} \\ \text{YT = 0.0} \\ \text{FLIQ = 0.0} \\ \text{FVAP = 0.0} \end{array}$

C Compute compositions

с

С

с

с

с

¢ с

¢

с с

¢

```
Anexo II
    Do 10 J = 1, NK
     YT = YT + FVV(I, J)
     XT = XT + FLL(I, J)
     FLIQ = FLIQ + FLL(I, J)
     FVAP = FVAP + FVV(I, J)
 10 Continue
    Do 20 J = 1, NK
     YY(J) = FVV(I, J) / YT
     XX(J) = FLL(I, J) / XT
 20 Continue
    TAUX = T(I)
    Call PP_PT(XX, P, TAUX, L', HLIQ, SLIQ, FUGCL, ZCOMP)
    Call PP_PT(YY, P, TAUX, V, HVAP, SVAP, FUGCV, ZCOMP)
С
    Total enthalpy of phases
С
    HL(I) = HLIQ * FLIQ
    HV(I) = HVAP * FVAP
С
    Enthalpy of components
С
    Do 30 J = 1, NK
    HLL(I, J) = HLIQ
     HVV(I, J) = HVAP
 30 Continue
    write(*,*) hliq
С
    Derivative of total enthalpy
    TAUX = T(I) + 1.0
    Call PP_PT(XX, P, TAUX,'L', HLIQP, SLIQ, FUGCL, ZCOMP)
    Call PP_PT(YY, P, TAUX, V', HVAPP, SVAP, FUGCV, ZCOMP)
    CL(I) = ((HLIQP - HLIQ) * FLIQ) / 1.0
    CV(I) = ((HVAPP - HVAP) * FVAP) / 1.0
 70 Continue
  Return
   End
   ╶<del>╔┍╞┥╞╎╞╎╕╔╒┥╗╔┿╓╡╎╞╎╞╎┥╓┥╞╔┥╿┙╛╎╎╞╢╔╞╔╓╵╽╵╞┥╞┊╞┊╹┥┾┿╔╔╓╡╎╏╠╞┨╞┊╞╞</del>
              FIM DA SUBROTINA ENT
   -
                                               +
   ┈╪╦╦┨┎╘╘╏┧┧╄╪┼┿╪╗╬╢┼╧┼┼╁┿╪╢╢╟┊╪╁╪┤╞┠┨┾╉╪╢╛┟┨┟┨┟┨┼┼┧╄┼┽╬╦┿╬╆╎╛╞┨┊
          +++++
               SUBROTINA KFAC
   +
                                             +
       ╍╬<del>╔┥┲╕┆┇┊┇┊┇┊╗╔╗╓╗╬╗╎┇╎╞┇╡╞╅╎╢╬┇╘┨┥┊╿╏┥┇╲</del>╁╄┽╄╋╋╋╋╋╋╋╸╋╋╋╋╋╋╋╋╋
   -<del>f-+</del>
   Subroutine KFAC(NST, T, P,FLL,FVV, XKF, DFAC)
С
С
С
   This subroutine calculates the separation coefficient XKF
С
    and the partial derivatives of XKF with respect to component
Ċ
    flow in the vapour phase, liquid phase and temperature.
С
С
    10/08/94
               Modified for PR
                                   RJZ
С
С
С
```

С Parameter (NMSTAG = 60, NMCOMP = 10)

REAL XX(10), YY(10),FLL(500,15),FVV(500,15), \$XKF(500,15),XXP(10), FUGCL(10), FUGCLP(10), FUGCLT(10), \$YYP(10),FUGCV(10),FUGCVT(10),ZCOMPL(10), \$ZCOMPV(10),ZCOMPLP(10),ZCOMPVP(10),ZCOMPLT(10), \$ZCOMPVT(10),DFAC(500,15,15),T(500),XT,YT,HLIQ,HVAP,SLIQ,SVAP, \$TP,P INTEGER NK, NK1, I, NST, J, K,NCOMP COMMON /COMP/ NCOMP

NK = NCOMP

C CALCULATION OF THE SEPARATION FACTOR C _____

NKI = NK + 1

C LOOP OVER ALL STAGES

DO 110 I = 1, NST XT = 0.0 YT = 0.0

C Compute compositions

DO 10 J = 1, NK YT = YT + FVV(I, J)XT = XT + FLL(I, J)10 CONTINUE DO 20 J = 1, NK YY(J) = FVV(I, J) / YTXX(J) = FLL(I, J) / XT20 CONTINUE

C Compute fugacity coefficients

```
TAUX = T(I)
```

C

Call PP_PT(XX, P, TAUX,'L', HLIQ, SLIQ, FUGCL,ZCOMPL) Call PP_PT(YY, P, TAUX,'V', HVAP, SVAP, FUGCV,ZCOMPV) DO 30 J = 1, NK XKF(I, J) = FUGCL(J) / FUGCV(J) 30 CONTINUE

Call PP_PT(XXP,P, TAUX,'L', HLIQ, SLIQ, FUGCLP,ZCOMPLP)

DO 50 J = 1, NK DFAC(I, J, K) = FUGCLP(J) / FUGCV(J) * - XKF(I, J) 50 CONTINUE

60 CONTINUE

С

Derivative of separation factor relative to component flow in

vapour phase

С

```
C
    Do 90 K = 1, NK
     Do 70 J = 1, NK
      YYP(J) = FVV(I, J) / (YT + 1.0)
 70
     Continue
     YYP(K) = (FVV(I, K) + 1.0) / (YT + 1.0)
     TAUX = T(I)
     Call PP_PT(YYP, P, TAUX,'V', HVAP, SVAP, FUGCVP, ZCOMPVP)
     Do 80 J == 1, NK.
      DFAC(I, J, K + 1 + NK) = FUGCL(J) / FUGCVP(J)
                  - XKF(I, J)
  *
 80
     Continue
 90 Continue
C
    CALL GET_KEY@(KEY)
С
    Derivative of separation factor relative to temperature
С
    TAUX = T(I) + 1.0E1
    Call PP_PT(XX, P, TAUX,'L', HLIQ, SLIQ, FUGCLT, ZCOMPLT)
   Call PP_PT(YY, P, TAUX, V', HVAP, SVAP, FUGCVT, ZCOMPVT)
    Do 100 J = 1, NK
    DFAC(I, J, NK + 1) = (FUGCLT(J) / FUGCVT(J)
               - XKF(I, J)) / 1.0E1
 100 Continue
 110 Continue
   Return
   End
с
   SUBROTINA INTPR
   +
                                             +
с
  + Calcula os coeficientes da interpolação Lagrangeana +
Ċ
   ┉╦╦╪┼╗╼┉╗╌┙┾╲╈╔┉╬╪╪╪╞╞╞╎╛╡╎┊┥╡╞╡╊┅┽╅┤╞┨┟╞╞╞╡┇╡╞╎┇╎╏╞╞╞╡╋╡╏╕┥┾
с
с
         subroutine intpr(nd,nt,x,root,dif1,xintp)
         REAL root(nd),dif1(nd),xintp(nd)
¢
         pol = 1.
         do 5 i = 1, nt
         y = x - root(i)
         xintp(i) = 0.
         if (y.eq.0) xintp(i) = 1.
 5 pol = pol*y
         if (pol.eq.0) go to 10
         do 6 i = 1, nt
 6 \operatorname{xintp}(i) = \operatorname{pol/difl}(i)/(x-\operatorname{root}(i))
 10 return
         end
с
   ╺<del>╔╶┇╘┥┉╡╔┥┍┝╪╪┽┥╏┥┥╇╔╔┇╞┪┥╞╠╞╕┥╋╞┋╪╡╿╎┝╞╋┥╔╎╞╪╎╞╵╞╘╡╵┝╞</del>╶
С
              FIM DA SUBROTINA INTPR
                                                   ÷
с
   +
¢
   с
c subrotina ORTPOL
Ċ
c calculo dos zeros do polinomio ortogonal discreto definido pelo
c produto interno (p,q)=soma(p(x(i))*q(x(i))*w(i),i=1,...,np) e das
c derivadas do polínomio interpolador.
c sao tambem calculdos oa coeficientes da formula de recursao, trans-
c feridos para o programa principal atraves dos vetores beta e gama.
Ċ
```

```
subroutine ortpol(np,n,n0,n1,dif1,root,xne)
   IMPLICIT REAL(A-H,O-Z)
   REAL dif1(15),dif2(15),root(15),w(150),p1(150),p2(150),
   $
          p3(150),xne(2),y(150),beta(150),gama(150)
¢
c calculo dos coeficientes da formula de recursao. Estes coeficientes
c sao transitoriamente armazenados em dif1 e dif2
с
    znp=np-1
   do 20 i=1,np
   y(i)≕i-l
   w(i)=y(i)*(znp-y(i))
w(i)=1.
с
20 continue
   do 100 j=1,n
   h=0.
   b≕0.
   if(j.gt.1) goto 35
   do 10 i=1,np
 10 h=h+w(i)
   do 25 i=1,np
25 b=b+y(i)*w(i)
с
   dif1(1)=b/h
   dif2(1)=0.
   beta(1)=dif1(1)
   gama(1)=dif2(1)
С
   hl≕h
   do 30 i=1,np
   pl(i)=1.
30 p2(i)=y(i)-difl(j)
   if(n.gt.1) goto 100
   goto 100
с
35 do 40 i=1,np
40 h=h+w(i)*p2(i)**2
¢
   do 50 i=1,np
50 b=b+y(i)*w(i)*p2(i)**2
   dif1(j)=b/h
   dif2(j)=h/h1
   beta(j)=dif1(j)
   gama(j)=dif2(j)
   hl=h
¢
   do 60 i=1,np
60 p3(i)=(y(i)-dif1(j))*p2(i)-dif2(j)*p1(i)
   if(n.gt.2.and.j.lt.n) goto 65
с
65 do 70 i=1,np
  p1(i)=p2(i)
70 p2(i)=p3(i)
100 continue
с
c determinacao dos zeros pelo metodo de Newton
с
200 x=0.
   do 300 i=1,n
225 xd=0.
   xn≕l.
   xd1=0.
   xn1==0.
   do 250 j=1,n
   xp=(x-dif1(j))*xn-dif2(j)*xd
   xpl=(x-difl(j))*xnl-dif2(j)*xdl+xn
   xd=xn
   xd1=xn1
   xn=xp
250 xn1=xp1
¢
   zc=1.0
   z=xn/xn l
```

```
if(i.eq.1) goto 270
do 260 j=2,i
260 zc=zc-z/(x-root(j-1))
270 z=z/zc
    x≕x-z
    if(abs(z).gt.1E-4) goto 225
    root(i)=x
x=x+1
300 continue
с
c adicionar os eventuais pontos de interpolação nos extremos
с
    nt=n+n0+n1
    if(n0.eq.0) goto 350
do 310 i=1,n
j=n+1-i
310 root(j+1)=root(j)
root(1)=xne(1)
350 if(n1.eq.1) root(nt)=xne(2)
с
c calculo das primeiras derivadas do polínomio a armazenar em difl
с
    do 400 i=1,nt
    x=root(i)
dif1(i)=1.
    do 400 j=1,nt
    if(j.eq.i) goto 400
y1=x-root(j)
dif1(i)=y1*dif1(i)
400 continue
    return
    end
```

<u>Anexo II</u>

Sub-rotina MA28

c######date 01 jan 1984 copyright ukaea, harwell. c#####alias ma28ad ma28bd ma28cd c###### calls ma30 mc20 mc22 mc23 mc24 subroutine ma28ad(n, nz, a, licn, irn, lirn, icn, u, ikeep, iw, w, * iflag) c this subroutine performs the lu factorization of a c the parameters are as follows c n order of matrix not altered by subroutine. c nz number of non-zeros in input matrix not altered by subroutine. c a is a real array length licn, holds non-zeros of matrix on entry c and non-zeros of factors on exit. reordered by mc20a/ad and c mc23a/ad and altered by ma30a/ad. c licn integer length of arrays a and icn. not altered by subroutine. c irn integer array of length lirn. holds row indices on input. c used as workspace by ma30a/ad to hold column orientation of c matrix c lim integer length of array irn. not altered by the subroutine. c icn integer array of length licn. holds column indices on entry c and column indices of decomposed matrix on exit, reordered by c mc20a/ad and mc23a/ad and altered by ma30a/ad. c u real variable set by user to control bias towards numeric or c sparsity pivoting. u=1.0 gives partial pivoting while u=0, does c not check multipliers at all. values of u greater than one are treated as one while negative values are treated as zero. not С c altered by subroutine. c ikeep integer array of length 5*n used as workspace by ma28a/ad (see later comments). it is not required to be set on entry с с and, on exit, it contains information about the decomposition. it should be preserved between this call and subsequent calls с to ma28b/bd or ma28c/cd. с ikeep(i,1),i=1,n holds the total length of the part of row i c in the diagonal block. с row ikeep(i,2),i=1,n of the input matrix is the ith row in с с pivot order. column ikeep(i,3), i=1,n of the input matrix is the ith column с in pivot order. с с ikeep(i,4),i=1,n holds the length of the part of row i in the l part of the l/u decomposition. с ikeep(i,5),i=1,n holds the length of the part of row i in the С off-diagonal blocks. if there is only one diagonal block, с ikeep(1.5) will be set to -1. с c iw integer array of length 8*n. if the option nsrch.le.n is c used, then the length of array iw can be reduced to 7*n. c w real array length n. used by mc24a/ad both as workspace and to return growth estimate in w(1). the use of this array by ma28a/ad С is thus optional depending on common block logical variable grow. C c iflag integer variable used as error flag by routine. a positive c or zero value on exit indicates success. possible negative values are -1 through -14. с с integer n, nz, licn, lirn, iftag integer irn(lirn), icn(licn), ikeep(n,5), iw(n,8) REAL a(licn), u, w(n) С c common and private variables. c common block ma28f/fd is used merely to communicate with common block ma30f/fd so that the user need not declare this common block in his main program. с c the common block variables are as follows ... c lp,mp integer default value 6 (line printer). unit number c for error messages and duplicate element warning resp. c nlp,mlp integer unit number for messages from ma30a/ad and c mc23a/ad resp. set by ma28a/ad to value of lp. c lblock logical default value true, if true mc23a/ad is used c to first permute the matrix to block lower triangular form. c grow logical default value true. if true then an estimate c of the increase in size of matrix elements during l/u c decomposition is given by mc24a/ad. c eps,rmin,resid real/double precision variables not referenced

```
c by ma28a/ad.
```

c irrcp, icncp integer set to number of compresses on arrays irn and c $% \left({\frac{{{\left[{{c_{1}} \right]}}}{{\left[{{c_{1}} \right]}}}} \right)$

c minim, minicn integer minimum length of arrays irn and icn/a

c respectively, for success on future runs. c irank integer estimated rank of matrix.

c mirnep,micnep,mirank,mirn,micn integer variables. used to

c communicate between ma30f/fd and ma28f/fd values of abovenamed

c variables with somewhat similar names.

c abort1,abort2 logical variables with default value true. if false

c then decomposition will be performed even if the matrix is

c structurally or numerically singular respectively.

c aborta, abortb logical variables used to communicate values of

c abort1 and abort2 to ma30a/ad.

c abort logical used to communicate value of abort1 to mc23a/ad. c abort3 logical variable not referenced by ma28a/ad.

c idisp integer array length 2. used to communicate information

c on decomposition between this call to ma28a/ad and subsequent

c calls to ma28b/bd and ma28c/cd. on exit, idisp(1) and

c idisp(2) indicate position in arrays a and icn of the

c first and last elements in the l/u decomposition of the

c diagonal blocks, respectively.

c numnz integer structural rank of matrix.

c num integer number of diagonal blocks.

c large integer size of largest diagonal block.

с

c see block data for further comments on common block variables. c see code for comments on private variables.

с REAL tol, themax, big, dxmax, errmax, dres, cgce, * toll, big1, upriv, rmin, eps, resid, zero integer idisp(2) logical grow, lblock, abort, abort1, abort2, abort3, aborta, * abortb, lbig, lbig1 common /ma28ed/ lp, mp, lblock, grow common /ma28fd/ eps, rmin, resid, irncp, icncp, minirn, minicn, * irank, abort1, abort2 common /ma28gd/ idisp common /ma28hd/ tol, themax, big, errmax, dres, cgce, * ndrop, maxit, noiter, nsrch, istart, lbig common /ma30id/ tol1, big1, ndrop1, nsrch1, Ibig1 common /ma30ed/ nlp, aborta, abortb, abort3 common /ma30fd/ mirnep, mienep, mirank, mirn, mien common /mc23bd/ mlp, numnz, num, large, abort common /lpivot/ lpiv(10),lnpiv(10),mapiv,manpiv,iavpiv, ianpiv,kounti external ma28jd с

c some initialization and transfer of information between

c common blocks (see earlier comments). data zero /0.0d0/ iflag = 0

aborta = abort1 abortb = abort2 abort = abort1 mlp = lp nlp = lp tol1 = tol lbig1 = lbig nsrch1 = nsrch

c upriv private copy of u is used in case it is outside

 c range zero to one and is thus altered by ma30a/ad. upriv = u
 c simple data check on input variables and array dimensions.

if (n.gt.0) go to 10 iflag = -8 if (1p.ne.0) write (1p,99999) n go to 210 10 if (nz.gt.0) go to 20 iflag = -9 if (1p.ne.0) write (1p,99998) nz go to 210 20 if (licn.ge.nz) go to 30

```
iflag = -10
    if (lp.ne.0) write (lp,99997) licn
    go to 210
  30 if (lim.ge.nz) go to 40
    iflag = -11
    if (lp.ne.0) write (lp,99996) lim
    go to 210
с
c data check to see if all indices lie between 1 and n.
  40 do 50 i=1,nz
     if (irn(i).gt.0 .and. irn(i).le.n .and. icn(i).gt.0 .and.
    * icn(i).le.n) go to 50
     if (iflag.eq.0 .and. lp.ne.0) write (lp,99995)
     iflag = -12
     if (lp.ne.0) write (lp,99994) i, a(i), irn(i), icn(i)
  50 continue
    if (iflag.lt.0) go to 220
с
c sort matrix into row order.
    call mc20ad(n, nz, a, icn, iw, irn, 0)
c part of ikeep is used here as a work-array. ikeep(i,2) is
c the last row to have a non-zero in column i. ikeep(i,3)
   is the off-set of column i from the start of the row.
с
    do 60 i=1,n
     ikeep(i,2) = 0
     ikeep(i,1) = 0
  60 continue
с
c check for duplicate elements .. summing any such entries and
c printing a warning message on unit mp.
c move is equal to the number of duplicate elements found.
    move = 0
c the loop also calculates the largest element in the matrix, themax.
    themax = zero
c jl is position in arrays of first non-zero in row.
    j1 = iw(1,1)
    do 130 i=1,n
     iend = nz + 1
     if (i,ne,n) iend = iw(i+1,1)
     length = iend - j1
     if (length.eq.0) go to 130
     i2 = iend - 1
     newjl = jl - move
     do 120 jj=j1,j2
      j = icn(jj)
      themax = abs(a(jj))
      if (ikeep(j,2).eq.i) go to 110
c first time column has ocurred in current row.
      ikeep(j,2) = i
      ikeep(j,3) = jj - move - newj1
      if (move.eq.0) go to 120
c shift necessary because of previous duplicate element.
      newpos = jj - move
      a(newpos) = a(jj)
      icn(newpos) = icn(jj)
      go to 120
c duplicate element.
 110 move = move + 1
      length = length - 1
      jay = ikeep(j,3) + newj1
      if (mp.ne.0) write (mp,99993) i, j, a(jj)
      a(jay) = a(jay) + a(jj)
themax = abs(a(jay))
 120 continue
     ikeep(i,1) = length
     jl = iend
 130 continue
c
c knum is actual number of non-zeros in matrix with any multiple
   entries counted only once.
с
    knum = nz - move
    if (.not.lblock) go to 140
```

С

```
c perform block triangularisation.
    call mc23ad(n, icn, a, licn, ikeep, idisp, ikeep(1,2),
    *ikeep(1,3), ikeep(1,5), iw(1,3), iw)
    if (idisp(1).gt.0) go to 170
    iflag = -7
    if (idisp(1).eq.-1) iflag = -1
    if (lp.ne.0) write (lp,99992)
    go to 210
c
c block triangularization not requested.
c move structure to end of data arrays in preparation for
c ma30a/ad.
c also set lenoff(1) to -1 and set permutation arrays.
 140 do 150 i=1,knum
     ii = knum - i + 1
     newpos = licn - i + 1
     icn(newpos) = icn(ii)
     a(newpos) = a(ii)
  150 continue
    idisp(1) = 1
    idisp(2) = licn - knum + 1
    do 160 i=1,n
     ikeep(i,2) = i
     ikeep(i,3) = i
  160 continue
    ikeep(1,5) = -1
 170 \text{ if (lbig) big} = \text{themax}
    if (nsrch.le.n) go to 180
c
c perform l/u decomosition on diagonal blocks.
    call ma30ad(n, icn, a, licn, ikeep, ikeep(1,4), idisp,
    *ikeep(1,2), ikeep(1,3), irn, lirn, iw(1,2), iw(1,3), iw(1,4),
   *iw(1,5), iw(1,6), iw(1,7), iw(1,8), iw, upriv, iflag)
    go to 190
c this call if used if nsrch has been set less than or equal n.
c in this case, two integer work arrays of length can be saved.
  180 call ma30ad(n, icn, a, licn, ikeep, ikeep(1,4), idisp,
    * ikeep(1,2), ikeep(1,3), irn, lirn, iw(1,2), iw(1,3), iw(1,4),
   * iw(1,5), iw, iw, iw(1,6), iw, upriv, iflag)
с
c transfer common block information.
  190 \text{ minim} = \text{max0(mirn,nz)}
    minicn = max0(micn,nz)
    imcp = mimcp
    icnep = mienep
    irank = mirank
    ndrop = ndrop1
    if (lbig) big = big1
    if (iflag.ge.0) go to 200
    if (lp.ne.0) write (lp,99991)
    go to 210
c
c reorder off-diagonal blocks according to pivot permutation.
  200 \text{ i1} = \text{idisp}(1) - 1
    if (i1.ne.0) call mc22ad(n, icn, a, i1, ikeep(1,5), ikeep(1,2),
    * ikeep(1,3), iw, im)
    il = idisp(1)
    iend = licn - i1 + 1
c
c optionally calculate element growth estimate.
    if (grow) call mc24ad(n, icn, a(i1), iend, ikeep, ikeep(1,4), w)
c increment growth estimate by original maximum element.
     if (\text{grow}) w(1) = w(1) + \text{themax}
     if (grow .and. n.gt.1) w(2) = themax
c set flag if the only error is due to duplicate elements.
    if (iflag.ge.0 .and. move.ne.0) iflag = -14
    go to 220
  210 if (lp.ne.0) write (lp,99990)
  220 return
 99999 format (36x, 17hn out of range =, i10)
 99998 format (36x, 18hnz non positive =, i10)
 99997 format (36x, 17hlien too small =, i10)
 99996 format (36x, 17hlirn too small = , i10)
```

99995 format (54h error return from ma28a/ad because indices found out, * 8hof range) 99994 format (1x, i6, 22hth element with value, 1pd22.14, 9h is out o, * 21hf range with indices , i8, 2h ,, i8) 99993 format (31h duplicate element in position, i8, 2h,, i8, * 12h with value, 1pd22.14) 99992 format (36x, 26herror return from mc23a/ad) 99991 format (36x, 26herror return from ma30a/ad) 99990 format (36h+error return from ma28a/ad because) end c#####date 01 jan 1984 copyright ukaea, harwell. c######alias ma30ad subroutine ma30ad(nn, icn, a, licn, lenr, lenrl, idisp, ip, iq, * irn, lim, lenc, ifirst, lastr, nextr, lastc, nextc, iptr, ipc, * u, iflag) c if the user requires a more convenient data interface then the ma28 package should be used. the ma28 subroutines call the ma30 с subroutines after checking the user's input data and optionally с using mc23a/ad to permute the matrix to block triangular form. C c this package of subroutines (ma30a/ad, ma30b/bd, ma30c/cd and c ma30d/dd) performs operations pertinent to the solution of a general sparse n by n system of linear equations (i.e. solve ¢ с ax=b), structually singular matrices are permitted including those with row or columns consisting entirely of zeros (i.e. с including rectangular matrices). it is assumed that the с non-zeros of the matrix a do not differ widely in size. if с necessary a prior call of the scaling subroutine mc19a/ad may be c made. с c a discussion of the design of these subroutines is given by duff and reid (acm trans math software 5 pp 18-35,1979 (css 48)) while с fuller details of the implementation are given in duff (harwell с с report aere-r 8730,1977). the additional pivoting option in с ma30a/ad and the use of drop tolerances (see common block ma30i/id) were added to the package after joint work with reid, с schaumburg, wasniewski and zlatev (duff, reid, schaumburg, с wasniewski and zlatev, harwell report css 135, 1983). с c c ma30a/ad performs the lu decomposition of the diagonal blocks of the с permutation paq of a sparse matrix a, where input permutations p1 and q1 are used to define the diagonal blocks. there may be с non-zeros in the off-diagonal blocks but they are unaffected by ¢ ma30a/ad. p and p1 differ only within blocks as do q and q1. the с permutations p1 and q1 may be found by calling mc23a/ad or the с с matrix may be treated as a single block by using p1=q1=i, the ¢ matrix non-zeros should be held compactly by rows, although it should be noted that the user can supply the matrix by columns ¢ to get the lu decomposition of a transpose. с с c the parameters are ... c this description should also be consulted for further information on most of the parameters of ma30b/bd and ma30c/cd. С c c n is an integer variable which must be set by the user to the order c of the matrix, it is not altered by ma30a/ad. c icn is an integer array of length licn. positions idisp(2) to c licn must be set by the user to contain the column indices of the non-zeros in the diagonal blocks of p1*a*q1. those belonging с to a single row must be contiguous but the ordering of column c indices with each row is unimportant, the non-zeros of row i c precede those of row i+1,i=1,...,n-1 and no wasted space is с allowed between the rows. on output the column indices of the с lu decomposition of paq are held in positions idisp(1) to с idisp(2), the rows are in pivotal order, and the column indices с of the l part of each row are in pivotal order and precede those c с of u, again there is no wasted space either within a row or between the rows. icn(1) to icn(idisp(1)-1), are neither с required nor altered. if mc23a/ad been called, these will hold с information about the off-diagonal blocks. с c a is a real/double precision array of length licn whose entries

- c idisp(2) to lich must be set by the user to the values of the
- c non-zero entries of the matrix in the order indicated by icn.
- c non-zero encres or the matrix in the order materialed by ten
- on output a will hold the lu factors of the matrix where again
 the position in the matrix is determined by the corresponding

с values in icn. a(1) to a(idisp(1)-1) are neither required nor altered. с

- c licn is an integer variable which must be set by the user to the
- length of arrays icn and a. it must be big enough for a and icn с
- to hold all the non-zeros of I and u and leave some "elbow с
- room". it is possible to calculate a minimum value for licn by с
- a preliminary run of ma30a/ad, the adequacy of the elbow room с can be judged by the size of the common block variable icnep. it с
- is not altered by ma30a/ad. с
- c lenr is an integer array of length n. on input, lenr(i) should
- equal the number of non-zeros in row i, i=1,...,n of the
- diagonal blocks of p1*a*q1. on output, lenr(i) will equal the с
- total number of non-zeros in row i of l and row i of u. с
- c lenrl is an integer array of length n. on output from ma30a/ad,
- lenrl(i) will hold the number of non-zeros in row i of l. c c idisp is an integer array of length 2. the user should set idisp(1)
- c to be the first available position in a/icn for the lu
- с
- decomposition while idisp(2) is set to the position in a/icn of the first non-zero in the diagonal blocks of p1*a*o1. on output,
- с idisp(1) will be unaltered while idisp(2) will be set to the с
- position in a/icn of the last non-zero of the lu decomposition. с
- c ip is an integer array of length n which holds a permutation of
- c the integers 1 to n. on input to ma30a/ad, the absolute value of ip(i) must be set to the row of a which is row i of p1*a*q1. a С
- negative value for ip(i) indicates that row i is at the end of a
- с diagonal block. on output from ma30a/ad, ip(i) indicates the row
- с
- of a which is the i th row in pag. ip(i) will still be negative С

for the last row of each block (except the last). с

- c iq is an integer array of length n which again holds a
- permutation of the integers 1 to n. on input to ma30a/ad, iq(j) с
- must be set to the column of a which is column j of p1*a*q1. on с
- output from ma30a/ad, the absolute value of iq(j) indicates the с
- column of a which is the j th in paq. for rows, i say, in which с structural or numerical singularity is detected iq(i) is с
- c negated.
- c irn is an integer array of length lirn used as workspace by c ma30a/ad.
- c lim is an integer variable, it should be greater than the
- c largest number of non-zeros in a diagonal block of p1*a*q1 but
- need not be as large as licn. it is the length of array irn and
- should be large enough to hold the active part of any block, с
- plus some "elbow room", the a posteriori adequacy of which can c
- be estimated by examining the size of common block variable с
- irncp. с

c lenc, ifirst, lastr, nextr, lastc, nextc are all integer arrays of

- length n which are used as workspace by ma30a/ad. if nsrch is c
- set to a value less than or equal to n, then arrays lastc and с
- nextc are not referenced by ma30a/ad and so can be dummied in с
- с the call to ma30a/ad.

c iptr,ipc are integer arrays of length n which are used as workspace c by ma30a/ad.

c u is a real/double precision variable which should be set by the

- c user to a value between 0. and 1.0. if less than zero it is
- c reset to zero and if its value is 1.0 or greater it is reset to
- 0.9999 (0.9999999999 in d version). it determines the balance с
- between pivoting for sparsity and for stability, values near ¢
- с zero emphasizing sparsity and values near one emphasizing
- stability, we recommend u=0.1 as a possible first trial value. с
- the stability can be judged by a later call to mc24a/ad or by с setting lbig to .true. с
- c iflag is an integer variable, it will have a non-negative value if
- ma30a/ad is successful, negative values indicate error c
- conditions while positive values indicate that the matrix has c
- been successfully decomposed but is singular. for each non-zero с
- value, an appropriate message is output on unit lp. possible с
- с non-zero values for iflag are ...
- с
- c -1 the matrix is structually singular with rank given by irank in c common block ma30f/fd.
- c+1 if, however, the user wants the lu decomposition of a
- c structurally singular matrix and sets the common block variable
- abort1 to .false., then, in the event of singularity and a
- c successful decomposition, iflag is returned with the value +1

- c and no message is output.
- c -2 the matrix is numerically singular (it may also be structually
- singular) with estimated rank given by irank in common block
 ma30f/fd.
- $c + 2 \ \ the \ \ user \ can choose to continue the decomposition even when a$
- c zero pivot is encountered by setting common block variable
- c abort2 to .false. if a singularity is encountered, iflag will
- c then return with a value of +2, and no message is output if the
- c decomposition has been completed successfully.
- c -3 lim has not been large enough to continue with the
- c decomposition. if the stage was zero then common block variable
- c minim gives the length sufficient to start the decomposition on
- c this block. for a successful decomposition on this block the
- c user should make lirn slightly (say about n/2) greater than this
- c value.
- c -4 licn not large enough to continue with the decomposition.
- c -5 the decomposition has been completed but some of the lu factors
- c have been discarded to create enough room in a/icn to continue
- c the decomposition, the variable minicn in common block ma30f/fd
- c then gives the size that licn should be to enable the
- c factorization to be successful. if the user sets common block
- c variable abort3 to .true., then the subroutine will exit
- c immediately instead of destroying any factors and continuing.
- c -6 both licn and lim are too small, termination has been caused by
- c lack of space in irrn (see error iflag=-3), but already some of
- c the lu factors in a/icn have been lost (see error iflag=-5).
- c minich gives the minimum amount of space required in a/ich for
- c decomposition up to this point.
- c

REAL a(licn), u, au, umax, amax, zero, pivrat, pivr, * tol, big, anew, aanew, scale

- integer iptr(nn), pivot, pivend, dispc, oldpiv, oldend, pivrow, * rowi, ipc(nn), idisp(2), colupd
- integer icn(licn), lenr(nn), lenrl(nn), ip(nn), iq(nn),
- * lenc(nn), irn(lirn), ifirst(nn), lastr(nn), nextr(nn),
- * lastc(nn), nextc(nn)
- logical abort1, abort2, abort3, lbig
- c for comments of common block variables see block data subprogram. common /ma30ed/ lp, abort1, abort2, abort3 common /ma30fd/ irncp, iencp, irank, minirn, minien common /ma30id/ tol, big, ndrop, nsrch, lbig
- common /lpivot/ lpiv(10), lnpiv(10), mapiv, manpiv, iavpiv,
- ianpiv,kountl

iactiv = idisp(2)

external ma30jd с data umax/.99999999900/ data zero /0.0d0/ msrch = nsrch ndrop = 0 do 1272 kk=1,10 Inpiv(kk)=0 lpiv(kk)=0 1272 continue mapiv = 0manpiv = 0iavpiv = 0ianpiv = 0kountl = 0 minirn = 0 minicn = idisp(1) - 1morei = 0irank = nn imcp = 0icncp = 0iflag = 0 c reset u if necessary. u = umaxc ibeg is the position of the next pivot row after elimination step using it. с u = zero ibeg = idisp(1)c jactiv is the position of the first entry in the active part of a/icn.

```
c nzrow is current number of non-zeros in active and unprocessed part
c of row file icn.
    nzrow = licn - iactiv + 1
    minicn = nzrow + minicn
с
c count the number of diagonal blocks and set up pointers to the
c beginnings of the rows.
c num is the number of diagonal blocks.
    num = 1
    iptr(1) = iactiv
    if (nn.eq.1) go to 20
   nnm1 = nn - 1
    do 10 i=1,nnm1
     if (ip(i).lt.0) num = num + 1
     iptr(i+1) = iptr(i) + lenr(i)
  10 continue
c ilast is the last row in the previous block.
  20 ilast = 0
c
c **** lu decomposition of block nblock ****
с
c each pass through this loop performs lu decomposition on one
c of the diagonal blocks.
   do 1000 nblock=1,num
     istart = ilast + 1
     do 30 irows=istart,nn
      if (ip(irows).lt.0) go to 40
  30 continue
     irows = nn
  40 ilast = irows
c n is the number of rows in the current block.
c istart is the index of the first row in the current block.
c ilast is the index of the last row in the current block.
c jactiv is the position of the first entry in the block.
c itop is the position of the last entry in the block.
     n = ilast - istart + 1
     if (n.ne.1) go to 90
с
c code for dealing with 1x1 block.
     lenrl(ilast) = 0
     ising = istart
     if (lenr(ilast).ne.0) go to 50
c block is structurally singular.
     irank = irank - 1
     ising = -ising
     if (iflag.ne.2 .and. iflag.ne.-5) iflag = 1
     if (.not.abort1) go to 80
     idisp(2) = iactiv
     iflag = -1
     if (lp.ne.0) write (lp,99999)
    return
¢
    go to 1120
  50 scale = abs(a(iactiv))
     if (scale.eq.zero) go to 60
     if (lbig) big = scale
    go to 70
  60^{\circ} ising = -ising
    irank = irank - 1
     iptr(ilast) = 0
     if (iflag.ne.-5) if lag = 2
     if (.not.abort2) go to 70
     idisp(2) = iactiv
iflag = -2
     if (lp.ne.0) write (lp,99998)
     go to 1120
  70 a(ibeg) = a(iactiv)
     icn(ibeg) = icn(iactiv)
     iactiv = iactiv + 1
     iptr(istart) = 0
     ibeg = ibeg + 1
     nzrow = nzrow - 1
```

```
80 lastr(istart) = istart
      ipc(istart) = -ising
     go to 1000
с
c non-trivial block.
  90 itop = licn
     if (ilast.ne.nn) itop = iptr(ilast+1) - 1
с
c set up column oriented storage.
     do 100 i=istart,ilast
      lenrl(i) = 0
      lenc(i) = 0
100 continue
     if (itop-iactiv.lt.lirn) go to 110
     minirn = itop - iactiv + 1
     pivot = istart - 1
     go to 1100
¢
c calculate column counts,
 110 do 120 ii=iactiv,itop
       i = icn(ii)
       lenc(i) = lenc(i) + 1
 120 continue
c set up column pointers so that ipc(j) points to position after end
c of column j in column file.
     ipc(ilast) = lirn + 1
jl = istart + 1
     do 130 jj=j1,ilast
      j = ilast - jj + j1 - 1
      ipc(j) = ipc(j+1) - lenc(j+1)
 130 continue
     do 150 indrow=istart,ilast
      j1 = iptr(indrow)
      j2 = j1 + lenr(indrow) - I
      if (j1.gt.j2) go to 150
      do 140 jj=j1,j2
       j = icn(jj)
        ipos = ipc(i) - 1
        irn(ipos) = indrow
        ipc(j) = ipos
 140 continue
 150 continue
c dispc is the lowest indexed active location in the column file.
     dispc = ipc(istart)
     nzcol = lirn - dispc + 1
     minirn = max0(nzcol,minirn)
     nzmin = 1
с
c initialize array ifirst. ifirst(i) = +/- k indicates that row/col
c k has i non-zeros. if ifirst(i) = 0, there is no row or column
    with i non zeros.
с
     do 160 i=1,n
      ifirst(i) = 0
 160 continue
¢
c compute ordering of row and column counts.
c first run through columns (from column n to column 1).
     do 180 jj=istart,ilast
      j = ilast - jj + istart
      nz = lenc(j)
      if (nz.ne.0) go to 170
      ipc(j) = 0
      go to 180
 170 if (nsrch.le.nn) go to 180
      isw = ifirst(nz)
      ifirst(nz) = -j
      lastc(j) = 0
      nextc(j) = -isw
      iswl = abs(isw)
      if (isw.ne.0) lastc(isw1) = j
 180 continue
c now run through rows (again from n to 1).
     do 210 ii=istart,ilast
```
<u>Anexo II</u>

i = ilast - ii + istart nz = lenr(i)if (nz.ne.0) go to 190 iptr(i) = 0lastr(i) = 0go to 210 190 isw = ifirst(nz)ifirst(nz) = i if (isw.gt.0) go to 200 nextr(i) = 0lastr(i) = isw go to 210 200 nextr(i) = isw lastr(i) = lastr(isw) lastr(isw) = i210 continue с с c **** start of main elimination loop **** do 980 pivot=istart,ilast с c first find the pivot using markowitz criterion with stability c control. c jcost is the markowitz cost of the best pivot so far,.. this c pivot is in row ipiv and column jpiv. nz2 = nzminicost = n*n с c examine rows/columns in order of ascending count. do 340 l=1,2 pivrat = zero isrch = 111 = 1c a pass with l equal to 2 is only performed in the case of singularity. do 330 nz=nz2,n if (jcost.le.(nz-1)**2) go to 420 ijfir = ifirst(nz) if (ijfir) 230, 220, 240 220 if (ll.eq.1) nzmin = nz + 1go to 330 230 11 == 2 ijfir = -ijfir go to 290 240 11 = 2c scan rows with nz non-zeros. do 270 idummy=1,n if (jcost.le.(nz-1)**2) go to 420 if (isrch.gt.msrch) go to 420 if (ijfir.eq.0) go to 280 c row ijfir is now examined. i = iifir ijfir = nextr(i) c first calculate multiplier threshold level. amax = zero j1 = iptr(i) + lenrl(i)j2 = iptr(i) + lenr(i) - 1do 250 jj=j1,j2 amax = abs(a(jj))250 continue au = amax*u isrch = isrch + 1c scan row for possible pivots do 260 jj=j1,j2 if (abs(a(jj)).le.au .and. l.eq.1) go to 260 j = icn(jj)kcost = (nz-1)*(lenc(j)-1)if (kcost.gt.jcost) go to 260 pivr = zero if (amax.ne.zero) pivr = abs(a(jj))/amax if (kcost.eq.jcost and (pivr.le.pivrat .or. nsrch.gt.nn+1)) go to 260

c best pivot so far is found. jcost = kcost ijpos = jj ipiv = i jpiv = j if (msrch.gt.nn+1 and, jcost.le.(nz-1)**2) go to 420 pivrat = pivr 260 continue 270 continue с c columns with nz non-zeros now examined. 280 ijfir = ifirst(nz) ijfir = -lastr(ijfir) if (jcost.le.nz*(nz-1)) go to 420 290 if (msrch.le.nn) go to 330 do 320 idummy=1,n if (ijfir.eq.0) go to 330 j = ijfir ijfir = nextc(ijfir) i1 = ipc(j)i2 = i1 + nz - 1c scan column j. do 310 ii=i1,i2 i = irn(ii)kcost = (nz-1)*(lenr(i)-lenrl(i)-1) if (kcost.ge.jcost) go to 310 c pivot has best markowitz count so far ... now check its c suitability on numeric grounds by examining the other non-zeros c in its row. j1 = iptr(i) + lenrl(i)j2 = iptr(i) + lenr(i) - 1c we need a stability check on singleton columns because of possible c problems with underdetermined systems. amax = zero do 300 jj=j1,j2 amax = abs(a(jj))if (icn(jj).eq.j) jpos = jj 300 continue if (abs(a(jpos)).le.amax*u .and. l.eq.1) go to 310 jcost = kcost ipiv = i jpiv = j ijpos = jpos if (amax.ne.zero) pivrat = abs(a(jpos))/amax if (jcost.le.nz*(nz-1)) go to 420 310 continue ¢ 320 continue ¢ 330 continue c in the event of singularity, we must make sure all rows and columns c are tested. msrch = nс c matrix is numerically or structurally singular ... which it is will be diagnosed later. с irank = irank - 1 340 continue c assign rest of rows and columns to ordering array. c matrix is structurally singular. if (iflag.ne.2 and iflag.ne.-5) iflag = 1 irank = irank - ilast + pivot + 1 if (.not.abort1) go to 350 idisp(2) = iactiviflag = -1 if (lp.ne.0) write (lp,99999) go to 1120 350 k = pivot - 1 do 390 i=istart,ilast if (lastr(i).ne.0) go to 390 k = k + 1lastr(i) = kif (lenrl(i).eq.0) go to 380

```
minicn = max0(minicn,nzrow+ibeg-1+morei+lenrl(i))
        if (iactiv-ibeg.ge.lenrl(i)) go to 360
        call ma30dd(a, icn, iptr(istart), n, iactiv, itop, .true.)
c check now to see if ma30d/dd has created enough available space.
        if (iactiv-ibeg.ge.lenrl(i)) go to 360
c create more space by destroying previously created lu factors.
        morei = morei + ibeg - idisp(1)
        ibeg = idisp(1)
        if (lp.ne.0) write (lp.99997)
        iflag = -5
        if (abort3) go to 1090
         j1 = iptr(i)
 360
       j2 = j1 + lenrl(i) - 1
       iptr(i) = 0
       do 370 jj=j1,j2
         a(ibeg) = a(ij)
         icn(ibeg) = icn(jj)
        icn(jj) = 0
ibeg = ibeg + 1
 370
         continue
       nzrow = nzrow - lenrl(i)
 380
        if (k.eq.ilast) go to 400
 390
        continue
 400 k = pivot - 1
      do 410 i=istart,ilast
       if (ipc(i).ne.0) go to 410
       k = k + 1
       ipc(i) = k
       if (k.eq.ilast) go to 990
 410 continue
с
c the pivot has now been found in position (ipiv, jpiv) in location
c ijpos in row file.
c update column and row ordering arrays to correspond with removal
  of the active part of the matrix.
с
 420 ising = pivot
      if (a(ijpos).ne.zero) go to 430
c numerical singularity is recorded here.
      ising = -ising
if (iflag.ne.-5) iflag = 2
      if (.not.abort2) go to 430
      idisp(2) = iactiv
      iflag = -2
      if (lp.ne.0) write (lp,99998)
      go to 1120
 430 oldpiv = iptr(ipiv) + lenrl(ipiv)
oldend = iptr(ipiv) + lenr(ipiv) - 1
c changes to column ordering.
      if (nsrch.le.nn) go to 460
      colupd = nn + 1
       lenpp = oldend-oldpiv+1
       if (lenpp.lt.4) lpiv(1) = lpiv(1) + 1
       if (lenpp.ge.4 and lenpp.le.6) lpiv(2) = lpiv(2) + 1
       if (lenpp.ge.7 and, lenpp.le, 10) lpiv(3) = lpiv(3) + 1
       if (lenpp.ge.11 and lenpp.le.15) lpiv(4) = lpiv(4) + 1
       if (lenpp.ge.16 and lenpp.le.20) lpiv(5) = lpiv(5) + 1
       if (lenpp.ge.21 and lenpp.le.30) lpiv(6) = lpiv(6) + 1
       if (lenpp.ge.31 and lenpp.le.50) lpiv(7) = lpiv(7) + 1
       if (lenpp.ge.51 and lenpp.le.70) lpiv(8) = lpiv(8) + 1
       if (lenpp.ge.71 and lenpp.le.100) lpiv(9) = lpiv(9) + 1
       if (lenpp.ge.101) lpiv(10) = lpiv(10) + 1
       mapiv = max0(mapiv,lenpp)
       iavpiv = iavpiv + lenpp
      do 450 jj=oldpiv,oldend
       j = icn(jj)
       lc = lastc(j)
       nc = nextc(i)
       nextc(j) = -colupd
       if (jj.ne.ijpos) colupd = j
       if (nc.ne.0) lastc(nc) = lc
       if (lc.eq.0) go to 440
       nextc(lc) = nc
       go to 450
```

440 nz = lenc(j)isw = ifirst(nz)if (isw.gt.0) lastr(isw) = -ncif (isw.lt.0) if irst(nz) = -nc 450 continue c changes to row ordering. 460 il = ipc(jpiv) i2 = i1 + lenc(jpiv) - 1do 480 ii=i1,i2 i = irn(ii)lr = lastr(i)nr = nextr(i)if (nr.ne.0) lastr(nr) = lrif (lr.le.0) go to 470 nextr(lr) = nrgo to 480 470 nz = lenr(i) - lenrl(i)if (nr.ne.0) if irst(nz) = nr if (nr.eq.0) ifirst(nz) = lr 480 continue с c move pivot to position lenrI+1 in pivot row and move pivot row c to the beginning of the available storage. c the l part and the pivot in the old copy of the pivot row is c nullified while, in the strictly upper triangular part, the c column indices, j say, are overwritten by the corresponding entry of iq (iq(j)) and iq(j) is set to the negative of the с displacement of the column index from the pivot entry. с if (oldpiv.eq.ijpos) go to 490 au = a(oldpiv) a(oldpiv) = a(ijpos) a(ijpos) = au icn(ijpos) = icn(oldpiv) icn(oldpiv) = jpiv c check to see if there is space immediately available in a/icn to c hold new copy of pivot row. 490 minicn = max0(minicn,nzrow+ibeg-1+morei+lenr(ipiv)) if (iactiv-ibeg.ge.lenr(ipiv)) go to 500 call ma30dd(a, icn, iptr(istart), n, iactiv, itop, true.) oldpiv = iptr(ipiv) + lenrl(ipiv) oldend = iptr(ipiv) + lenr(ipiv) - 1c check now to see if ma30d/dd has created enough available space. if (iactiv-ibeg.ge.lenr(ipiv)) go to 500 c create more space by destroying previously created lu factors. morei = morei + ibeg - idisp(1)ibeg = idisp(1)if (lp.ne.0) write (lp,99997) iflag = -5if (abort3) go to 1090 if (iactiv-ibeg.ge.lenr(ipiv)) go to 500 c there is still not enough room in a/icn. iflag = -4 go to 1090 c copy pivot row and set up iq array. ijpos = 0 500 jl = iptr(ipiv) с do 530 jj=j1,oldend a(ibeg) = a(jj)icn(ibeg) = icn(ij)if (ijpos.ne.0) go to 510 if (icn(jj).eq.jpiv) ijpos = ibeg icn(jj) = 0go to 520 k = ibeg - ijpos 510 j = icn(jj)icn(jj) = iq(j)iq(j) = -k520 ibeg = ibeg + 1530 continue с ijpl = ijpos + 1pivend = ibeg - 1

```
lenpiv = pivend - iipos
      nzrow = nzrow - lenrl(ipiv) - 1
      iptr(ipiv) = oldpiv + 1
      if (lenpiv.eq.0) iptr(ipiv) = 0
с
c remove pivot row (including pivot) from column oriented file.
      do 560 jj=ijpos,pivend
       i = icn(ji)
       i1 = ipc(j)
       lenc(j) = lenc(j) - 1
c i2 is last position in new column.
       i2 = ipc(j) + lenc(j) - 1
       if (i2.lt.i1) go to 550
       do 540 ii=i1,i2
        if (irn(ii).ne.ipiv) go to 540
        irm(ii) = irm(i2+1)
        go to 550
 540
         continue
 550
        irn(i2+1) = 0
 560 continue
      nzcol = nzcol - lenpiv - l
с
c go down the pivot column and for each row with a non-zero add
c the appropriate multiple of the pivot row to it.
c we loop on the number of non-zeros in the pivot column since
   ma30d/dd may change its actual position.
С
c
      nzpc = lenc(jpiv)
      if (nzpc.eq.0) go to 900
      do 840 iii=1 nzpc
       ii = ipc(ipiv) + iii - 1
       i = irn(ii)
c search row i for non-zero to be eliminated, calculate multiplier,
   and place it in position lenrl+1 in its row.
¢
c idrop is the number of non-zero entries dropped from row i
      because these fall beneath tolerance level.
¢
c
       idrop = 0
       j1 = iptr(i) + lenrl(i)
       iend = iptr(i) + lenr(i) - 1
       do 570 jj=j1,iend
        if (icn(jj).ne.jpiv) go to 570
c if pivot is zero, rest of column is and so multiplier is zero.
         au = zero
         if (a(ijpos).ne.zero) au = -a(jj)/a(ijpos)
         if (lbig) big = abs(au)
         a(jj) = a(j1)
         a(i1) = au
         icn(jj) = icn(j1)
        icn(j1) = jpiv
lenrl(i) = lenrl(i) + 1
         go to 580
 570
         continue
c jump if pivot row is a singleton.
 580
        if (lenpiv.eq.0) go to 840
c now perform necessary operations on rest of non-pivot row i.
       rowi = j1 + 1
       iop = 0
c jump if all the pivot row causes fill-in.
       if (rowi.gt.iend) go to 650
c perform operations on current non-zeros in row i.
c innermost loop.
        lenpp = iend-rowi+1
        if (lenpp.lt.4) lnpiv(1) = lnpiv(1) + 1
        if (lenpp.ge.4 .and. lenpp.le.6) lnpiv(2) = lnpiv(2) + 1
       if (lenpp.ge.7 and lenpp.le.10) lnpiv(3) = lnpiv(3) + 1
        if (lenpp.ge.11 and lenpp.le.15) lnpiv(4) = lnpiv(4) + 1
        if (lenpp.ge.16 and lenpp.le.20) lnpiv(5) = lnpiv(5) + 1
        if (lenpp.ge.21 and lenpp.le.30) lnpiv(6) = lnpiv(6) + 1
        if (lenpp.ge.31 and lenpp.le.50) lnpiv(7) = lnpiv(7) + 1
        if (lenpp.ge.51 and lenpp.le.70) lnpiv(8) = lnpiv(8) + 1
        if (lenpp.ge.71 .and. lenpp.le.100) lnpiv(9) = lnpiv(9) + 1
        if (lenpp.ge.101) lnpiv(10) = lnpiv(10) + 1
```

```
manpiv = max0(manpiv,lenpp)
        ianpiv = ianpiv + lenpp
        kountl = kountl + 1
        do 590 jj=rowi,iend
         j = icn(jj)
         if (iq(j).gt.0) go to 590
          iop = iop + 1
          pivrow = ijpos - iq(j)
         a(jj) = a(jj) + au*a(pivrow)
if (lbig) big = abs(a(jj))
          icn(pivrow) = -icn(pivrow)
         if (abs(a(jj)).lt.tol) idrop = idrop + 1
  590
          continue
¢
c jump if no non-zeros in non-pivot row have been removed
c
      because these are beneath the drop-tolerance tol.
с
        if (idrop.eq.0) go to 650
с
c run through non-pivot row compressing row so that only
     non-zeros greater than tol are stored, all non-zeros
с
     less than tol are also removed from the column structure.
С
с
        jnew = rowi
        do 630 jj=rowi,iend
         if (abs(a(jj)).lt.tol) go to 600
         a(jnew) = a(jj)
         icn(jnew) = icn(jj)
         jnew = jnew + 1
         go to 630
с
c remove non-zero entry from column structure.
С
 600
          i = icn(jj)
         i1 = ipc(j)
         i2 = i1 + lenc(j) - 1
         do 610 ii=i1.i2
          if (irn(ii).eq.i) go to 620
 610
          continue
          irm(ii) = irm(i2)
 620
         irn(i2) = 0
         lenc(j) = lenc(j) - 1
         if (nsrch.le.nn) go to 630
c remove column from column chain and place in update chain.
         if (nextc(j).lt.0) go to 630
c jump if column already in update chain.
        lc = lastc(j)
        nc = nextc(j)
        nextc(j) = -colupd
        colupd = j
        if (nc.ne.0) lastc(nc) = lc
        if (lc.eq.0) go to 622
        nextc(lc) = nc
        go to 630
 622
          nz = lenc(j) + 1
        isw = ifirst(nz)
        if (isw.gt:0) lastr(isw) = -nc
        if (isw.lt.0) if irst(nz) = -nc
 630
        continue
       do 640 jj=jnew,iend
        icn(jj) = 0
 640
         contínue
c the value of idrop might be different from that calculated earlier
c because, we may now have dropped some non-zeros which were not
с
   modified by the pivot row.
       idrop = iend + 1 - jnew
       iend = inew - 1
       lenr(i) = lenr(i) - idrop
       nzrow = nzrow - idrop
       nzcol = nzcol - idrop
       ndrop = ndrop + idrop
       ifill = lenpiv - iop
 650
c jump is if there is no fill-in.
```

if (ifill.eq.0) go to 750 c now for the fill-in. minicn = max0(minicn,morei+ibeg-1+nzrow+ifill+lenr(i)) c see if there is room for fill-in. c get maximum space for row i in situ. do 660 jdiff=1,ifill inpos = iend + jdiff if (inpos.gt.licn) go to 670 if (icn(inpos).ne.0) go to 670 660 continue c there is room for all the fill-in after the end of the row so it c can be left in situ. c next available space for fill-in. iend = iend + 1go to 750 c jmore spaces for fill-in are required in front of row. jmore = ifill - jdiff + 1670 il = iptr(i) c we now look in front of the row to see if there is space for c the rest of the fill-in. do 680 jdiff=1,jmore jnpos = il - jdiff if (jnpos.lt.iactiv) go to 690 if (icn(jnpos).ne.0) go to 700 680 continue 690 jnpos = i1 - jmore go to 710 c whole row must be moved to the beginning of available storage. 700 jnpos = iactiv - lenr(i) - ifill c jump if there is space immediately available for the shifted row. if (jnpos.ge.ibeg) go to 730 710 call ma30dd(a, icn, iptr(istart), n, iactiv, itop, .true.) i1 = iptr(i)iend = i1 + lenr(i) - 1inpos = iactiv - lenr(i) - ifillif (jnpos.ge.ibeg) go to 730 c no space available so try to create some by throwing away previous c lu decomposition. morei = morei + ibeg - idisp(1) - lenpiv - 1 if (lp.ne.0) write (lp,99997) iflag = -5 if (abort3) go to 1090 c keep record of current pivot row, ibeg = idisp(1)icn(ibeg) = jpiv a(ibeg) = a(ijpos) ijpos = ibeg do 720 jj=ijp1,pivend ibeg = ibeg + 1a(ibeg) = a(ij)icn(ibeg) = icn(jj) 720 continue ijp1 = ijpos + 1 pivend = ibeg ibeg = ibeg + 1 if (jnpos.ge.ibeg) go to 730 e this still does not give enough room. iflag = -4go to 1090 730 iactiv = min0(iactiv,jnpos) c move non-pivot row i. iptr(i) = inposdo 740 jj=i1,iend a(jnpos) = a(jj)icn(inpos) = icn(jj) jnpos = jnpos + 1 icn(jj) = 0740 continue c first new available space. iend = jnpos 750 nzrow = nzrow + ifill c innermost fill-in loop which also resets icn. idrop = 0

do 830 jj=ijp1,pivend

j = icn(jj)if (j.lt.0) go to 820 anew = au*a(jj) aanew = abs(anew) if (aanew.ge.tol) go to 760 idrop = idrop + 1ndrop = ndrop + 1nzrow = nzrow - 1 minicn = minicn - 1 ifill = ifill - 1 go to 830 760 if (lbig) big = aanew a(iend) = anew icn(iend) = iiend = iend +1с c put new entry in column file. minim = max0(minim,nzcol+lenc(j)+1) jend = ipc(j) + lenc(j)jroom = nzpc - iii + 1 + lenc(j)if (jend.gt.lirn) go to 770 if (irn(jend).eq.0) go to 810 770 if (jroom.lt.dispc) go to 780 c compress column file to obtain space for new copy of column. call ma30dd(a, irn, ipc(istart), n, dispc, lirn, .false.) if (jroom.lt.dispc) go to 780 jroom = dispc - 1 if (jroom.ge.lenc(j)+1) go to 780 c column file is not large enough. go to 1100 c copy column to beginning of file. jbeg = ipc(j)780 jend = ipc(j) + lenc(j) - 1jzero = dispc - 1 dispc = dispc - jroom idispc = dispc do 790 ii=jbeg.jend irm(idispc) = irm(ii)irn(ii) = 0idispc = idispc + 1790 continue ipc(j) = dispcjend = idispc do 800 ii=jend,jzero im(ii) = 0800 continue 810 irm(jend) = inzcol = nzcol + 1lenc(j) = lenc(j) + 1c end of adjustment to column file. go to 830 с 820 icn(jj) = -j830 continue if (idrop.eq.0) go to 834 do 832 kdrop=1,idrop icn(iend) = 0iend = iend + 1832 continue lenr(i) = lenr(i) + ifill834 c end of scan of pivot column. 840 continue с с c remove pivot column from column oriented storage and update row c ordering arrays. il = ipc(jpiv) i2 = ipc(jpiv) + lenc(jpiv) - 1nzcol = nzcol - lenc(jpiv) do 890 ii=i1,i2 i = im(ii)irn(ii) = 0

nz = lenr(i) - lenrl(i)if (nz.ne.0) go to 850 lastr(i) = 0go to 890 ifir = ifirst(nz) 850 ifirst(nz) = i if (ifir) 860, 880, 870 860 lastr(i) = ifir nextr(i) = 0go to 890 870 lastr(i) = lastr(ifir) nextr(i) = ifirlastr(ifir) = igo to 890 880 lastr(i) = 0nextr(i) = 0nzmin = minO(nzmin,nz)890 continue c restore ig and nullify u part of old pivot row. c record the column permutation in lastc(jpiv) and the row c permutation in lastr(ipiv). 900 ipc(jpiv) = -ising lastr(ipiv) = pivot if (lenpiv.eq.0) go to 980 nzrow = nzrow - lenpiv jval = ijp1 izer = iptr(ipiv)iptr(ipiv) = 0do 910 jcount=1,lenpiv j = icn(jval) iq(j) = icn(jzer)icn(jzer) = 0jval = jval + 1 jzer = jzer + 1910 continue c adjust column ordering arrays. if (nsrch.gt.nn) go to 920 do 916 jj=ijp1, pivend j = icn(jj)nz = lenc(j)if (nz.ne.0) go to 914 ipc(j) = 0go to 916 914 nzmin = min0(nzmin,nz) 916 continue go to 980 920 jj = colupd do 970 jdummy=1,nn j=jj if (j.eq.nn+1) go to 980 jj = -nextc(j) nz = lenc(j)if (nz.ne.0) go to 924 ipc(j) = 0go to 970 924 ifir = ifirst(nz) lastc(j) = 0if (ifir) 930, 940, 950 930 ifirst(nz) = -j ifir = -ifir lastc(ifir) = j nextc(j) = ifirgo to 970 940 ifirst(nz) = -j nextc(j) = 0go to 960 950 lc = -lastr(ifir) lastr(ifir) = -j nextc(j) = lcif (lc.ne.0) lastc(lc) = j960 nzmin = minO(nzmin,nz)970 continue 980 continue

с

```
c **** end of main elimination loop ****
с
c reset iactiv to point to the beginning of the next block.
 990 if (ilast.ne.nn) iactiv = iptr(ilast+1)
 1000 continue
с
C ********
c **** end of deomposition of block ****
C *************
с
c record singularity (if any) in iq array.
   if (irank.eq.nn) go to 1020
   do 1010 i=1,nn
    if (ipc(i).lt.0) go to 1010
    ising = ipc(i)
    iq(ising) = -iq(ising)
ipc(i) = -ising
 1010 continue
с
c run through lu decomposition changing column indices to that of new
c order and permuting lenr and lenrl arrays according to pivot
¢
   permutations.
1020 istart = idisp(1)
   iend = ibeg - 1
   if (iend.lt.istart) go to 1040
   do 1030 jj=istart,iend
    jold = icn(jj)
    icn(jj) = -ipc(jold)
 1030 continue
 1040 do 1050 ii=1,nn
    i = lastr(ii)
    nextr(i) = lenr(ii)
    iptr(i) = lenrl(ii)
 1050 continue
   do 1060 i=1,nn
    lenrl(i) = iptr(i)
    lenr(i) = nextr(i)
1060 continue
с
c update permutation arrays ip and iq.
   do 1070 ii=1,nn
    i = lastr(ii)
    i = -ipc(ii)
    nextr(i) = iabs(ip(ii)+0)
    iptr(j) = iabs(iq(ii)+0)
 1070 continue
   do 1080 i=1,nn
    if (ip(i).lt.0) nextr(i) = -nextr<math>(i)
    ip(i) = nextr(i)
    if (iq(i).lt.0) iptr(i) = -iptr(i)
    iq(i) = iptr(i)
1080 continue
   ip(nn) = iabs(ip(nn)+0)
   idisp(2) = iend
   go to 1120
с
c *** error returns ***
1090 idisp(2) = iactiv
   if (lp.eq.0) go to 1120
   write (lp,99996)
   go to 1110
1100 if (iflag.eq.-5) iflag = -6
   if (iflag ne -6) if lag = -3
   idisp(2) = iactiv
   if (lp.eq.0) go to 1120
   if (iflag.eq.-3) write (lp,99995)
   if (iflag.eq.-6) write (lp,99994)
1110 pivot = pivot - istart + 1
   write (lp,99993) pivot, nblock, istart, ilast
   if (pivot.eq.0) write (lp,99992) minim
```

150

с 1120 return 99999 format (54h error return from ma30a/ad because matrix is structur, * 13hally singular) 99998 format (54h error return from ma30a/ad because matrix is numerica, * 12hlly singular) 99997 format (48h lu decomposition destroyed to create more space) 99996 format (54h error return from ma30a/ad because licn not big enoug, * 1hh) 99995 format (54h error return from ma30a/ad because lirn not big enoug, * 1hh) 99994 format (51h error return from ma30a/ad lirn and lien too small) 99993 format (10h at stage, i5, 10h in block, i5, 16h with first row, * i5, 14h and last row , i5) 99992 format (34h to continue set lim to at least, i8) end subroutine ma30dd(a, icn, iptr, n, iactiv, itop, reals) c this subroutine performs garbage collection operations on the c arrays a, icn and irn. c iactiv is the first position in arrays a/icn from which the compress starts. on exit, jactiv equals the position of the first entry Ċ in the compressed part of a/icn с ¢ REAL a(itop) logical reals integer iptr(n) integer icn(itop) c see block data for comments on variables in common. common /ma30fd/ irncp, icncp, irank, minirn, minicn с if (reals) icncp = icncp + 1if (.not.reals) irncp = imcp + 1c set the first non-zero entry in each row to the negative of the c row/col number and hold this row/col index in the row/col pointer. this is so that the beginning of each row/col can с с be recognized in the subsequent scan. do 10 j=1,n k = iptr(j)if (k.lt.iactiv) go to 10 iptr(j) = icn(k)icn(k) = -j10 continue kn = itop + 1kl = itop - iactiv + 1c go through arrays in reverse order compressing to the back so that there are no zeros held in positions factiv to itop in icn. с c reset first entry of each row/col and pointer array iptr. do 30 k=1,kl jpos = itop - k + 1if (icn(jpos).eq.0) go to 30 kn = kn - 1if (reals) a(kn) = a(jpos) if (icn(jpos).ge.0) go to 20 c first non-zero of row/col has been located j = -icn(jpos) icn(jpos) = iptr(j) iptr(j) = kn20 icn(kn) = icn(jpos) 30 continue iactiv = kn return end c#####date 01 jan 1984 copyright ukaea, harwell. c######alias mc13d subroutine mc13d(n,icn,licn,ip,lenr,ior,ib,num,iw) integer ip(n) integer icn(licn),lenr(n),ior(n),ib(n),iw(n,3) call mc13e(n,icn,licn,ip,lenr,ior,ib,num,iw(1,1),iw(1,2),iw(1,3)) return end subroutine mc13e(n,icn,licn,ip,lenr,arp,ib,num,lowl,numb,prev) integer stp,dummy integer ip(n)

```
с
c arp(i) is one less than the number of unsearched edges leaving
с
   node i. at the end of the algorithm it is set to a
с
    permutation which puts the matrix in block lower
c triangular form.
c ib(i) is the position in the ordering of the start of the ith
c block. ib(n+1-i) holds the node number of the ith node
    on the stack.
с
c \; lowl(i) is the smallest stack position of any node to which a path
c from node i has been found. it is set to n+1 when node i
с
    is removed from the stack.
c numb(i) is the position of node i in the stack if it is on
c it, is the permuted order of node i for those nodes
c whose final position has been found and is otherwise zero.
c prev(i) is the node at the end of the path when node i was
c placed on the stack.
    integer icn(licn),lenr(n),arp(n),ib(n),lowl(n),numb(n),
   lprev(n)
с
c
c icnt is the number of nodes whose positions in final ordering have
¢
    been found.
    icnt=0
c num is the number of blocks that have been found.
    num=0
   nnml=n+n-l
с
c initialization of arrays.
    do 20 j=1,n
    numb(j)=0
   arp(i)=lenr(j)-1
  20 continue
с
с
    do 120 isn=1,n
c look for a starting node
    if (numb(isn).ne.0) go to 120
    iv=isn
c ist is the number of nodes on the stack ... it is the stack pointer.
    ist=1
c put node iv at beginning of stack.
    lowl(iv)=1
    numb(iv)=1
    ib(n)=iv
с
c the body of this loop puts a new node on the stack or backtracks.
    do 110 dummy=1,nnm1
    il=arp(iv)
c have all edges leaving node iv been searched.
    if (i1.lt.0) go to 60
    i2=ip(iv)+lenr(iv)-1
    i1=i2-i1
¢
c look at edges leaving node iv until one enters a new node or
c all edges are exhausted.
    do 50 ii=i1,i2
    iw=icn(ii)
c has node iw been on stack already.
    if (numb(iw).eq.0) go to 100
c update value of lowl(iv) if necessary.
 50 lowl(iv)=min0(lowl(iv),lowl(iw))
c
c there are no more edges leaving node iv.
    arp(iv)=-1
c is node iv the root of a block.
  60 if (lowl(iv).lt.numb(iv)) go to 90
c
c order nodes in a block.
    num=num+1
    ist1=n+1-ist
    lcnt=icnt+1
```

c peel block off the top of the stack starting at the top and

c working down to the root of the block.

do 70 stp=ist1,n iw=ib(stp) lowl(iw)=n+1 icnt=icnt+1 numb(iw)=icnt if (iw.eq.iv) go to 80 70 continue 80 ist=n-stp ib(num)=lent c are there any nodes left on the stack. if (ist.ne.0) go to 90 c have all the nodes been ordered. if (icnt.lt.n) go to 120 go to 130 ¢ c backtrack to previous node on path. 90 iw=iv iv=prev(iv) c update value of lowl(iv) if necessary. lowl(iv)=min0(lowl(iv),lowl(iw)) go to 110 ¢ c put new node on the stack. 100 arp(iv)=i2-ii-1 prev(iw)=iv iv≕iw ist=ist+1 lowl(iv)=ist numb(iv)=ist k=n+1-ist ib(k)=iv 110 continue ¢ 120 continue с c c put permutation in the required form. 130 do 140 i=1,n ii=numb(i) 140 arp(ii)-i return end c#####date 01 jan 1984 copyright ukaea, harwell. c######alias mc20ad mc20bd subroutine mc20ad(nc,maxa,a,inum,jptr,jnum,jdisp) ¢ integer inum(maxa),jnum(maxa) REAL a(maxa),ace,acep dimension jptr(nc) с *********** ¢ с null—jdisp c** clear jptr do 60 j=1,nc 60 jptr(j)=0 count the number of elements in each column. c** do 120 k=1,maxa j=jnum(k)+jdisp jptr(j)=jptr(j)+1 120 continue C** set the jptr array k=1 do 150 j=1,nc kr=k+jptr(j) jptr(j)=k 150 k=kr с c** reorder the elements into column order. the algorithm is an in-place sort and is of order maxa. с do 230 i=1,maxa establish the current entry. ¢ jce=jnum(i)+jdisp

```
if(jce.eq.0) go to 230
    ace=a(i)
   ice=inum(i)
     clear the location vacated.
с
   jnum(i)=null
     chain from current entry to store items.
с
   do 200 j=1,maxa
с
     current entry not in correct position. determine correct
     position to store entry.
с
    loc=jptr(jce)
   jptr(jce)=jptr(jce)+1
с
     save contents of that location.
    acep=a(loc)
    icep=inum(loc)
   jcep=jnum(loc)
     store current entry.
с
    a(loc)=ace
    inum(loc)=ice
   jnum(loc)=null
     check if next current entry needs to be processed.
¢
   if(jcep.eq.null) go to 230
с
     it does. copy into current entry.
    ace=acep
   ice=icep
 200 jce=jcep+jdisp
с
 230 continue
с
c**
      reset jptr vector.
   ja=1
   do 250 i=1.nc
   jb=jptr(j)
   jptr(j)=ja
 250 ja=jb
   return
   end
c#####date 01 jan 1984 copyright ukaea, harwell.
c######alias mc21a
   subroutine mc21a(n,icn,licn,ip,lenr,iperm,numnz,iw)
   integer ip(n)
   integer icn(licn),lenr(n),iperm(n),iw(n,4)
   call mc21b(n,icn,licn,ip,lenr,iperm,numnz,iw(1,1),iw(1,2),iw(1,3),
   1iw(1,4)
   return
   end
   subroutine mc21b(n,icn,licn,ip,lenr,iperm,numnz,pr,arp,cv,out)
   integer ip(n)
c pr(i) is the previous row to i in the depth first search.
c it is used as a work array in the sorting algorithm.
c elements (iperm(i),i) = 1, ... n are non-zero at the end of the
c algorithm unless n assignments have not been made. in which case
c (iperm(i),i) will be zero for n-numnz entries.
c cv(i) is the most recent row extension at which column i
c was visited.
c arp(i) is one less than the number of non-zeros in row i
c which have not been scanned when looking for a cheap assignment.
c out(i) is one less than the number of non-zeros in row i
c which have not been scanned during one pass through the main loop.
   integer icn(licn),lenr(n),iperm(n),pr(n),cv(n),
   larp(n),out(n)
с
c initialization of arrays.
   do 10 i=1.n
   arp(i)=lenr(i)-1
   cv(i)=0
  10 iperm(i)=0
   numnz=0
¢
Ċ
c main loop.
c each pass round this loop either results in a new assignment
c or gives a row with no assignment.
   do 130 jord=1,n
```

154

j=jord pr(j)=-1 do 100 k=1.jord c look for a cheap assignment in1=arp(j) if (in1.lt.0) go to 60 in2=ip(j)+lenr(j)-1 in1=in2-in1 do 50 ii=in1,in2 i=icn(ii) if (iperm(i).eq.0) go to 110 50 continue c no cheap assignment in row. arp(j)=-1 c begin looking for assignment chain starting with row j. 60 out(j)=lenr(j)-1 c inner loop. extends chain by one or backtracks. do 90 kk=1,jord in1=out(j) if (in 1.lt.0) go to 80 in2=ip(j)+lenr(j)-1 in1=in2-in1 c forward scan. do 70 ii=in1,in2 i=icn(ii) if (cv(i).eq.jord) go to 70 c column i has not yet been accessed during this pass. j1≕j j=iperm(i) cv(i)=jord pr(j)=j1 out(j1)=in2-ii-1 go to 100 70 continue с c backtracking step. 80 j=pr(j) if (j.eq.-1) go to 130 90 continue с 100 continue с c new assignment is made. 110 iperm(i)=j arp(j)=in2-ii-1 numnz-numnz+1 do 120 k=1,jord j=pr(j) if (j.eq.-1) go to 130 ii=ip(j)+lenr(j)-out(j)-2 i=icn(ii) iperm(i)=j 120 continue c 130 continue ¢ c if matrix is structurally singular, we now complete the c permutation iperm. if (numnz.eq.n) return do 140 i=1,n 140 arp(i)=0 k=0 do 160 i=1,n if (iperm(i).ne.0) go to 150 k=k+1out(k)=i go to 160 150 j=iperm(i) arp(j)=i 160 continue k=0 do 170 i=1,n if (arp(i).ne.0) go to 170

```
k=k+1
    ioutk=out(k)
    iperm(ioutk)=i
 170 continue
    return
    end
c#####date 01 jan 1984 copyright ukaea, harwell.
c######alias mc22ad
    subroutine mc22ad(n,icn,a,nz,lenrow,ip,iq,iw,iw1)
    REAL a(nz),aval
    integer iw(n,2)
    integer icn(nz),lenrow(n),ip(n),iq(n),iw1(nz)
    if (nz.le.0) go to 1000
    if (n.le.0) go to 1000
c set start of row i in iw(i,1) and lenrow(i) in iw(i,2)
    iw(1,1)=1
    iw(1,2)=lenrow(1)
    do 10 i=2,n
    iw(i,1)=iw(i-1,1)+lenrow(i-1)
 10 iw(i,2)=lenrow(i)
c permute lenrow according to ip. set off-sets for new position
c of row iold in iw(iold,1) and put old row indices in iw1 in
с
   positions corresponding to the new position of this row in a/icn.
    jj=1
    do 20 i=1,n
    iold=ip(i)
    iold=iabs(iold)
    length=iw(iold,2)
    lenrow(i)=length
    if (length.eq.0) go to 20
    iw(iold,1)=iw(iold,1)-jj
    j2=jj+length-1
    do 15 j=jj,j2
 15 iwl(j)=iold
    jj=j2+1
 20 continue
c set inverse permutation to iq in iw(.,2).
    do 30 i=1,n
    iold=iq(i)
    iold=iabs(iold)
 30 iw(iold,2)=i
c permute a and icn in place, changing to new column numbers.
с
c *** main loop ***
c each pass through this loop places a closed chain of column indices
   in their new (and final) positions ... this is recorded by
с
c setting the iwl entry to zero so that any which are subsequently
c encountered during this major scan can be bypassed.
    do 200 i=1,nz
    iold=iw1(i)
    if (iold.eq.0) go to 200
    ipos=i
    jval=icn(i)
c if row iold is in same positions after permutation go to 150.
    if (iw(iold,1).eq.0) go to 150
    aval=a(i)
c ** chain loop **
c each pass through this loop places one (permuted) column index
    in its final position .. viz. ipos.
с
    do 100 ichain=1.nz
c newpos is the original position in a/icn of the element to be placed
c in position ipos. it is also the position of the next element in
   the chain.
¢
    newpos=ipos+iw(iold,1)
c is chain complete ?
    if (newpos.eq.i) go to 130
    a(ipos)=a(newpos)
    jnum=icn(newpos)
    icn(ipos)=iw(jnum,2)
    ipos=newpos
    iold=iw1(ipos)
    iw1(ipos)=0
c ** end of chain loop **
```

100 continue 130 a(ipos)=aval 150 icn(ipos)=iw(jval,2) c *** end of main loop *** 200 continue ¢ 1000 return end c#####date 01 jan 1984 copyright ukaea, harwell. c######alias mc23ad c###### calls mc13 mc21 subroutine mc23ad(n,icn,a,licn,lenr,idisp,ip,iq,lenoff,iw,iw1) REAL a(licn) integer idisp(2), iw1(n,2) logical abort integer icn(licn),lenr(n),ip(n),iq(n),lenoff(n),iw(n,5) common /mc23bd/ lp,numnz,num,large,abort c input ... n,ien .. a,ien,lenr ¢ c set up pointers iw(.,1) to the beginning of the rows and set lenoff ¢ equal to lenr. iwl(1,1)=1lenoff(1)=lenr(1) if (n.eq.1) go to 20 do 10 i=2,n lenoff(i)=lenr(i) 10 iw1(i,1)=iw1(i-1,1)+lenr(i-1)c idisp(1) points to the first position in a/icn after the off-diagonal blocks and untreated rows. с 20 idisp(1)=iw1(n,1)+lenr(n)с c find row permutation ip to make diagonal zero-free. call mc21a(n,icn,licn,iw1,lenr,ip,numnz,iw) c c possible error return for structurally singular matrices. if (numnz.ne.n.and.abort) go to 170 ¢ c iw1(.,2) and lenr are permutations of iw1(.,1) and lenr/lenoff suitable for entry С to mc13d since matrix with these row pointer and length arrays с has maximum number of non-zeros on the diagonal. с do 30 ii=1,n i=ip(ii) iw1(ii,2)=iw1(i,1)30 lenr(ii)=lenoff(i) с c find symmetric permutation iq to block lower triangular form. call mc13d(n,icn,licn,iw1(1,2),lenr,iq,iw(1,4),num,iw) с if (num.ne.1) go to 60 с c action taken if matrix is irreducible. c whole matrix is just moved to the end of the storage. do 40 i=1,n lenr(i)=lenoff(i) ip(i)=i 40 iq(i)=i lenoff(1)=-1c idisp(1) is the first position after the last element in the off-diagonal blocks and untreated rows. С nz=idisp(1)-1 idisp(1)=1c idisp(2) is the position in a/icn of the first element in the diagonal blocks. c idisp(2)=licn-nz+1 large=n if (nz.eq.licn) go to 230 do 50 k=1,nz j=nz-k+1 jj=licn-k+1 a(jj)=a(j)50 icn(jj)=icn(j) c 230 = return

```
go to 230
с
c data structure reordered.
с
c form composite row permutation ... ip(i) = ip(iq(i)).
  60 do 70 ii=1,n
    i=iq(ii)
  70 iw(ii,1)=ip(i)
    do 80 i=1,n
  80 ip(i)=iw(i,1)
¢
c run through blocks in reverse order separating diagonal blocks
с
    which are moved to the end of the storage. elements in
    off-diagonal blocks are left in place unless a compress is
с
    necessary.
с
c
c ibeg indicates the lowest value of j for which icn(j) has been
    set to zero when element in position j was moved to the
с
    diagonal block part of storage.
с
    ibeg=licn+1
c iend is the position of the first element of those treated rows
   which are in diagonal blocks.
с
    iend=licn+1
c large is the dimension of the largest block encountered so far.
    large=0
с
c num is the number of diagonal blocks.
    do 150 k=1,num
    iblock=num-k+1
c il is first row (in permuted form) of block iblock.
c i2 is last row (in permuted form) of block iblock.
    il=iw(iblock,4)
    i2=n
    if (k.ne.1) i2=iw(iblock+1,4)-1
    large=max0(large,i2-i1+1)
c go through the rows of block iblock in the reverse order.
    do 140 ii=i1.i2
    inew=i2-ii+i1
c we now deal with row inew in permuted form (row iold in original
   matrix).
с
    iold=ip(inew)
c if there is space to move up diagonal block portion of row go to 110
    if (iend-idisp(1) ge.lenoff(iold)) go to 110
с
c in-line compress.
c moves separated off-diagonal elements and untreated rows to
   front of storage.
с
   jnpos=ibeg
    ilend=idisp(1)-1
    if (ilend.lt.ibeg) go to 190
    do 90 j=ibeg,ilend
    if (icn(j).eq.0) go to 90
    icn(inpos)=icn(i)
   a(inpos)=a(j)
   inpos=inpos+1
 90 continue
    idisp(1)=jnpos
   if (iend-jnpos.lt.lenoff(iold)) go to 190
   ibeg=licn+1
c reset pointers to the beginning of the rows.
   do 100 i=2.n
 100 iw1(i,1)=iw1(i-1,1)+lenoff(i-1)
с
c row iold is now split into diag, and off-diag, parts.
 110 irowb=iw1(iold,1)
   leni=0
   irowe=irowb+lenoff(iold)-1
c backward scan of whole of row iold (in original matrix).
   if (irowe.lt.irowb) go to 130
   do 120 ij=irowb irowe
   j=irowe-jj+irowb
   jold=icn(j)
c iw(.,2) holds the inverse permutation to iq.
```

159

..... it was set to this in mc13d. с jnew=iw(jold,2) c if (jnew.lt.il) then c element is in off-diagonal block and so is left in situ. if (jnew.lt.i1) go to 120 c element is in diagonal block and is moved to the end of the storage. iend=iend-1 a(iend)=a(j) icn(iend)=jnew ibeg=min0(ibeg.j) icn(j)=0 leni=leni+1 120 continue с lenoff(iold)=lenoff(iold)-leni 130 lenr(inew)=leni 140 continue c ip(i2)=-ip(i2)150 continue c resets ip(n) to positive value. ip(n) = -ip(n)c idisp(2) is position of first element in diagonal blocks. idisp(2)=iend С c this compress is used to move all off-diagonal elements to the c front of the storage. if (ibeg.gt.licn) go to 230 inpos=ibeg ilend=idisp(1)-1 do 160 j=ibeg,ilend if (icn(j).eq.0) go to 160 icn(jnpos)=icn(j) a(jnpos)=a(j) jnpos=jnpos+1 160 continue c idisp(1) is first position after last element of off-diagonal blocks. idisp(1)=jnpos go to 230 с c c error return 170 if (lp.ne.0) write(lp,180) numnz 180 format(33x,41h matrix is structurally singular, rank = ,i6) idisp(1)=1go to 210 190 if (lp.ne.0) write(lp,200) n 200 format(33x,33h licn not big enough increase by ,i6) idisp(1)=-2210 if (lp.ne.0) write(lp,220) 220 format(33h+error return from mc23ad because) ¢ 230 return end c######date 01 jan 1984 copyright ukaea, harwell. c######alias mc24ad subroutine mc24ad(n,icn,a,licn,lenr,lenrl,w) REAL a(licn),w(n),amaxl,wrowl,amaxu,zero integer icn(licn),lenr(n),lenrl(n) data zero/0.0d0/ amaxl=zero do 10 i=1,n 10 w(i)=zero j0=1 do 100 i=1,n if (lenr(i).eq.0) go to 100 j2=j0+lenr(i)-1 if (lenrl(i).eq.0) go to 50 c calculation of 1-norm of l. jl=j0+lenrl(i)-lwrowl=zero do 30 jj=j0,j1 30 wrowl=wrowl+abs(a(jj))

c amaxl is the maximum norm of columns of l so far found. amaxi=wrowl i0=j1+1 c calculation of norms of columns of u (max-norms). 50 i0=i0+1 if (j0.gt.j2) go to 90 do 80 jj=j0,j2 j=icn(jj) 80 w(j)=abs(a(jj)) 90 j0=j2+1 100 continue c amaxu is set to maximum max-norm of columns of u. amaxu=zero do 200 i=1.n 200 amaxu=w(i) c grofac is max u max-norm times max 1 1-norm. w(1)=amaxi*amaxu return end subroutine ma28cd(n, a, licn, icn, ikeep, rhs, w, mtype) с c this subroutine uses the factors from ma28a/ad or ma28b/bd to c solve a system of equations without iterative refinement. c the parameters are ... c n integer order of matrix not altered by subroutine. c a real/double precision array length licn. the same array as c was used in the most recent call to ma28a/ad or ma28b/bd. c licn integer length of arrays a and icn. not altered by c subroutine. c icn integer array of length licn, same array as output from c ma28a/ad. unchanged by ma28c/cd. c ikeep integer array of length 5*n. same array as output from c ma28a/ad. unchanged by ma28c/cd. c rhs real/double precision array length n. on entry, it holds the c right hand side. on exit, the solution vector. c w real/double precision array length n. used as workspace by c ma30c/cd. c mtype integer used to tell ma30c/cd to solve the direct equation (mtype=1) or its transpose (mtype.ne.1). с ¢ REAL a(licn), rhs(n), w(n), resid, mresid, eps, rmin integer idisp(2) integer icn(licn), ikeep(n,5) logical abort1, abort2 c common block variables. c unless otherwise stated common block variables are as in ma28a/ad. c those variables referenced by ma28c/cd are mentioned below. c resid real/double precision variable returns maximum residual of c equations where pivot was zero. c mresid real/double precision variable used by ma28c/cd to communicate between ma28f/fd and ma30h/hd. С c idisp integer array length 2 the same as that used by ma28a/ad. c it is unchanged by ma28b/bd. Ċ c further information on common block variables can be found in block data or ma28a/ad. с common /ma28fd/ eps, rmin, resid, irncp, icncp, minirn, minicn, * irank, abort1, abort2 common /ma28gd/ idisp common /ma30hd/ mresid с c this call performs the solution of the set of equations. call ma30cd(n, icn, a, licn, ikeep, ikeep(1,4), ikeep(1,5), * idisp, ikeep(1,2), ikeep(1,3), rhs, w, mtype) c transfer common block information. resid = mresid return end subroutine ma30cd(n, icn, a, licn, lenr, lenrl, lenoff, idisp, ip, * iq, x, w, mtype) c ma30c/cd uses the factors produced by ma30a/ad or ma30b/bd to solve c ax=b or a transpose x=b when the matrix pl*a*ql (paq) is block

- с lower triangular (including the case of only one diagonal block).
- С c
- c we now describe the argument list for ma30c/cd.
- c n is an integer variable set to the order of the matrix, it is not
- c altered by the subroutine.
- c icn is an integer array of length licn. entries idisp(1) to
- idisp(2) should be unchanged since the last call to ma30a/ad. if С
- the matrix has more than one diagonal block, then column indices с
- corresponding to non-zeros in sub-diagonal blocks of pag must с
- appear in positions 1 to idisp(1)-1. for the same row those с
- entries must be contiguous, with those in row i preceding those с
- in row i+1 (i=1,...,n-1) and no wasted space between rows. С
- entries may be in any order within each row. it is not altered с
- c by ma30c/cd.
- c a is a real/double precision array of length licn. entries
- c idisp(1) to idisp(2) should be unchanged since the last call to
- ma30a/ad or ma30b/bd. if the matrix has more than one diagonal c
- c block, then the values of the non-zeros in sub-diagonal blocks
- c must be in positions 1 to idisp(1)-1 in the order given by icn.
- it is not altered by ma30c/cd. с

c licn is an integer variable set to the size of arrays icn and a. c it is not altered by ma30c/cd.

- c lenr, lenrl are integer arrays of length n which should be
- c unchanged since the last call to ma30a/ad. they are not altered c by ma30c/cd.

c lenoff is an integer array of length n. if the matrix paq (or

c p1*a*q1) has more than one diagonal block, then lenoff(i),

- i=1,...,n should be set to the number of non-zeros in row i of С
- the matrix paq which are in sub-diagonal blocks. if there is c
- only one diagonal block then lenoff(1) may be set to -1, in с
- which case the other entries of lenoff are never accessed. it is С not altered by ma30c/cd. с
- c idisp is an integer array of length 2 which should be unchanged c since the last call to ma30a/ad. it is not altered by ma30c/cd.
- c ip,iq are integer arrays of length n which should be unchanged
- c since the last call to ma30a/ad. they are not altered by
- c ma30c/cd.
- c x is a real/double precision array of length n. it must be set by
- c the user to the values of the right hand side vector b for the
- c equations being solved. on exit from ma30c/cd it will be equal c to the solution x required.
- c w is a real/double precision array of length n which is used as c workspace by ma30c/cd.
- c mtype is an integer variable which must be set by the user. if
- mtype=1, then the solution to the system ax=b is returned; any С
- other value for mtype will return the solution to the system a ¢
- transpose x=b. it is not altered by ma30c/cd. С

REAL a(licn), x(n), w(n), wii, wi, resid, zero

- logical neg, nobloc
- integer idisp(2)

integer icn(licn), lenr(n), lenrl(n), lenoff(n), ip(n), iq(n) c see block data for comments on variables in common.

- common /ma30hd/ resid
- data zero /0.0/
- Ċ

c the final value of resid is the maximum residual for an inconsistent set of equations. С

- resid = zero
- c nobloc is .true. if subroutine block has been used previously and
- c is false, otherwise, the value false, means that lenoff
- will not be subsequently accessed. с
 - nobloc = lenoff(1).lt.0if (mtype.ne.1) go to 140
- с

c we now solve a * x = b.

- c neg is used to indicate when the last row in a block has been
- c reached, it is then set to true whereafter backsubstitution is
- с performed on the block. neg = .false.
- c ip(n) is negated so that the last row of the last block can be
- c recognised, it is reset to its positive value on exit.

ip(n) = -ip(n)c preorder vector ... w(i) = x(ip(i))do 10 ii=1,n i = ip(ii)i = iabs(i)w(ii) = x(i)10 continue c lt holds the position of the first non-zero in the current row of the с off-diagonal blocks. lt = 1 c ifirst holds the index of the first row in the current block. ifirst = 1c iblock holds the position of the first non-zero in the current row of the lu decomposition of the diagonal blocks. с iblock = idisp(1)c if i is not the last row of a block, then a pass through this loop adds the inner product of row i of the off-diagonal blocks and w ¢ с to w and performs forward elimination using row i of the lu decomposition. if i is the last row of a block then, after с c performing these aforementioned operations, backsubstitution is ¢ performed using the rows of the block. do 120 i=1,n wi = w(i)if (nobloc) go to 30 if (lenoff(i).eq.0) go to 30 c operations using lower triangular blocks. c ltend is the end of row i in the off-diagonal blocks. ltend = lt + lenoff(i) - 1do 20 jj=lt,ltend j = icn(jj)wi = wi - a(jj) * w(j)20 continue c It is set the beginning of the next off-diagonal row. lt = ltend + 1c set neg to .true. if we are on the last row of the block. 30 if (ip(i).lt.0) neg = .true. if (lenrl(i).eq.0) go to 50 c forward elimination phase. c iend is the end of the I part of row i in the lu decomposition. iend = iblock + lenrl(i) - 1do 40 jj=iblock,iend j = icn(jj)wi = wi + a(jj) * w(j)40 continue c iblock is adjusted to point to the start of the next row. 50 iblock = iblock + lenr(i) w(i) = wiif (.not.neg) go to 120 c back substitution phase. c j1 is position in a/icn after end of block beginning in row ifirst and ending in row i. j1 = iblockc are there any singularities in this block? if not, continue with c the backsubstitution. ib = i if (iq(i).gt.0) go to 70 do 60 iii=ifirst,i ib = i - iii + ifirst if (iq(ib).gt.0) go to 70 j1 = j1 - lenr(ib)resid = abs(w(ib))w(ib) = zero60 continue c entire block is singular. go to 110 c each pass through this loop performs the back-substitution c operations for a single row, starting at the end of the block and working through it in reverse order. с 70 do 100 iii=ifirst,ib ii = ib - iii + ifirst c j2 is end of row ii. i2 = i1 - 1

c j1 is beginning of row ii.

```
j1 = j1 - lenr(ii)
c jpiv is the position of the pivot in row ii.
      jpiv = j1 + lenrl(ii)
      jpivp1 = jpiv + 1
c jump if row ii of u has no non-zeros.
      if (j2.lt.jpivp1) go to 90
      wii = w(ii)
      do 80 jj=jpivp1 j2
       j = icn(jj)
       wii = wii - a(jj)*w(j)
 80
      continue
      w(ii) = wii
 90 w(ii) = w(ii)/a(jpiv)
 100 continue
 110 ifirst = i + 1
     neg = .false.
 120 continue
с
c reorder solution vector ... x(i) = w(iginverse(i))
    do 130 ii=1,n
     i = iq(ii)
     i = iabs(i)
     x(i) = w(ii)
 130 continue
    ip(n) = -ip(n)
   go to 320
с
С
c we now solve atranspose *x = b.
c preorder vector ... w(i)=x(iq(i))
 140 do 150 ii=1,n
     i = iq(ii)
     i = iabs(i)
     w(ii) = x(i)
 150 continue
c lj1 points to the beginning the current row in the off-diagonal
    blocks.
С
    lil = idisp(1)
c iblock is initialized to point to the beginning of the block after
    the last one ]
с
    iblock = idisp(2) + 1
c ilast is the last row in the current block.
    ilast = n
c iblend points to the position after the last non-zero in the
    current block.
¢
    iblend = iblock
c each pass through this loop operates with one diagonal block and
c the off-diagonal part of the matrix corresponding to the rows
    of this block. the blocks are taken in reverse order and the
¢
   number of times the loop is entered is min(n,no. blocks+1).
С
    do 290 numblk=1,n
     if (ilast.eq.0) go to 300
     iblock = iblock - lenr(ilast)
c this loop finds the index of the first row in the current block.
c it is first and iblock is set to the position of the beginning
¢
    of this first row.
     do 160 k=1,n
      ii = ilast - k
      if (ii.eq.0) go to 170
      if (ip(ii).lt.0) go to 170
       iblock = iblock - lenr(ii)
 160 continue
 170 if if = ii + 1
c j1 points to the position of the beginning of row i (lt part) or pivot
     jI = iblock
c forward elimination.
c each pass through this loop performs the operations for one row of the
c block. if the corresponding entry of w is zero then the
     operations can be avoided.
¢
      do 210 i=ifirst,ilast
       if (w(i).eq.zero) go to 200
c jump if row i singular.
       if (iq(i).lt.0) go to 220
```

c j2 first points to the pivot in row i and then is made to point to the first non-zero in the u transpose part of the row. с j2 = j1 + lenrl(i)wi = w(i)/a(j2)if (lenr(i)-lenrl(i).eq.1) go to 190 j2 = j2 + 1c j3 points to the end of row i. j3 = j1 + lenr(i) - 1do 180 jj=j2,j3 j = icn(jj)w(j) = w(j) - a(jj)*wi180 continue w(i) = wi190 200 j1 = j1 + lenr(i)210 continue go to 240 c deals with rest of block which is singular. 220 do 230 ii=i,ilast resid = abs(w(ii)) w(ii) = zero 230 continue c back substitution. c this loop does the back substitution on the rows of the block in c the reverse order doing it simultaneously on the l transpose part of the diagonal blocks and the off-diagonal blocks. ¢ 240 jl = iblend do 280 iback=ifirst_ilast i = ilast - iback + ifirst c j1 points to the beginning of row i. j1 = j1 - lenr(i)if (lenrl(i).eq.0) go to 260 c j2 points to the end of the l transpose part of row i. j2 = j1 + lenrl(i) - 1do 250 jj=j1,j2 j = icn(jj)w(j) = w(j) + a(jj) * w(i)250 continue 260 if (nobloc) go to 280 c operations using lower triangular blocks. if (lenoff(i).eq.0) go to 280 c lj2 points to the end of row i of the off-diagonal blocks. li2 = li1 - 1c lj1 points to the beginning of row i of the off-diagonal blocks. ij1 = ij1 - ienoff(i)do 270 jj=lj1,lj2 j = icn(jj)w(j) = w(j) - a(jj) * w(i)270 continue 280 continue iblend = j1ilast = ifirst - 1 290 continue c reorder solution vector ... x(i)=w(ipinverse(i)) 300 do 310 ii=1,n i = ip(ii)i = iabs(i) $\mathbf{x}(\mathbf{i}) = \mathbf{w}(\mathbf{i}\mathbf{i})$ 310 continue с 320 return end block data ma28jd ¢ c comments on all the common block variables are given here even

c though some are not initialized by block data.

c lp,mp are used by the subroutine as the unit numbers for its warning

c and diagnostic messages default value for both is 6 (for line

c printer output), the user can either reset them to a different

c stream number or suppress the output by setting them to zero.

c while lp directs the output of error diagnostics from the

c principal subroutines and internally called subroutines, mp

c controls only the output of a message which warns the user that he

- c has input two or more non-zeros a(i), ..., a(k) with the same row
- and column indices. the action taken in this case is to proceed с
- с using a numerical value of a(i)+...+a(k). in the absence of other
- errors, iflag will equal -14 on exit.
- c lblock is a logical variable which controls an option of first preordering the matrix to block lower triangular form (using с
- harwell subroutine mc23a), the preordering is performed if lblock ¢
- с is equal to its default value of .true. if lblock is set to
- .false., the option is not invoked and the space allocated to с
- ikeep can be reduced to 4*n+1. c
- c grow is a logical variable. if it is left at its default value of
- с .true., then on return from ma28a/ad or ma28b/bd, w(1) will give
- С an estimate (an upper bound) of the increase in size of elements
- encountered during the decomposition. if the matrix is well с
- ¢ scaled, then a high value for w(1), relative to the largest entry
- in the input matrix, indicates that the lu decomposition may be с
- inaccurate and the user should be wary of his results and perhaps с с
- increase u for subsequent runs. we would like to emphasise that ¢
- this value only relates to the accuracy of our lu decomposition с and gives no indication as to the singularity of the matrix or the
- С accuracy of the solution. this upper bound can be a significant
- overestimate particularly if the matrix is badly scaled. if an ¢
- accurate value for the growth is required, lbig (q.v.) should be с
- ¢ set to .true.

c eps,rmin are real variables. if, on entry to ma28b/bd, eps is less

- than one, then rmin will give the smallest ratio of the pivot to c
- с the largest element in the corresponding row of the upper
- triangular factor thus monitoring the stability of successive с
- factorizations. if min becomes very large and w(1) from с
- ma28b/bd is also very large, it may be advisable to perform a с
- c new decomposition using ma28a/ad.
- c resid is a real variable which on exit from ma28c/cd gives the value
- of the maximum residual over all the equations unsatisfied because of dependency (zero pivots). с
- c irnep,iencp are integer variables which monitor the adequacy of "elbow
- room" in irn and a/icn respectively. if either is quite large (say с
- greater than n/10), it will probably pay to increase the size of c
- the corresponding array for subsequent runs. if either is very low с
- с or zero then one can perhaps save storage by reducing the size of с the corresponding array.
- c minim, minicn are integer variables which, in the event of a
- successful return (iflag ge 0 or iflag=-14) give the minimum size Ċ
- of irn and a/icn respectively which would enable a successful run с
- с on an identical matrix. on an exit with iflag equal to -5, minicn
- gives the minimum value of icn for success on subsequent runs on Ċ
- an identical matrix. in the event of failure with iflag= -6, -4, с
- с -3, -2, or -1, then minicn and minirn give the minimum value of
- licn and lirn respectively which would be required for a с
- successful decomposition up to the point at which the failure с
- occurred. с

c irank is an integer variable which gives an upper bound on the rank of the matrix. С

- c abort1 is a logical variable with default value .true. if abort1 is
- set to .false. then ma28a/ad will decompose structurally singular matrices (including rectangular ones). С
- c abort2 is a logical variable with default value true. if abort2 is set to .false, then ma28a/ad will decompose numerically singular с
- С matrices.
- c idisp is an integer array of length 2. on output from ma28a/ad, the
- c indices of the diagonal blocks of the factors lie in positions
- idisp(1) to idisp(2) of a/icn. this array must be preserved C
- ç between a call to ma28a/ad and subsequent calls to ma28b/bd,
- с ma28c/cd or ma28i/id.
- c tol is a real variable. if it is set to a positive value, then any
- c non-zero whose modulus is less than tol will be dropped from the
- с factorization. the factorization will then require less storage
- с but will be inaccurate. after a run of ma28a/ad with tol positive
- it is not possible to use ma28b/bd and the user is recommended to С
- use ma28i/id to obtain the solution. the default value for tol is С С 0.0.
- c themax is a real variable. on exit from ma28a/ad, it will hold the c largest entry of the original matrix.
- c big is a real variable, if lbig has been set to true, big will hold

c the largest entry encountered during the factorization by ma28a/ad c or ma28b/bd. c dxmax is a real variable. on exit from ma28i/id, dxmax will be set to c the largest component of the solution. c errmax is a real variable. on exit from ma28i/id, if maxit is c positive, errmax will be set to the largest component in the c estimate of the error. c dres is a real variable. on exit from ma28i/id, if maxit is positive, c dres will be set to the largest component of the residual. c cgce is a real variable, it is used by ma28i/id to check the c convergence rate. if the ratio of successive corrections is c not less than egee then we terminate since the convergence с rate is adjudged too slow. c ndrop is an integer variable. if tol has been set positive, on exit from ma28a/ad, ndrop will hold the number of entries dropped from c the data structure. c maxit is an integer variable. it is the maximum number of iterations performed by ma28i/id, it has a default value of 16. с c notter is an integer variable. it is set by ma28i/id to the number of c iterative refinement iterations actually used. c nsrch is an integer variable. if nsrch is set to a value less than n, then a different pivot option will be employed by ma28a/ad. this с may result in different fill-in and execution time for ma28a/ad. с if nsrch is less than or equal to n, the workspace array iw can be с reduced in length. the default value for nsrch is 32768. с c istart is an integer variable. if istart is set to a value other than c zero, then the user must supply an estimate of the solution to ma28i/id, the default value for istart is zero. с c lbig is a logical variable. if lbig is set to .true., the value of the c largest element encountered in the factorization by ma28a/ad or ma28b/bd is returned in big. setting lbig to .true. will ¢ с increase the time for ma28a/ad marginally and that for ma28b/bd с by about 20%. the default value for lbig is .false. с REAL eps, rmin, resid, tol, themax, big, dxmax,dmax1,dmin1,abs, * errmax, dres, cgce logical lblock, grow, abort1, abort2, lbig common /ma28ed/ lp, mp, lblock, grow common /ma28fd/ eps, rmin, resid, irncp, icncp, minirn, minicn, * irank, abort1, abort2 common /ma28gd/ idisp(2) common /ma28hd/ tol, themax, big, errmax, dres, cgce, * ndrop, maxit, noiter, nsrch, istart, lbig data eps /1d-04/, tol /0.0d0/, cgce /0.5d0/ data maxit /16/ data lp /6/, mp /6/, nsrch /32768/, istart /0/ data lblock /.true./, grow /.true./, lbig /.false./ data abort1 /.true./, abort2 /.true./ end block data ma30id c although all common block variables do not have default values, we comment on all the common block variables here. с c c common block ma30e/ed holds control parameters c common /ma30ed/ lp, abort1, abort2, abort3 c the integer lp is the unit number to which the error messages are c sent. Ip has a default value of 6. this default value can be reset by the user, if desired. a value of 0 suppresses all С messages C c the logical variables abort1, abort2, abort3 are used to control the c conditions under which the subroutine will terminate. c if abort1 is true, then the subroutine will exit immediately on c detecting structural singularity. c if abort2 is true, then the subroutine will exit immediately on detecting numerical singularity. с c if abort3 is .true, then the subroutine will exit immediately when the available space in a/icn is filled up by the previously с decomposed, active, and undecomposed parts of the matrix. с c the default values for abort1, abort2, abort3 are set to true., true.

c and .false. respectively.

с

c the variables in the common block ma30f/fd are used to provide the

- user with information on the decomposition.
- с common /ma30fd/ irnep, icnep, irank, minirn, minicn с
- c irncp and icncp are integer variables used to monitor the adequacy
- of the allocated space in arrays irn and a/icn respectively, by c
- taking account of the number of data management compresses
- required on these arrays. if irncp or icncp is fairly large (say с
- greater than n/10), it may be advantageous to increase the size с
- of the corresponding array(s). irncp and icncp are initialized с
- to zero on entry to ma30a/ad and are incremented each time the С
- compressing routine ma30d/dd is entered.
- c icncp is the number of compresses on a/icn.
- c irncp is the number of compresses on irn.
- c irank is an integer variable which gives an estimate (actually an
- upper bound) of the rank of the matrix. on an exit with iflag с
- equal to 0, this will be equal to n. с
- c minim is an integer variable which, after a successful call to
- c ma30a/ad, indicates the minimum length to which irn can be
- reduced while still permitting a successful decomposition of the с
- same matrix, if, however, the user were to decrease the length с
- of irn to that size, the number of compresses (irncp) may be с
- very high and quite costly, if lirn is not large enough to begin с
- the decomposition on a diagonal block, minim will be equal to с
- the value required to continue the decomposition and iflag will ¢
- be set to -3 or -6. a value of lim slightly greater than this с
- с (say about n/2) will usually provide enough space to complete
- the decomposition on that block, in the event of any other с
- С failure minim gives the minimum size of irn required for a
- successful decomposition up to that point. Ċ
- c minicn is an integer variable which after a successful call to
- ma30a/ad, indicates the minimum size of licn required to enable с
- a successful decomposition. in the event of failure with iflag=
- -5, minion will, if abort3 is left set to .false., indicate the с
- minimum length that would be sufficient to prevent this error in с
- c a subsequent run on an identical matrix, again the user may
- prefer to use a value of icn slightly greater than minicn for с
- subsequent runs to avoid too many conpresses (icncp). in the c event of failure with iflag equal to any negative value except
- с
- -4, minicn will give the minimum length to which licn could be с ¢ reduced to enable a successful decomposition to the point at
- which failure occurred. notice that, on a successful entry с
- idisp(2) gives the amount of space in a/icn required for the ¢
- decomposition while minicn will usually be slightly greater с
- because of the need for "elbow room". if the user is very с
- unsure how large to make licn, the variable minicn can be used с
- to provide that information. a preliminary run should be с
- performed with abort3 left set to .false. and licn about 3/2 с
- с times as big as the number of non-zeros in the original matrix.
- unless the initial problem is very sparse (when the run will be с
- successful) or fills in extremely badly (giving an error return с
- with iflag equal to -4), an error return with iflag equal to -5 с
- should result and minicn will give the amount of space required с
- с for a successful decomposition.
- c

c common block ma30g/gd is used by the ma30b/bd entry only. c common /ma30gd/ eps, rmin

- c eps is a real/double precision variable, it is used to test for
- small pivots. its default value is 1.0e-4 (1.0d-4 in d version).
- if the user sets eps to any value greater than 1.0, then no с
- check is made on the size of the pivots, although the absence of с
- с such a check would fail to warn the user of bad instability, its
- absence will enable ma30b/bd to run slightly faster. an a с
- ¢ posteriori check on the stability of the factorization can be
- obtained from mc24a/ad. с
- c rmin is a real/double precision variable which gives the user some
- information about the stability of the decomposition. at each с
- stage of the lu decomposition the magnitude of the pivot apiv с
- is compared with the largest off-diagonal entry currently in its с
- row (row of u), rowmax say. if the ratio с
- min (apiv/rowmax) c
- where the minimum is taken over all the rows, is less than eps С
- then rmin is set to this minimum value and iflag is returned с
- with the value +i where i is the row in which this minimum
- occurs. if the user sets eps greater than one, then this test ç

c is not performed. in this case, and when there are no small

- c pivots rmin will be set equal to eps.
- с

c common block ma30h/hd is used by ma30c/cd only.

c common /ma30hd/ resid

- c resid is a real/double precision variable. in the case of singular
- c or rectangular matrices its final value will be equal to the
- c maximum residual for the unsatisfied equations; otherwise its value will be set to zero.
- c value will c
- c common block ma30i/id controls the use of drop tolerances, the
- c modified pivot option and the the calculation of the largest
- c entry in the factorization process, this common block was added
- c to the ma30 package in february, 1983.
- c common /ma30id/ tol, big, ndrop, nsrch, lbig
- c tol is a real/double precision variable. if it is set to a positive
- c value, then ma30a/ad will drop from the factors any non-zero
- c whose modulus is less than tol. the factorization will then
- c require less storage but will be inaccurate. after a run of
- c ma30a/ad where entries have been dropped, ma30b/bd should not
- c be called. the default value for tol is 0.0.

c big is a real/double precision variable. if lbig has been set to

- c .true., big will be set to the largest entry encountered during
- c the factorization.
- c ndrop is an integer variable. if tol has been set positive, on exit
- c from ma30a/ad, ndrop will hold the number of entries dropped c from the data structure.
- c nsrch is an integer variable. if nsrch is set to a value less than
- c or equal to n, then a different pivot option will be employed by
- c ma30a/ad. this may result in different fill-in and execution
- c time for ma30a/ad. if nsrch is less than or equal to n, the

workspace arrays lastc and nextc are not referenced by ma30a/ad.
 the default value for nsrch is 32768.

c lbig is a logical variable. if lbig is set to .true., the value of

- c the largest entry encountered in the factorization by ma30a/ad
- c is returned in big. setting lbig to true. will marginally
- c increase the factorization time for ma30a/ad and will increase
- c that for ma30b/bd by about 20%. the default value for lbig is

c .false. c

> REAL eps, rmin, tol, big logical abort1, abort2, abort3, lbig common /ma30ed/ lp, abort1, abort2, abort3 common /ma30g/ eps, rmin common /ma30id/ tol, big, ndrop, nsrch, lbig data eps /1.0d-4/, tol /0.0d0/, big /0.0d0/ data lp /6/, nsrch /32768/ data lbig / false./ data abort1 /.true./, abort2 /.true./, abort3 /.false./ end

block data mc23cd logical abort common /mc23bd/ lp,numnz,num,large,abort data lp/6/,abort/.false./ end