UNIVERSIDADE FEDERAL DA PARAíBA

CENTRO DE CIÊNCIAS E TECNOLOGIA

CURSO DE MESTRADO EM INFORMÁTICA

TÉCNICAS DE TRANSFORMAÇÃO DE PROGRAMAS FONTES - ESTUDO E APLICAÇÃO NA TRANSFORMAÇÃO DE PROGRAMAS FORTRAN-66 PARA FORTRAN-77

FRANCISCO DE ASSIS DA COSTA SILVA

CAMPINA GRANDE - PB

ABRIL - 1992

FRANCISCO DE ASSIS DA COSTA SILVA

TÉCNICAS DE TRANSFORMAÇÃO DE PROGRAMAS FONTES - ESTUDO E APLICAÇÃO NA TRANSFORMAÇÃO DE PROGRAMAS FORTRAN-66 PARA FORTRAN-77

Dissertação apresentada ao CURSO DE MESTRADO EM INFORMÁTICA da Universidade Federal da Paraíba, em cumprimento a parte das exigências para obtenção do Grau de Mestre.

MÁRIO TOYOTARO HATTORI
Orientador

CAMILO DE LELIS GONDIM MEDEIROS
Co-Orientador

S586t Silva, Francisco de Assis da Costa
Tecnicas de transformacao de programas fontes : estudo e aplicacao na transformacao de programas Fortran-66 para Fortran-77 / Francisco de Assis da Costa Silva. - Campina Grande, 1992.
186 f. : il.

Dissertacao (Mestrado em Informatica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Programas de Sistemas 2. Programas Fontes 3. Programas Fortran 4. Informatica 5. Dissertacao I. Hattori, Mario Toyotaro, M.Sc. II. Medeiros, Camilo de Lelis Gondim, M.Sc. III. Universidade Federal da Paraiba - Campina Grande (PB) IV. Título

CDU 004.45(043)

TECNICAS DE TRANSFORMAÇAO DE PROGRAMAS FONTES - ESTUDO E APLICAÇAO NA TRANSFORMAÇAO DE PROGRAMAS FORTRAN-66 PARA FORTRAN-77

FRANCISCO DE ASSIS DA COSTA SILVA

DISSERTAÇAO APROVADA EM 15.04.1992

MARIO TOYOTARO HATTORI, M.Sc
Orientador

CAMILO DE LELIS GONDIM MEDEIROS, M.Sc
Componente da Banca

BRUNO CORREIA DA NOBREGA QUEIROZ, M.Sc
Componente da Banca

MAURO CAVALCANTE PEQUENO, Dr.
Componente da Banca

Campina Grande, 15 de abril de 1992

A meus pais, Saly e Hilda.

RESUMO

Neste trabalho é inicialmente feita uma resenha e depois é discutida a viabilidade de aplicação de várias técnicas de transformação de programas fontes na conversão de programas escritos em FORTRAN-66 para programas equivalentes em FORTRAN-77.

Na tentativa de identificar trechos de programas dos quais os GOTO's pudessem ser eliminados, o fluxo de controle desses programas foi modelado por grafos.

E por fim, são apresentados o projeto e a implementação de um sistema protótipo de conversor automático de programas para executar a transformação de programas escritos em FORTRAN-66 para programas em FORTRAN-77.

ABSTRACT

This work surveys and discusses the feasibility of applying several transformation technics for converting FORTRAN-66 programs into equivalent FORTRAN-77 programs.

In the quest for identifying program parts where GOTO's can be eliminated, control flows were modeled by graphs.

The design and implementation of a prototype FORTRAN-66 to FORTRAN-77 conversion system are presented.

SUMÁRIO

LISTA DAS ILUSTRAÇÕES

1 - INTRODUÇÃO

1.1 - OBJETIVOS

Programas em FORTRAN-66 eram obrigados a usar frequentemente o comando GOTO porque a linguagem não tinha comandos de iteração como WHILE nem estruturas do tipo IF-THEN-ELSE. Com o surgimento da versão FORTRAN-77 apareceu a estrutura IF-THEN-ELSE e assim, surgiu o interesse em reestruturar programas FORTRAN-66 transformando os trechos desses programas onde a estrutura IF-THEN-ELSE poderia ser usada, a fim de melhorar a sua legibilidade.

O objetivo principal deste trabalho é de estudar técnicas de transformação de programas fontes e verificar a sua aplicabilidade na transformação de algumas construções do FORTRAN-66 para FORTRAN-77 e de construções do FORTRAN-77 para FORTRAN-77 tais como multiplicação de matriz por vetor, com o objetivo de melhorar a eficiência dos programas quando executados em computadores com arquiteturas especiais dos modernos supercomputadores.

Com o advento dos supercomputadores, a preocupação com eficiência do software, atualmente considerada não prioritária, volta a ser um importante aspecto a ser considerado no processo de desenvolvimento de software. Ao desenvolver software para supercomputadores, procura-se explorar o máximo de sua velocidade de execução.

Uma das questões interessantes que tentaremos responder é a possibilidade ou não de aplicação de uma técnica. Por exemplo, consideremos o fragmento de um programa em FORTRAN-66:

```
      ......

      IF (.NOT. EXTALL) GOTO 50
      IF (ABS (B1 - A1) .GT. SMALL) ERLARG = ERLARG + ERR012
      IF (EXTRAP) GOTO 70
   50 CONTINUE
      WIDTH = ABS (BLIST(MAXERR) - ALIST(MAXERR))
      IF (WIDTH .GT. SMALL) GOTO 140
      IF (EXTALL) GOTO 60
      SMALL = SMALL * 5.0E+00
      IF (2.5E-01 * WIDTH * DOMEGA .GT. 2.0E+00) GOTO 140
      EXTALL = .TRUE.
      GOTO 130
   60 CONTINUE
      EXTRAP = .TRUE.
      NRMAX = 2
   70 CONTINUE
      IF (IERRO .EQ. 3 .OR. ERLARG .LE. ERTEST) GOTO 90
      ......
   90 CONTINUE
      ......
  130 CONTINUE
      ......
  140 CONTINUE
      ......
```

Numa primeira análise, parece não haver possibilidade de aplicar transformações capazes de eliminar os GOTO's. O problema é saber como detectar fragmentos não transformáveis para atingir alguma meta (no exemplo, eliminar GOTO's).

Para tentar resolver o problema de eliminação do comando GOTO tentaremos encontrar uma ferramenta de modelagem que nos responda formalmente quando uma construção IF (...) GOTO pode ser transformada e quando não pode ser transformada para uma estrutura IF-THEN ou IF-THEN-ELSE com a eliminação do GOTO.

Finalmente, como uma aplicação das técnicas vistas será apresentado um protótipo de um conversor automático de programas para transformar programas fontes escritos em FORTRAN-66 para FORTRAN-77, com ênfase na eliminação ou reorganização do GOTO para melhorar a estrutura.

1.2 - RELEVÂNCIA DO TRABALHO

O FORTRAN-77 trouxe novos recursos de programação, entre os quais a flexibilidade de utilização da estrutura IF-THEN-ELSE, um dos recursos que facilita a Programação Estruturada. Isso despertou o interesse em transformar a Biblioteca Transportável de Análise Numérica (BITAN) existente na Universidade Federal da Paraíba (UFPB), Campus II, Campina Grande - PB, composta de 260 algoritmos matemáticos implementados utilizando a linguagem FORTRAN-66, para a obtenção de uma nova versão desses algoritmos em FORTRAN-77 já que a versão mais antiga impôs, por falta de recursos, construções condenadas pela Programação Estruturada.

O interesse em transformar a BITAN consiste no fato de buscar a preservação do acervo dos programas científicos da BITAN com a inclusão ou substituição de construções obscuras por novas construções usando os novos recursos da linguagem FORTRAN, obtendo assim programas mais legíveis, bem estruturados e mais eficientes. A reutilização do software é um dos paradigmas da moderna Engenharia de Software.

As técnicas de transformação de programas fontes vêm se tornando ferramentas importantes e poderosas no processo de desenvolvimento de software. Essas técnicas são empregadas há muito tempo para atingirem diferentes metas no processo de desenvolvimento de software. Uma das metas é a melhoria de programas existentes, e isso motivou o interesse em estudar a viabilidade e a aplicabilidade das técnicas de transformação na conversão de programas escritos em FORTRAN-66 para FORTRAN-77.

Outro ponto relevante é verificar as implicações práticas da teoria de Bohm & Jacopini [02] segundo a qual qualquer programa pode ser expresso por uma sequência de código, IF-THEN-ELSE e um comando WHILE.

1.3 - APRESENTAÇÃO

Como resultado de nosso estudo de técnicas de transformação de programas fontes, descrevemos e comentamos algumas técnicas de transformação no segundo capítulo.

No terceiro capítulo fazemos considerações sobre Sistemas Transformacionais e apresentamos um desses sistemas que consideramos importante.

Características da nova versão FORTRAN, o FORTRAN-77, e Regras Transformacionais que exploram essas características na transformação de programas FORTRAN-66 para programas em FORTRAN-77, são analisadas no quarto capítulo.

No quinto capítulo, apresentamos o projeto do sistema protótipo desenvolvido.

Finalmente, no sexto capítulo, concluimos fazendo comentários gerais a respeito do trabalho bem como identificamos pontos que poderão ser melhorados tanto com relação ao sistema protótipo quanto ao trabalho em si, e apresentamos sugestões para futuros trabalhos.

1.4 - CONCEITOS

A terminologia usada neste trabalho foi tirada de Partsch & Steingbruggen [20].

Sistemas Transformacionais ou **Sistemas de Transformação** são sistemas implementados para dar apoio à Programação Transformacional.

Programação Transformacional, é uma metodologia de construção de programas por aplicações sucessivas de Regras Transformacionais.

Regras de Transformação, também chamadas **Regras**, **Transformações** ou em determinadas ocasiões, **Transformações Fonte-a-Fonte**, são mapeamentos parciais de um Esquema de Programa para um outro. Em sua forma mais geral, uma transformação é uma relação entre dois esquemas de programa P e P'. Ela é dita ser válida se uma certa relação semântica é mantida entre P e P'.

Esquema de Programa é a representação de uma classe de programas relacionados. Um esquema de programa provém do programa correspondente por parametrização.

Loveman [17] define **Otimização** como sendo a aplicação de um conjunto de regras para manipular várias representações de um programa explorando invariâncias locais ou globais de programa com a finalidade de melhorá-lo de alguma forma.

Hattori & Queiroz [14] definem que um software é **Portátil**, caso seja compilável e executável sem nenhuma alteração, em qualquer ambiente que aceite a mesma linguagem fonte. Um software portátil satisfaz o atributo **portabilidade**. Um

software é **Transportável** quando construções que impossibilitam alcançar a portabilidade podem ser documentadas e usadas como parâmetros de modo que um programa possa efetuar as modificações necessárias ao se mover o software de um ambiente de computação para outro. Um software transportável satisfaz o atributo **transportabilidade.**

A diferença entre portabilidade e transportabilidade consiste então, no fato de que um software portátil é compilável e executável sem nenhuma alteração em qualquer ambiente computacional que aceite a mesma linguagem fonte, enquanto que, um software transportável só será se alguns parâmetros forem alterados.

Um **Grafo G** é uma tripla ordenada (V(G), E(G), ØG) consistindo de um conjunto não vazio de vértices (V(G)), um conjunto de arestas (E(G)), disjunto de V(G), e uma função incidência ØG que associa cada aresta de G com um par não ordenado de vértices (não necessariamente distintos) de G. Um **Grafo Planar** é aquele grafo que apresenta um diagrama onde suas arestas se encontram somente em seus extremos, ou seja, não há intersecção das arestas. Caso contrário, é um **Grafo Não Planar.** Detalhes sobre grafos e planaridade podem ser encontrados em Bondy & Murty [03] e em Reingold et al. [22].

1.5 - META-LINGUAGEM

No segundo capítulo, da seção 2.1 à seção 2.5, utilizamos uma meta-linguagem para ilustrar a aplicação de várias técnicas de transformação de programas estudadas.

A meta-linguagem definida consistirá de:

(a) Constantes:

As constantes são representadas por:

- constantes numéricas: 1, 3.14, 586, etc.
- constantes lógicas: true e false.
- constantes literais: 'JOSÉ DA SILVA', '12345', etc.

(b) Identificadores de Variáveis:

O Identificador de variável é formado por uma letra ou por uma letra seguida por um número arbitrário de letras ou dígitos. Por exemplo: A, A123, A1C, VALOR e etc.

Os tipos de variáveis são integer, real, logical e string.

Uma variável pode representar um conjunto de elementos. O tamanho do conjunto é delimitado entre colchetes. Por exemplo: A[10], B[N], C[M,N].

(c) Expressões aritméticas:

São válidos os seguintes operadores aritméticos: + (adição), - (subtração), * (multiplicação) e / (divisão).

(d) Expressões lógicas:

As expressões lógicas podem ser compostas de resultados de operações relacionais do tipo: =, ≠, >, ≥, < e ≤, e variáveis lógicas combinadas por operadores lógicos: and, or e not.

(e) Comando de atribuição:

O comando de atribuição apresenta a seguinte forma:

A <-- B, onde <-- é definido como sendo o operador de atribuição.

(f) Estruturas condicionais:

Podem ser utilizadas estruturas condicionais na forma:

```
(1) if condição then X <-- X + 1;
(2) if condição then Comando Simples else Comando Simples;
(3) if condição goto X;
(4) if condição then begin
                        Bloco de Comandos
                     end
                else begin
                        Bloco de comandos
                     end;
```

onde condição é uma expressão lógica.

(g) Estruturas de repetição:

Os comandos de repetição têm as seguintes formas:

(1) **FOR**

O comando FOR pode assumir as seguintes formas:

```
- for I <-- 1 step 1 until 10 do
      Comando Simples;


- for I <-- 1 step 1 until 10 do
    begin
      Bloco de Comandos;
    end;
```

(2) **REPEAT**

O comando REPEAT tem a seguinte estrutura:

```
repeat
  Comandos
until condição;
```

(05) **WHILE**

O comando WHILE pode ser utlizado nas seguintes formas:

- while condição do
  Comando Simples;

- while condição do
  begin
    Bloco de Comandos
  end;

(h) Comando GOTO:

As transferências incondicionais são executadas pelo comando GOTO, na forma GOTO L, onde L é um rótulo alfanumérico ou numérico. Por exemplo:

```
.....
L : comando
    comando
    goto L;
    .....
```

(i) Procedimentos:

Os procedimentos devem apresentar a seguinte estrutura:

```
procedure NOME (parâmetros);
declaração dos parâmetros;
begin

     corpo do procedimento (é obrigatório usar pelo
                            menos um comando return)

end.
```

2 - TÉCNICAS DE TRANSFORMAÇÃO DE PROGRAMAS FONTES

2.1 - INTRODUÇÃO

As técnicas de transformação de programas fontes juntamente com os sistemas transformacionais, sistemas responsáveis pela aplicação dessas técnicas, vêm se tornando ferramentas importantes e poderosas no processo de desenvolvimento de software.

Segundo Feather [9], motivados pelo aumento significativo dos custos de desenvolvimento e manutenção dos programas, os pesquisadores têm procurado novas metodologias para o seu desenvolvimento. Uma metodologia que vem se destacando é a de transformação de programas fontes.

Apresentamos neste capítulo técnicas de transformação de programas fontes, e no capítulo 3, abordamos aspectos relacionados com sistemas transformacionais.

As técnicas de transformação de programas fontes são empregadas para atingir diferentes metas no processo de desenvolvimento de software. Uma das metas é a modificação de programas.

A modificação de programas tenta conseguir:

- otimização a nível fonte; A otimização pode ser efetuada, por exemplo, eliminando variáveis redundantes e atribuições redundantes, eliminando subexpressões comuns, substituindo variáveis, computando constantes fora do ciclo de repetição e otimizando estruturas de controle;
- adaptação de programas a outros ambientes de computação;

- conversão de programas escritos em simples para dupla precisão, e vice-versa;
- melhorias nos programas que apresentam certas restrições operacionais tais como espaço restrito para executar programas e lentidão no tempo de resposta;
- uniformização de estilo de texto fonte.

Outra meta, muito explorada atualmente, é a utilização de técnicas de transformação para converter programas escritos em uma linguagem em programas equivalentes em uma outra linguagem.

As técnicas de transformação mais utilizadas inicialmente foram as técnicas para melhorar a eficiência dos programas. A ênfase na eficiência, era justificada porque a velocidade de processamento dos computadores era baixa comparada com a dos modernos computadores. Como problemas cada vez mais complexos precisam ser resolvidos, a eficiência dos programas continua importante.

A preocupação com a eficiência pode implicar num sacrifício de confiabilidade e de legibilidade dos programas. A legibilidade é muito importante ser considerada no processo de desenvolvimento de software, principalmente nas fases de documentação e manutenção.

Surgiram também técnicas de transformação que facilitam alcançar a portabilidade e a transportabilidade de softwares.

Hoje em dia uma maior ênfase está sendo dada às técnicas que procuram oferecer maior legibilidade aos programas.

Apresentamos a seguir técnicas de transformação de programas fontes que o nosso estudo mostrou serem interessantes e

importantes.

Algumas das técnicas de transformação que vamos apresentar já são raramente utilizadas, pois a criação de novas metodologias para facilitar o processo de desenvolvimento de software e o aperfeiçoamento de recursos de software disponíveis como as linguagens de programação, naturalmente conduzem à produção de programas mais legíveis e eficientes.

Não podemos esquecer, entretanto, que muitas rotinas que compõem determinadas bibliotecas foram escritas sem a ajuda de ferramentas e metodologias de desenvolvimento de software disponíveis hoje. Devido a isso, pode haver a necessidade de utilizar algumas técnicas de transformação de programas para converter esses programas para torná-los mais legíveis e assim, facilitar a sua manutenção.

Para uma melhor organização da apresentação, dividimos as técnicas de transformação de programas fontes em cinco (5) grupos de acordo com as características das técnicas. Algumas técnicas podem ser incluídas em diferentes grupos. A característica mais marcante norteará a inclusão de uma técnica num grupo.

As técnicas de transformação são agrupadas de acordo com a finalidade desejada: simplificação, aumento da eficiência, melhoria da estrutura, eliminação ou uso mais racional do GOTO, e exploração de recursos de hardware. O GOTO merecerá uma seção à parte, embora pertença ao grupo de melhoria da estrutura.

2.2 - TRANFORMAÇÕES PARA SIMPLIFICAÇÃO

Apresentamos neste grupo, algumas técnicas de transformação de programas utilizadas com a finalidade de simplificar determinadas expressões ou construções de programas, resultando algumas vezes, em expressões e construções mais legíveis e/ou mais eficientes.

Começamos mostrando técnicas de transformação apresentadas por Loveman [17], Standish et al. [26] e Tassel [27]. Algumas técnicas que realizam a mesma função recebem nomes diferentes nos três trabalhos. Quando isso acontece, adotamos o nome da técnica apresentada por Standish et al. [26].

2.2.1 - Computação de constantes:

Esta técnica consiste em simplificar expressões que já apresentam resultados óbvios. Por exemplo:

SQRT(9) ==> 3

2.2.2 - Eliminação de atribuição por igualdade:

Se A = B, então a atribuição A <-- B pode ser eliminada.

2.2.3 - Eliminação de referências a elementos de conjuntos:

Esta técnica consiste em substituir repetidas referências a um mesmo elemento de um conjunto com a criação de uma variável extra.

CÓDIGO FONTE	CÓDIGO TRANSFORMADO
A[I,J] <-- B * C	T <-- B * C
.....	A[I,J] <-- T
	.....
A[J,I] <-- A[I,J] * 2	A[J,I] <-- T * 2

Esta técnica é útil para melhorar a eficiência, pois para calcular o endereço de A[I,J] toda vez que referenciar A[I,J] é necessário buscar na memória os valores das variáveis I e J, enquanto que o acesso a uma variável sem indexação é direto.

2.2.4 - Eliminação de atribuições redundantes:

Uma atribuição de valor a uma variável é redundante se esta não for referenciada antes da ocorrência de uma outra atribuição. Por exemplo:

CÓDIGO FONTE	CÓDIGO TRANSFORMADO
X <-- 2 * A + B	VAZIO
Y <-- 3 * B - C	Y <-- 3 * B - C
Z <-- D / 4 + E	Z <-- D / 4 + E
X <-- Y + 2 * Z	X <-- Y + 2 * Z

A atribuição da primeira linha é redundante porque X não é usado antes da atribuição da última linha.

2.2.5 - Eliminação de atribuições inúteis:

Esta técnica é utilizada para eliminar comandos de atribuições que atribuem valores a variáveis que não são usadas subsequentemente no programa. Por exemplo: suponha que X, Y e Z sejam variáveis de entrada e U, V e W variáveis de saída. Assumimos que as variáveis de entrada tenham valores iniciais, e que somente os valores de saída são necessários. Em particular, R não é uma variável de saída.

CÓDIGO FONTE	CÓDIGO INTERMEDIÁRIO	CÓDIGO TRANSFORMADO
U <-- X + 2 * Y	U <-- X + 2 * Y	U <-- X + 2 * Y
T <-- 3 * U + Z	T <-- 3 * U + Z	T <-- 3 * U + Z
S <-- T - 1	S <-- T - 1	VAZIO
R <-- S + X	VAZIO	VAZIO
V <-- U / 6	V <-- U / 6	V <-- U / 6
W <-- T + U	W <-- T + U	W <-- T + U

Quando empregamos a técnica de eliminação de atribuições inúteis conseguimos reduzir o número de comandos e assim, melhoramos o desempenho do programa.

2.2.6 - Reordenação de atribuições para eliminação de armazenamentos temporários:

Esta técnica consiste em eliminar variáveis utilizadas para armazenar resultados temporários, com a reutilização de variáveis que não mais serão referenciadas no programa. A variável a ser eliminada deve ser definida após a definição e utilização da variável que a substituirá. Por exemplo:

CÓDIGO FONTE	CÓDIGO TRANSFORMADO
X <-- 2 * A + B	X <-- 2 * A + B
T <-- X / 4 - C	T <-- X / 4 - C
U <-- X + M * I	U <-- X + M * I
.....	.....

sem referências a X a partir deste ponto; X pode ser usado como variável auxiliar de outra expressão.

.....	.....
Y <-- A - 2 * B	X <-- A - 2 * B
T <-- C - Y / 4	T <-- C - X / 4
U <-- M * I + Y	U <-- M * I + X
.....	.....

sem referências a Y a partir deste ponto

.....	.....

Esta técnica é empregada para economizar espaço de memória.

2.2.7 - Eliminação de subexpressões comuns:

Suponha que uma subexpressão E de um comando de atribuição (X <-- E) é usada duas ou mais vezes em uma sequência de comandos e considere que T é uma nova variável que não aparece em outra parte do programa. Assim, podemos eliminar subexpressões da forma:

```
     CÓDIGO FONTE          |    CÓDIGO TRANSFORMADO
                           |
   X <-- 4 * A * C         |      T <-- 4 * A * C
      .....                |      X <-- T
                           |         .....
   Y <-- Y + (4 * A * C)   |      Y <-- Y + T
```

É importante alertar que nenhuma variável da subexpressão 4 * A * C pode receber novo valor entre os comandos de atribuição X e Y, respectivamente.

2.2.8 - Simplificação de condicionais triviais:

Esta técnica, como o próprio nome sugere, consiste em simplificar estruturas condicionais. Por exemplo:

```
a) if B then true else false      ==>      B
b) if B then false else true      ==>  not B
c) if true then P else Q          ==>      P
d) if false then P else Q         ==>      Q
e) if B then P else P             ==>      P
f) if B then P else VAZIO         ==>  if B then P
g) if B then VAZIO else Q         ==>  if not B then Q
h) if not B then P else Q         ==>  if B then Q else P
i) if B then C
   if not B then D                ==>  if B then C else D
```

2.2.9 - Eliminação de declarações inúteis:

Se uma variável for declarada mas nunca usada no escopo de sua declaração, a declaração pode ser eliminada. Por exemplo:

CÓDIGO FONTE	CÓDIGO TRANSFORMADO
.....	.....
integer X, Y	integer Y
.....	.....
X não é usada	X não é usada
.....	.....
end	end

2.2.10 - Eliminação de identidades aritméticas:

Identidades aritméticas dos tipos abaixo discriminadas podem ser simplificadas para as suas respectivas formas equivalentes, conforme podemos observar nos exemplos a seguir:

```
A + 0  ==>  A          A - 0  ==>  A
0 + A  ==>  A          0 - A  ==> -A
A * 1  ==>  A          A / 1  ==>  A
1 * A  ==>  A          0 / A  ==>  0
0 * A  ==>  0          A * 0  ==>  0
```

Atentar para os casos especiais:

a) A / 0 ==> indefinido;

b) Considerando que A = 0 e B ≠ 0,

B / A ==> indefinido

2.2.11 - Eliminação do Comando VAZIO:

Esta técnica é utilizada para eliminar o Comando VAZIO. Por exemplo:

a) while false do ==> VAZIO;
   Comando Simples;

Como o escopo do comando WHILE só é executado se a análise da condição do WHILE for verdadeira, no exemplo acima, o escopo do comando WHILE nunca será executado pois a condição é falsa. Outros exemplos de eliminação do comando VAZIO são apresentados a seguir.

b) if false then Comando Simples; ==> VAZIO;

c) Considerando que B é false,

```
L : if B then begin
      Comandos;          ==>   L : VAZIO;
      goto L;
    end;
```

d) for I:= 1 step 1 until N do VAZIO; ==> VAZIO;

2.2.12 - Distribuição em estruturas condicionais:

Suponha que @ é um operador binário do tipo aritmético, relacional ou de atribuição. Considerando que uma estrutura condicional é um operando de um operador do tipo acima citado, temos que:

(1) X @ (if B then C else D) ==> if B then (X @ C) else (X @ D);

(2) (if B then C else D) @ X ==> if B then (C @ X) else (D @ X);

Exemplificando:

```
(a) X + (if A > 5 then Y else Z)  ==>  if A > 5 then (X + Y)
                                       else (X + Z);

(b) (if A > 5 then Y else Z) + X  ==>  if A > 5 then (Y + X)
                                       else (Z + X);
```

Esta técnica, não comum nos dias de hoje, é utilizada para melhorar a legibilidade.

2.2.13 - Simplificação com 0 e 1:

Esta técnica consiste em simplificar expressões que envolvem operações com as constantes numéricas 0 e 1. Considerando que ⟨ # ⟩ é uma relação, temos que:

a) $A - B \langle \# \rangle 0 \Longrightarrow A \langle \# \rangle B$
b) $0 \langle \# \rangle A - B \Longrightarrow B \langle \# \rangle A$
c) se $B > 0$, então:
$A / B \langle \# \rangle 1 \Longrightarrow A \langle \# \rangle B$
$1 \langle \# \rangle A / B \Longrightarrow B \langle \# \rangle A$
d) se $A > 0$ e $B > 0$, então:
$(1 / A) \langle \# \rangle (1 / B) \Longrightarrow B \langle \# \rangle A$

2.2.14 - Simplificação para true e false:

Esta técnica é utilizada para simplificar expressões relacionais. Por exemplo:

$A = A \Longrightarrow$ true $A \leq A \Longrightarrow$ true $A \geq A \Longrightarrow$ true
$A \neq A \Longrightarrow$ false $A < A \Longrightarrow$ false $A > A \Longrightarrow$ false

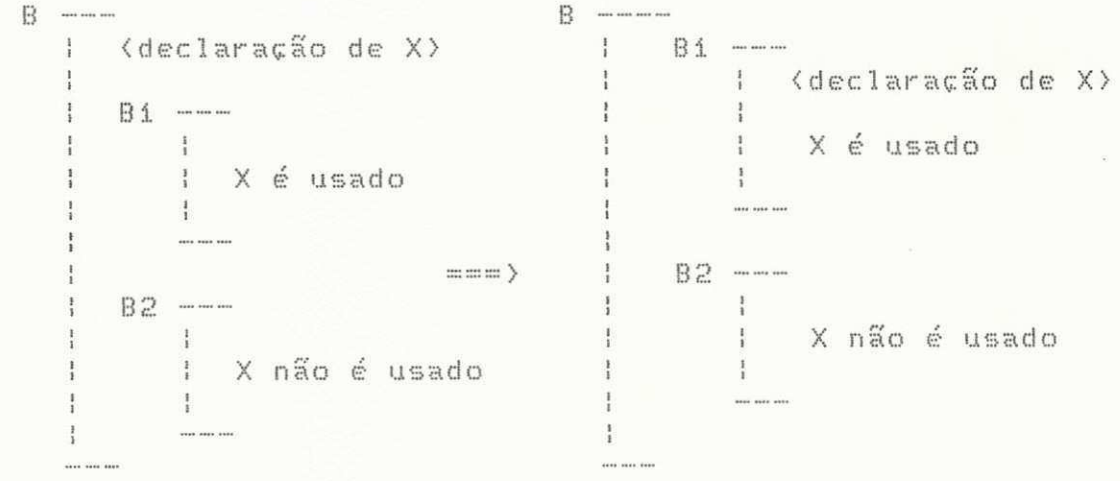
2.2.15 - Eliminação de constantes booleanas:

Esta técnica consiste em simplificar expressões que utilizam operadores lógicos e que apresentam resultados óbvios. Por exemplo:

A and true ==> A A or true ==> true
true and A ==> A true or A ==> true
A and false ==> false A or false ==> A
false and A ==> false false or A ==> A
not true ==> false not false ==> true

2.2.16 - Usando estrutura de bloco para salvar espaço:

Variáveis declaradas em blocos disjuntos podem usar o mesmo espaço desde que suas ativações ocorram em tempos disjuntos. Assim, se uma variável X for usada no bloco B1 mas não no bloco B2, e é declarada em algum bloco global a B1 e B2, recuperamos espaço de memória movendo a declaração para o bloco B1. Por exemplo:



2.2.17 - Negação de relações lógicas:

É interessante aplicar esta técnica para simplificar as expressões lógicas e consequentemente torná-las mais legíveis.

a) not (A = B) ==> A ≠ B

b) not (A ≠ B) ==> A = B

c) not (A < B) ==> A ≥ B

d) not (A ≤ B) ==> A > B

e) not (A > B) ==> A ≤ B

f) not (A ≥ B) ==> A < B

2.3 - TRANSFORMAÇÕES PARA MELHORAR EFICIÊNCIA

Muitas técnicas foram criadas e empregadas com o propósito de melhorar a eficiência de execução dos programas. Eficiência volta a ser assunto de discussão com a nova era dos supercomputadores.

O termo eficiência envolve a preocupação com espaço e tempo de execução.

Apresentamos a seguir, um conjunto de técnicas de transformação muito utilizadas e que exemplificam a grande variedade de aplicações das técnicas empregadas para melhorar o desempenho dos programas. Algumas técnicas que buscam eficiência já foram apresentadas no grupo de simplificação e outras serão apresentadas nos grupos de estrutura e estrutura de GOTO e recursos de hardware.

2.3.1 - Eliminação de cadeia de gotos:

Esta técnica é empregada para alterar o destino de uma transferência do comando GOTO que faz referência a uma linha de código cujo conteúdo é um outro comando GOTO. Por exemplo:

CÓDIGO FONTE	CÓDIGO TRANSFORMADO
goto L1; ..... L1 : goto L2; ..... L2 : goto L3; ..... L3 :	goto L3; ..... L1 : goto L3; ..... L2 : goto L3; ..... L3 : .....

Lembrar que os rótulos L1 e L2 podem ser excluídos,

desde que nenhuma referência seja feita a eles em outro ponto do programa.

2.3.2 - Fusão de ciclos de repetição:

A técnica de fusão de ciclos de repetição consiste em fundir ciclos de repetição que apresentam o mesmo escopo das variáveis de controle. Por exemplo:

```
for I <-- 1 step 1 until 20 do A[I] <-- B[I];
for J <-- 5 step 5 until 100 do C[J/5] <-- 0;
```

O segundo controle de repetição pode ser transformado renomeando J por I e recalculando o incremento e o limite superior:

```
for I <-- 1 step 1 until 20 do A[I] <-- B[I];
for I <-- 1 step 1 until 20 do C[I] <-- 0;
```

fazendo a fusão dos ciclos de repetição, conseguimos:

```
for I <-- 1 step 1 until 20 DO
  begin
    A[I] <-- B[I];
    C[I] <-- 0;
  end;
```

Existem casos onde podemos ter comandos entre os ciclos de repetição envolvidos numa fusão. Por exemplo:

```
for I <-- 1 step 1 until 20 do
   A[I] <-- B[I];
J <-- K * 20;
L <-- X * 2 + Y;
for I <-- 1 step 1 until 20 do
   C[I] <-- 0;
```

Nestes casos, ao se aplicar a técnica de fusão de ciclos de repetição alguns cuidados devem ser tomados. É necessário, por exemplo, que os comandos entre os ciclos de repetição não sejam rotulados, não sejam GOTO's e que sejam

totalmente independentes dos escopos dos ciclos de repetição.

2.3.3 - Remoção de comandos invariantes:

A técnica de remoção de comandos invariantes consiste em transferir comandos que não dependem da variável de controle do escopo de um comando de controle de repetição para fora do escopo. O trecho de programa:

```
for V <-- 1 step 1 until N do
  begin
    X <-- Y + 2;
    A[V] <-- B[V - X];
  end;
```

pode ser substituído por:

```
X <-- Y + 2
for V <-- 1 step 1 until N do
    A[V] <-- B[V - X];
```

Existem casos onde comandos invariantes não podem ser removidos. Um desses casos ocorre quando uma variável definida no comando invariante é referenciada no escopo do ciclo de repetição antes de sua nova definição. Por exemplo:

```
J <-- N + 5;
for I <-- 1 step 1 until 10 do
   begin
     K <-- I + J;
     J <-- M;
   end;
```

2.3.4 - Substituição da variável de controle do ciclo de repetição:

Esta técnica elimina uma variável de controle e substitui por outra. Por exemplo:

```
CÓDIGO FONTE                    CÓDIGO TRANSFORMADO

 begin                            begin
 I <-- 1;                         K <-- 25;
 K <-- 25;                        while K ≤ 2500 DO
 while  I ≤ 100 do                  begin
   begin                              A[K] <-- 0;
     A[K] <-- 0;                      K <-- K + 25;
     K <-- K + 25;                  end;
     I <-- I + 1;                 end;
   end;
end;
```

2.3.5 - Desmembramento do ciclo de repetição:

Esta técnica é utilizada tanto para melhorar a eficiência como para melhorar a legibilidade de trechos de programas.

Podemos aplicar esta técnica em duas situações:

(1) considerando que J ≤ B:

```
for K <-- A step 1 until B do
  if K ≤ J then Comandos else Comandos;
```

podemos substituir o fragmento de programa anterior por:

```
for K <-- A step 1 until J do
  Comandos;
for K <-- J + 1 step 1 until B do
  Comandos;
```

(2) considerando que I - 1 ≤ B:

```
for K <-- A step 1 until B do
  if I ≤ K then Comandos else Comandos;
```

podemos substituir o fragmento de programa anterior por:

```
for K <-- A step 1 until I - 1 do
  Comandos;
for K <-- I step 1 until B do
  Comandos;
```

Para exemplificar melhor a aplicação desta técnica, suponha que temos o seguinte trecho de programa:

```
for I <-- 1 step 1 until 30 do
   if I < 15 then A[I] <-- B[I]
             else A[I] <-- - B[I];
```

aplicando a técnica de desmembramento do ciclo de repetição, podemos substituir o trecho anterior por:

```
for I <-- 1 step 1 until 15 do
   A[I] <-- B[I];
for I <-- 16 step 1 until 30 do
   A[I] <-- - B[I];
```

2.3.6 - Introdução de ninho condicional:

A estrutura condicional:

```
if A then S1; S2
```

pode ser substituída pela seguinte estrutura:

```
if A then S1 else S2;
```

desde que S1 seja um comando RETURN ou comando GOTO e S2 não seja rotulado.

2.3.7 - Eliminação de testes de pesquisa exaustiva por extensão da estrutura de dados:

O exemplo que vamos apresentar para ilustrar a aplicação da técnica de eliminação de testes de pesquisa exaustiva por extensão da estrutura de dados é uma adaptação do exemplo apresentado por Knuth [16]. Considere pesquisar uma tabela T para ver se ela contém um elemento X. Suponha que a tabela tem elementos numerados de 1 a N representados por T[1], T[2], ...,

T[N]. Um procedimento para determinar se T contém X é o seguinte:

```
procedure SEARCH(T,X); array T[1:N]; integer I;
begin
  I <-- N;
  while I > 0 do
    if T[I] = X
      then return(true)
      else I <-- I - 1;
  return(false);
end.
```

Aumentando o número de elementos na tabela para M, onde M <-- N + 1, obtemos o seguinte trecho transformado:

```
procedure SEARCH(T,X); array T[1:M]; integer I;
begin
  T[M] <-- X;
  I <-- 1;
  while T[I] ≠ X do I <-- I + 1;
  return(I ≠ M);
end.
```

Observamos que a maior preocupação de Knuth [16] quando apresenta uma solução para um determinado problema, como visto no exemplo anterior, é conseguir melhor eficiência dos programas, mesmo que para isso seja necessário sacrificar a legibilidade.

Loveman [17] também se interessava basicamente por técnicas de transformação que facilitavam a otimização, consequentemente, a eficiência.

Podemos ilustrar a abordagem de Loveman [17] com o seguinte exemplo: assuma que MULT é um procedimento que multiplicará matrizes L x M e M x N, gerando uma matriz L x N como mostramos a seguir.

```
procedure MULT(X:MATRIZ[L,M];Y:MATRIZ[M,N];Z:MATRIZ[L:N]);
   for I <-- 1 step 1 until L do
     for J <-- 1 step 1 until N do
       begin
         Z[I,J] <-- 0;
         for K <-- 1 step 1 until M do
            Z[I,J] <-- X[I,K] * Y[K,J] + Z[I,J];
       end;
end.
```

Vamos exemplificar uma chamada particular a MULT para multiplicar as matrizes A e B gerando a matriz produto C, em que A é uma matriz diagonal. Em uma matriz diagonal, I ≠ J implica em X[I,J] = 0.

```
declare A:MATRIZ[10,10]; B:MATRIZ[10,20]; C:MATRIZ[10,20];
assuma A ser uma matriz diagonal;
MULT(A,B,C);
```

Expandindo diretamente MULT em suas respectivas linhas com propagação de constantes, temos:

```
for I <-- 1 step 1 until 10 do
  for J <-- 1 step 1 until 20 do
    begin
      C[I,J] <-- 0;
      for K <-- 1 step 1 until 10 do
        C[I,J] <-- A[I,K] * B[K,J] + C[I,J];
    end;
```

Baseado no conhecimento de que A é uma matriz diagonal, podemos substituir A[I,K] por (if (I ≠ K) then 0 else A[I,K]). O comando de atribuição no ciclo de repetição mais interno torna-se então:

C[I,J] <--(if (I ≠ K) then 0 else A[I,K]) * B[K,J] + C[I,J]).

Expandindo o escopo do IF, temos:

```
for I <-- 1 step 1 until 10 do
  for J <-- 1 step 1 until 20 do
    begin
      C[I,J] <-- 0;
      for K <-- 1 step 1 until 10 do
        if (I ≠ K) then
          C[I,J] <-- 0 * B[K,J] + C[I,J]
        else
          C[I,J] <-- A[I,K] * B[K,J] + C[I,J];
    end;
```

Aplicando um subconjunto de técnicas apresentadas anteriormente de simplificação de expressões, eliminação de variáveis, eliminação de atribuições redundantes, obtemos o seguinte código simplificado:

```
for I <-- 1 step 1 until 10 do
  for J <-- 1 step 1 until 20 do
    C[I,J] <-- A[I,I] * B[I,J];
```

O processo completo de transformações parciais para o exemplo mostrado anteriormente é apresentado por Loveman [17].

Um outro exemplo interessante que ilustra a aplicação de técnicas de transformação é apresentado por Standish et al. [25]. O exemplo demonstra como usar um pequeno conjunto de técnicas de transformação para melhorar um fragmento de programa, que com suas novas adaptações é capaz de executar o mesmo trecho de programa aproximadamente seis vezes mais rápido.

O pequeno trecho de programa apresentado por Standish et al. [25] multiplica duas matrizes triangulares superiores A e B (N x N). Os elementos do produto da matriz C são obtidos pela fórmula:

$$C(I,J) = \sum_{K=1}^{N} [A(I,K) * B(K,J)]$$

O fragmento de programa correspondente para computar C é:

```
for I <-- 1 step 1 until N do
  for J <-- 1 step 1 until N do
    begin
      C[I,J] <-- 0;
      for K <-- 1 step 1 until N do
        C[I,J] <-- C[I,J] + A[I,K] * B[K,J];
    end;
```

Dado que A e B são matrizes triangulares superiores, podemos escrever:

```
A[I,K] <-- if (I ≤ K) then A[I,K] else 0;
B[K,J] <-- if (K ≤ J) then B[K,J] else 0;
```

Queremos agora simplificar o fragmento de programa

anterior com respeito às últimas expressões condicionais com a intenção de eliminar produtos envolvendo elementos de matrizes que são nulos.

Para melhorar o fragmento de programa, começamos transformando o produto A[I,K] * B[K,J] usado no comando de atribuição C[I,J] <-- C[I,J] + A[I,K] * B[K,J] no ciclo de repetição mais interno do fragmento do programa original. Expandimos este produto, substituindo as expressões condicionais apresentadas anteriormente. Assim, A[I,K] * B[K,J] é expandido para:

(1) (if I ≤ K then A[I,K] else 0) * (if K ≤ J then B[K,J] else 0)

Usando a técnica de distribuição em estruturas condicionais, a expressão anterior é transformada para:

(2) (if K ≤ J then (if I ≤ K then A[I,K] else 0) * B[K,J]
else (if I ≤ K then A[I,K] else 0) * 0)

Aplicando em seguida a técnica de eliminação de identidade aritmética, temos:

(3) (if K ≤ J then (if I ≤ K then A[I,K] else 0) * B[K,J]
else 0)

Aplicando novamente a técnica de distribuição em estruturas condicionais na expressão anterior, obtemos:

(4) (if K ≤ J then (if I ≤ K then A[I,K]*B[K,J] else 0 * B[K,J])
else 0)

Transformando a última expressão resultante com mais uma aplicação da técnica de eliminação de identidade aritmética, conseguimos a seguinte expressão resultante:

(5) (if K ≤ J then (if I ≤ K then A[I,K]*B[K,J] else 0)
else 0)

Podemos então substituir o produto C[I,J] <-- C[I,J] + A[I,K] * B[K,J] pela expressão **(5)** e aplicar outras

transformações como segue.

```
(6) C[I,J] <-- C[I,J]+(if K ≤ J then (if I ≤ K then A[I,K]*B[K,J]
                                                else 0)
                                     else 0)
```

Aplicando quatro vezes seguidas a técnica de distribuição em estruturas condicionais, obtemos:

```
(7) (if K ≤ J then (if I ≤ K then C[I,J] <-- C[I,J]+A[I,K]*B[K,J]
                                else C[I,J] <-- C[I,J] + 0)
                   else C[I,J] <-- C[I,J] + 0)
```

Ao aplicar, em seguida, duas vezes a técnica de eliminação de identidades aritméticas, conseguimos:

```
(8) (if K ≤ J then (if I ≤ K then C[I,J] <-- C[I,J]+A[I,K]*B[K,J]
                                else C[I,J] <-- C[I,J])
                   else C[I,J] <-- C[I,J])
```

Transformando a expressão resultante **(8)** com utilização da técnica de eliminação de atribuição por igualdade, obtemos:

```
(9) (if K ≤ J then (if I ≤ K then C[I,J] <-- C[I,J]+A[I,K]*B[K,J]
                                else VAZIO)
                   else VAZIO)
```

A atribuição C[I,J] <-- C[I,J] + A[I,K] * B[K,J] no ciclo de repetição mais interno do fragmento do programa original pode ser agora substituída pela expressão resultante **(9)**, dando uma nova versão ao ciclo de repetição como podemos observar a seguir.

```
          for K <-- 1 step 1 until N do
            if K ≤ J then (if I ≤ K then C[I,J] <-- C[I,J] +
(10)                                              A[I,K] * B[K,J]
                                        else VAZIO)
                         else VAZIO;
```

Aplicando a técnica de desmembramento do ciclo de repetição ao trecho anterior **(10)**, obtemos:

```
          for K <-- 1 step 1 until J do
            (if I ≤ K then C[I,J] <-- C[I,J] + A[I,K] * B[K,J]
(11)                      else VAZIO);
          for K <-- J + 1 step 1 until N do
            VAZIO;
```

O segundo ciclo de repetição como podemos observar, é um caso onde podemos aplicar a técnica de eliminação do comando vazio e assim, desaparecerá. Ao primeiro ciclo de repetição podemos também aplicar a técnica de eliminação de comando VAZIO, e em seguida de novo dividi-lo usando a técnica de desmembramento do ciclo de repetição. Aplicando tais técnicas conseguimos o seguinte trecho resultante:

```
          for K <-- 1 step 1 until I - 1 do
            VAZIO;
(12)      for K <-- I step 1 until J do
            C[I,J] <-- C[I,J] + A[I,K] * B[K,J];
```

O primeiro ciclo de repetição do trecho anterior **(12)** desaparecerá ao ser aplicada a técnica de eliminação de comando VAZIO. Assim, conseguimos o seguinte trecho:

```
        for K <-- I step 1 until J do
            C[I,J] <-- C[I,J] + A[I,K] * B[K,J];
```

Podemos agora substituir o trecho do ciclo de repetição mais interno do programa original por esta versão melhorada do trecho e obtemos a seguinte forma final do programa após a substituição do trecho:

```
          for I <-- 1 step 1 until N do
            for J <-- 1 step 1 N until do
              begin
                C[I,J] <-- 0;
                for K <-- I step 1 until J do
                  C[I,J] <-- C[I,J] + A[I,K] * B[K,J];
              end;
```

Ilustramos assim, pelo exemplo de Standish et al. [25], a utilização de um subconjunto de técnicas de transformação

apresentadas anteriormente, capaz de melhorar o desempenho dos programas.

2.4 - TRANSFORMAÇÕES DE ESTRUTURAS

Apresentamos neste grupo algumas técnicas de transformação de programas utilizadas para substituir determinadas construções por outras construções equivalentes com o objetivo de substituir estruturas não disponíveis em uma linguagem de programação por outras estruturas aceitáveis durante o processo de conversão de um programa em uma linguagem para outro em outra linguagem.

Mostramos a seguir técnicas de transformação muito utilizadas para substituir comandos de iteração com estrutura semelhante ao FOR, por estruturas WHILE e REPEAT. As técnicas são:

a) A iteração:

```
for V <-- A step B until C do S;
```

pode ser substituída pelas seguintes linhas de código:

```
V <-- A
while ((V - C) * sign(B) ≤ 0) do
  begin
    S;
    V <-- V + B
  end;
```

b) A iteração:

```
for I <-- 1 step 1 until N do
    A[I] <-- 0;
```

pode ser substituída usando o comando REPEAT da seguinte forma:

```
I <-- 1;
repeat
  A[I] <-- 0;
  I <-- I + 1;
until I > N;
```

Uma técnica utilizada para eliminação do comando REPEAT é a de substituir este comando pelo comando WHILE. Exemplificando a técnica, temos:

```
repeat S until B;
```

pode ser substituído por:

```
S;
while not B do
  S;
```

Outra técnica conhecida de transformação, é a de substituir recursão por iteração.

A técnica de remover recursão de programas é útil quando se deseja converter programas escritos em uma linguagem que permite trabalhar com recursividade para outra que não permite.

Remover recursão é duplamente importante, pois podemos melhorar o desempenho dos programas bem como economizar espaço de memória durante a execução dos programas.

Standish et al. [26] apresentam o seguinte exemplo para ilustrar a aplicação da técnica de substituir recursão por iteração:

```
procedure FACTORIAL (N); integer N;
    return(if N = 0 then 1 else N*FACTORIAL(N-1)).
```

Ao ser aplicada a técnica de substituição de recursão por iteração, obtemos o seguinte trecho de código:

```
procedure FACTORIAL (N); integer N;
begin
  FACTORIAL <-- 1;
  while N > 0 do
    begin
      FACTORIAL <-- N * FACTORIAL;
      N <-- N - 1
    end
end.
```

Outra técnica de transformação interessante é a técnica de abstração procedural. Standish et al. [26] justificam a aplicação desta técnica citando que: "programadores novatos frequentemente repetem a mesma, ou quase a mesma, sequência de instruções". E ilustra a aplicação da técnica com o exemplo a seguir. Uma maneira de codificar a computação do cosseno do ângulo entre dois vetores V e W pela relação $\frac{V^T W}{\|V\|^2 \|W\|^2}$ seria:

```
S <-- 0
for I <-- 1 step 1 until 3 do
    S <-- S + V[I] * W[I];
R <-- 0
for I <-- 1 step 1 until 3 do
    R <-- R + V[I] * V[I];
T <-- 0
for I <-- 1 step 1 until 3 do
    T <-- T + W[I] * W[I];
RESULT <-- S / (R * T)
```

Pelo reconhecimento de ciclos de repetição comuns podemos substituir o código anterior pelo seguinte:

```
real procedure DOT(X,Y); real array X,Y[1:3];
begin integer I;
  DOT <-- 0;
  for I <-- 1 step 1 until 3 do
      DOT <-- DOT + X[I] * Y[I];
end;
RESULT <-- DOT(V,W) / (DOT(V,V) * DOT(W,W))
```

Mas é sobre o comando GOTO, sem nenhuma dúvida, que são aplicadas as mais variadas e discutidas técnicas de

transformação.

Nas seções seguintes, apresentamos algumas considerações a respeito de técnicas de transformação empregadas com a finalidade de abolir, conservar, ou mesmo introduzir o comando GOTO no escopo dos programas.

2.5 - TRANSFORMAÇÕES DE ESTRUTURA GOTO

Começamos mostrando o quanto o polêmico comando GOTO já foi assunto de discussões e de um grande número de trabalhos técnicos dedicados à questão de sua eliminação.

O comando GOTO volta a ser alvo de discussão nesta seção. Apesar de todas as discussões, o comando GOTO continua presente como comando básico na maioria das linguagens existentes, principalmente nas linguagens para computação científica.

Apresentamos em seguida uma breve resenha histórica do GOTO extraída de Knuth [16].

2.5.1 - Resenha Histórica

O primeiro programador que sistematicamente começou a evitar trabalhar com o comando GOTO foi D. V. Schorre, no início dos anos 60, então professor da Universidade da Califórnia (UCLA), Los Angeles, Estados Unidos.

A convicção de Schorre a respeito do assunto de eliminar GOTO era tamanha, que em 1963, ele criou META II - um subconjunto de ALGOL e eliminou do conjunto de comandos disponíveis o comando GOTO. Este subconjunto tornou-se assim a primeira experiência em eliminar GOTO do conjunto de comandos

básicos disponíveis em uma linguagem de programação.

As idéias de Schorre tiveram repercussão de forma que anos depois Knuth o desafiou a escrever um programa para o problema das oito rainhas sem utilizar o comando GOTO, e Schorre respondeu com um programa usando procedimentos recursivos e variáveis booleanas.

Mas só foi em meados dos anos 60 que Schorre publicou um artigo sobre o assunto declarando que o novo método, o de eliminar GOTO, facilitava os processos de elaboração, modificação e depuração de programas.

Sobre sua experiência com o novo método, Schorre escreveu: "desde o verão de 1960, eu tenho escrito programas em uma forma diferente, usando convenções de margens para indicar o fluxo de controle. Eu julguei nunca achar necessário fazer exceções a estas convenções por usar comando GOTO. Algumas pessoas acharam essa forma diferente, melhor que os fluxogramas que eu tinha desenhado antes, que não eram muito apresentáveis".

Peter Naur, em 1963, também se pronunciou sobre a eliminação do GOTO. Seus comentários tornaram-se os primeiros publicados a respeito do uso de GOTO, intitulado "GOTO STATEMENTS AND GOOD ALGOL STYLE".

Naur escreveu o seguinte: "se você examinar com atenção, verificará com surpresa que frequentemente um comando GOTO que desvia o fluxo de volta para um trecho anterior de um programa (ciclo), realmente é um comando FOR disfarçado. E você estará satisfeito em verificar como a clareza dos algoritmos melhora quando você insere o comando FOR onde ele for possível. ... se a

finalidade (de um curso de programação) é de ensinar programação ALGOL, o uso de fluxogramas causará mais danos que vantagens, em minha opinião".

Começava assim a ser questionado o uso de um dos mais utilizados comandos das linguagens de programação existentes na época. Mesmo assim o GOTO continuou existindo na definição das linguagens criadas posteriormente, mas a sua utilização diminuiu. O surgimento da metodologia estruturada decretou a não utilização do GOTO.

Depois de Naur, George Forsythe também aderiu ao grupo que expurgou o comando GOTO de seus algoritmos submetidos a Communications of the ACM.

Em 1965, Edsger Dijkstra publicava o seguinte comentário sobre o assunto: "dois gerentes de departamentos de programação de diferentes cidades e diferentes graus de conhecimento - um principalmente científico, o outro principalmente comercial - têm me comunicado, independentemente um do outro, suas observações de que a produtividade de seus programadores era inversamente proporcional à densidade de comandos GOTO em seus programas .... Eu tenho feito várias experiências de programação ... em versões modificadas de ALGOL 60 nas quais o comando GOTO foi abolido .... As últimas versões foram mais difíceis de fazê-las: nós estamos familiarizados com o desvio e isto requer algum esforço para esquecê-lo! Em todos os casos tentados, entretanto, o programa sem o comando GOTO tornou-se menor e mais claro".

Peter Landin, poucos meses depois do pronunciamento de Dijkstra no ACM Programming Languages and Pragmatics Conference, escreveu o seguinte: "existe um jogo no qual envolve programas em

ALGOL 60 - reescrevê-los evitando o comando GOTO. Esse jogo visa então produzir programas transparentes, mais fáceis de entender, depurar, modificar e incorporar dentro de um programa maior".

O assunto de eliminar GOTO ganhou tal proporcão que Dijkstra submeteu um pequeno artigo à Communications of the ACM dedicado à discussão do comando GOTO. Para acelerar a publicação do artigo, o editor decidiu publicá-lo como uma carta aberta, com um novo título, "GOTO STATEMENT CONSIDERED HARMFUL". O artigo tornou-se rapidamente bem conhecido; ele expressava a convicção de Dijkstra de que GOTO "devia ser abolido de todas as linguagens de programação de alto nível".

Em 1973, a edição de Datamation publicou 5 (cinco) artigos sobre programação estruturada e eliminação de GOTO.

2.5.2 - Eliminação do GOTO

Bohm & Jacopini [02], em meados dos anos 60, publicaram um artigo no qual demonstravam que qualquer programa podia ser transformado sistematicamente em outro programa. O novo programa executa a mesma computação do original, e pode ser constituído de comandos do programa original usando somente três operações básicas: composição, desvio condicional e iteração, mais alguns possíveis comandos de atribuição e testes em variáveis auxiliares. Assim, em princípio, o comando GOTO pode sempre ser removido.

Partindo do comentário acima de Bohm & Jacopini [02], apresentamos algumas técnicas de transformação que têm como objetivo eliminar o uso do comando GOTO.

Knuth [16] relaciona que na prática, as iterações têm a seguinte forma:

```
A : S;
    if B then goto Z;
    T;
    goto A;
Z :
```

Onde S e T representam sequências de código.

A prática usual para evitar o GOTO em tais iterações é:

- duplicar o código de S, escrevendo:

```
S;
while not B do
  begin
    T;
    S;
  end;
```

- ou duplicar o código de B e fazer um teste redundante escrevendo:

```
repeat
  S;
  if not B then T;
until B;
```

Outras técnicas utilizadas para abolir o uso do comando GOTO são apresentadas a seguir.

A estrutura condicional:

```
    if B goto A;
    S;
A : S1;
```

pode ser substituída pela estrutura:

```
if not B then S;
S1;
```

considerando que não são feitas outras referências ao rótulo A no programa, eliminamos o rótulo A.

A sequência de código:

```
        if B goto X;
        S1;
        goto Y;
    X : S2;
    Y : S3;
```

pode ser substituída pela estrutura condicional:

```
        if B then S2
           else S1;
        S3;
```

ou pela sequência de código:

```
        if not B then S1
                 else S2;
        S3;
```

supondo que, em ambos os casos, não são feitas referências aos rótulos X e Y em outras partes do programa.

Uma outra técnica utilizada para substituir o comando GOTO é a técnica de repetição de código. Em situações do tipo:

```
        .....
        if B goto X;
        S;
        .....
    X : return;
        end;
```

quando o desvio ocorre no início de um programa para uma linha de comando no final é preferível como mostrado a seguir, repetir linhas de comando, no caso somente RETURN, para fazer parte do escopo do comando IF ao invés de criar uma estrutura IF-THEN com escopo do início ao final do programa.

```
.....

if B then
   return;
S;
.....

return;
end;
```

Supondo que não são feitas outras referências ao rótulo X no programa, eliminamos o rótulo X.

Um outro exemplo de utilização da técnica de repetição de código é apresentado a seguir.

Considere a seguinte sequência de código:

```
      if B goto X;
      FACT <-- 0;
      RESULT <-- 1;
      if C goto Y;
X :   NRMAX <-- 1;
      .....

Y :   RESULT <-- AREA * FACT;
```

Após o emprego da técnica de repetição de código obtemos o seguinte trecho:

```
if B
   then begin
          NRMAX <-- 1;
          .....

        end
   else begin
          FACT <-- 0;
          RESULT <-- 1;
          if not C
             then begin
                    NRMAX <-- 1;
                    .....

                  end
        end;
RESULT <-- AREA * FACT;
```

O algoritmo que mostraremos a seguir, extraído de Knuth [16], é parte do bem conhecido esquema "árvore de pesquisa e

inserção".

```
COMPARE : if A[I] < X
            then if L[I] ≠ 0 then I<-- L[I]; goto COMPARE
                             else L[I]<-- J; goto INSERT
            else if R[I] ≠ 0 then I<-- R[I]; goto COMPARE
                             else R[I]<-- J; goto INSERT;
INSERT :  A[J]<-- X;
```

Knuth [16] cita que o exemplo mostrado tem quatro comandos GOTO que podem facilmente ser eliminados introduzindo uma variável booleana que torna-se verdade quando L[I] ou R[I] são iguais a 0. Essa é uma outra técnica de transformação, a técnica de utilização de variáveis booleanas para eliminação de GOTO.

Aplicando então, a técnica de utilizar variáveis booleanas para eliminar GOTO no algoritmo anterior, obtemos o seguinte algoritmo:

```
T<-- true;
while T do
  begin if A[I] < X
          then if L[I] ≠ 0 then I<-- L[I];
                           else L[I]<-- J; T<-- false
          else if R[I] ≠ 0 then I<-- R[I]
                           else R[I]<-- J; T<-- false;
  end;
A[J]<-- X;
```

A técnica de utilizar variáveis booleanas para eliminar GOTO apresenta algumas desvantagens. A maior desvantagem, sem nenhuma dúvida, é a de causar um aumento significativo no tamanho do programa resultante submetido a esta técnica.

Outra técnica utilizada para eliminar o uso de GOTO é citada por Backer [1]. Segundo ele, De Balbine escreveu um programa chamado "Structuring engine" que tenta estruturar programas FORTRAN. O programa de De Balbine emprega uma técnica

não muito utilizada, a de criar subrotinas especiais para eliminar o GOTO.

As técnicas apresentadas são algumas das técnicas existentes para eliminar o uso de GOTO. Cada técnica é apropriada em determinada situação. Entretanto, algumas técnicas quando empregadas, não necessariamente produzem programas mais legíveis que os submetidos à transformação.

2.5.3 - Introdução do GOTO

Existem algumas técnicas que adotam a filosofia de introduzir o comando GOTO nos programas.

Algumas situações justificam a introdução do comando GOTO nos programas embora, como já foi discutido anteriormente, deva ser empregada em último caso. A necessidade de GOTO ocorre, por exemplo, na conversão de estrutura while em Pascal para o padrão ANSI FORTRAN-77 que não tem estrutura semelhante ao while.

Apresentamos a seguir técnicas conhecidas de transformações que podemos incluir neste grupo.

Uma técnica que se enquadra neste grupo, é a técnica de utilização do comando GOTO para substituir comandos de iteração do tipo FOR, WHILE e REPEAT. Por exemplo, o comando de iteração:

```
for V <-- A step B until C do S;
```

pode ser substituído por:

```
begin
     V <-- A;
L1 : if ((V-C) * sign(B) > 0) then goto L2;
     S;
     V <-- V + B;
     goto L1;
L2 :
end;
```

Já o comando de iteração:

```
    while B do S;
```

pode ser substituído pelo trecho de código:

```
    L : if B then
        begin
         S;
         goto L;
        end;
```

O comando de iteração:

```
      repeat S until B;
```

pode ser substituído por:

```
    L : S;
        if not B then goto L;
```

2.5.4 - Programação Estruturada Versus GOTO

Nesta seção apresentamos uma breve discussão sobre o polêmico assunto Comando GOTO versus Programação Estruturada.

O surgimento da Metodologia de Programação Estruturada visando facilitar a escrita e, consequentemente o entendimento de programas, desencadeou a grande discussão sobre a utilização do comando GOTO.

O que muita gente confunde é que Metodologia Estruturada não é só escrever programas sem o GOTO, mas sim, empregar um conjunto de regras com a finalidade de aumentar a produtividade dos programadores, aumentar a legibilidade dos programas produzidos e diminuir os problemas relacionados com o excesso de testes dentro de programas.

Com a aceitação das propostas da Metodologia Estruturada criou-se uma verdadeira barreira contra a utilização do comando GOTO.

A restrição à utilização do GOTO é compreensível porque a utilização indiscriminada do GOTO tende a obscurecer a estrutura dos programas. Assim, na Programação Estruturada evita-se utilizá-lo empregando estruturas fáceis de serem entendidas como WHILE e IF-THEN-ELSE.

A eliminação do comando GOTO dos programas não implica necessariamente em obter programas mais legíveis e melhor estruturados. O que realmente interessa é a estrutura do programa que é o crucial para a legibilidade.

Existem determinadas situações tais como:

```
      IF(A.LE.B) GOTO 10
      B = B+1
      C = C+2
   10 D = D+3
```

onde é perfeitamente óbvia e clara a finaliade do GOTO. Esse aspecto é reforçado por Knuth [16], segundo o qual existem diversas situações na fase de programação onde o comando GOTO se apresenta de forma inofensiva. O comando GOTO era até desejável quando se estava programando em ALGOL ou PL/I.

Knuth [16] escreveu que o programa de Dijkstra, grande divulgador e responsável pela Programação Estruturada, publicado em 1968 sobre Controle de Processos Concorrentes intitulado "SOLUTION OF A PROBLEM IN CONCURRENT PROGRAMMING CONTROL" usava três comandos GOTO, cujas finalidades eram fáceis de entender; dois destes GOTO poderiam desaparecer se o ALGOL 60 tivesse o comando WHILE. Knuth [16] relata que Dijkstra em seu artigo original, intitulado "STRUCTURED PROGRAMMING", não mencionou de nenhuma forma o comando GOTO.

O comando GOTO pode ser utilizado num programa desde que

se evite criar um **espaguete lógico**.

Para o exemplo apresentado anteriormente, é preferível algo do tipo:

```
IF(A.GT.B) THEN
  B = B+1
  C = C+2
ENDIF
D = D+3
```

totalmente adequada às estruturas sintáticas sugeridas pela Metodologia Estruturada.

2.6 - TRANSFORMAÇÕES PARA EXPLORAR CARACTERÍSTICAS DO HARDWARE

Apresentamos nesta seção, outras técnicas de transformação que se preocupam com a eficiência dos programas. Dedicamos uma seção a parte porque estas técnicas apresentam características diferentes dos grupos apresentados anteriormente, pois exploram características do hardware para melhorar o desempenho dos programas.

Hattori & Queiroz [14] citam que o surgimento do PC, estações de trabalhos e supercomputadores fizeram com que os pesquisadores passassem a se preocupar em como explorar de maneira adequada os novos ambientes de computação.

Ainda segundo Hattori & Queiroz [14], as novas arquiteturas de computadores, especialmente aquelas que procuram o paralelismo de operações como os supercomputadores, criaram uma demanda em computação numérica de novos algoritmos básicos ou adequação dos algoritmos conhecidos a essas arquiteturas. Mesmo a nível de software básico e compiladores existe uma demanda de novas abordagens para que os novos recursos de hardware sejam plenamente explorados.

Algumas das características básicas apresentadas pelos supercomputadores são o processamento vetorial com a execução de instruções em duto (pipeline). Instruções em duto dividem a execução de uma função em várias etapas, cada etapa realizando uma parte da função total em um ciclo de máquina. No final de cada ciclo, resultados parciais são passados para a próxima etapa e resultados parciais são recebidos das etapas anteriores.

Segundo Dongarra et al. [7], ao explorar as características básicas dos supercomputadores tais como processamento vetorial com a execução das instruções em duto, pode-se observar uma melhoria significativa na velocidade de execução de operações quando comparado com os computadores convencionais.

Ressaltamos que o objetivo desta seção não é abordar características das arquiteturas dos supercomputadores, mas sim, apresentar técnicas de transformação que adaptam determinados softwares às características de processamento vetorial com execução de instruções em duto dos supercomputadores a fim de conseguir melhorar o desempenho de programas.

Ilustramos a seguir a exploração de recursos de hardware com técnicas de transformação de programas apresentadas por Dongarra et al. [7]. Essas técnicas basicamente se concentram em reestruturar programas que implementam operações comuns da álgebra linear como multiplicação de matriz por vetor e multiplicação de matrizes em FORTRAN, para melhorar o desempenho desses programas.

2.6.1 - Produto de Matriz por Vetor

Seja Y <-- A * X, onde Y é um vetor com m componentes, A é uma matriz m por n, e X um vetor com n componentes. A forma natural de codificar a operação em FORTRAN é:

```
      DO 10 I = 1,M
        Y(I) = 0
        DO 5 J = 1,N
          Y(I) = Y(I) + A(I,J) * X(J)
    5   CONTINUE
   10 CONTINUE
```

conhecida como FORMA IJ.

Nesta forma, referências à matriz A são feitas por linha, mas esta não é a forma mais eficiente para ser executado numa arquitetura vetorial e mesmo numa convencional com paginação porque o acesso aos elementos de A, na computação de Y, não é feito a posições contíguas da memória porque o FORTRAN armazena matrizes por coluna.

Uma solução mais eficiente para o problema é a forma:

```
      DO 10 I = 1,M
        Y(I) = 0
   10 CONTINUE
      DO 25 J = 1,N
        DO 20 I = 1,M
          Y(I) = Y(I) + A(I,J) * X(J)
   20   CONTINUE
   25 CONTINUE
```

conhecida como FORMA JI.

Nesta forma, o fragmento de programa para multiplicação de matriz por vetor mudou. A repetição de Y(I) = 0 embutido no primeiro laço de DO no primeiro fragmento, foi retirado desse laço para constituir um laço independente no segundo fragmento.

Dongarra et al. [7] também recomendam que em geral a forma JI deve ser utilizada. Ele sugere que a forma JI deve ser

sempre usada quando m (número de linhas da matriz) e n (número de colunas da matriz) são aproximadamente do mesmo tamanho. Quando m é muito menor que n, a forma IJ deve ser levada em consideração. Neste caso, a forma IJ manipulará vetores com muitos componentes enquanto que a forma JI, com poucos componentes.

2.6.2 - Multiplicação Matricial

Sejam A e B duas matrizes de dimensões m x n e n x p, respectivamente. O produto de A e B será uma matriz C cuja dimensão será de m x p.

Há seis soluções possíveis para efetuar a operação de multiplicação matriz-matriz. Cada solução apresentada faz acesso aos elementos das matrizes armazenados na memória de forma diferente e dependendo da linguagem utilizada, o melhor desempenho de um programa vai depender do tipo de solução adotada para resolver o problema. As soluções apresentadas são:

(1)
```
      DO 15 I = 1,M
        DO 10 J = 1,P
          C(I,J) = 0
          DO 5 K = 1,N
            C(I,J) = C(I,J) + A(I,K) * B(K,J)
 5      CONTINUE
10    CONTINUE
15  CONTINUE
```

conhecida como FORMA IJK.

Nesta forma o produto de todas as colunas de B com uma linha de A é computado para produzir uma linha em C, um elemento por vez.

```
(2)       DO 15 J = 1,P
            DO 10 I = 1,M
              C(I,J) = 0
              DO 5 K = 1,N
                C(I,J) = C(I,J) + A(I,K) * B(K,J)
        5     CONTINUE
       10   CONTINUE
       15 CONTINUE
```

conhecida como FORMA JIK.

O produto de todas as linhas de A com uma coluna de B é computado para produzir uma coluna em C, um elemento por vez.

```
(3)       DO 15 I = 1,M
            DO 10 J = 1,P
              C(I,J) = 0
       10   CONTINUE
       15 CONTINUE
          DO 30 K = 1,N
            DO 25 I = 1,M
              DO 20 J = 1,P
                C(I,J) = C(I,J) + A(I,K) * B(K,J)
       20     CONTINUE
       25   CONTINUE
       30 CONTINUE
```

conhecida como FORMA KIJ.

Começa-se com todos os elementos da matriz C iguais a 0. Em seguida, uma linha de B é escalada pelos elementos de uma coluna de A e o resultado é usado para atualizar as linhas de C.

```
(4)       DO 15 J = 1,P
            DO 10 I = 1,M
              C(I,J) = 0
       10   CONTINUE
       15 CONTINUE
          DO 30 K = 1,N
            DO 25 J = 1,P
              DO 20 I = 1,M
                C(I,J) = C(I,J) + A(I,K) * B(K,J)
       20     CONTINUE
       25   CONTINUE
       30 CONTINUE
```

conhecida como FORMA KJI.

Também começamos com todos os elementos da matriz C iguais a 0. Em seguida, uma coluna de A é escalada pelos elementos de uma linha de B e o resultado é usado para atualizar as colunas de C.

```
(5)       DO 30 I = 1,M
            DO 25 J = 1,P
              C(I,J) = 0
      25    CONTINUE
            DO 20 K = 1,N
              DO 15 J = 1,P
                C(I,J) = C(I,J) + A(I,K) * B(K,J)
      15      CONTINUE
      20    CONTINUE
      30  CONTINUE
```

conhecida como FORMA IKJ.

Começamos zerando os elementos da linha da matriz C que será gerada. Em seguida, fixa-se um elemento de uma linha de A e multiplica-se este elemento por uma linha de B. Os resultados parciais são atualizados na matriz C. Logo após, pega-se o elemento seguinte na mesma linha de A e o multiplica pela linha seguinte à anteriormente utilizada de B. O processo é repetido até ser gerada uma linha completa na matriz C.

```
(6)       DO 30 J = 1,P
            DO 25 I = 1,M
              C(I,J) = 0
      25    CONTINUE
            DO 20 K = 1,N
              DO 15 I = 1,M
                C(I,J) = C(I,J) + A(I,K) * B(K,J)
      15      CONTINUE
      20    CONTINUE
      30  CONTINUE
```

conhecida como FORMA JKI.

O processo de geração da matriz C é o mesmo utilizado para a forma IJK. Só que nesta forma utilizam-se as colunas e não

as linhas durante o processo de atualização dos elementos de C.

Quando se emprega a linguagem FORTRAN para resolver multiplicação de matriz-matriz, a melhor forma é a forma KJI pois armazena os elementos por coluna, e em ambientes FORTRAN operações sobre colunas é preferível às operações sobre linhas devido o acesso a posições contíguas na memória.

Já em ambientes Pascal e C é preferível a forma KIJ que executa as operações de acesso e armazenamento de elementos de matrizes por linha.

Seria interessante efetuar automaticamente transformações como as mostradas anteriormente, e assim ter a flexibilidade de escrever trechos de programas de maneira natural e costumeira como os apresentados para multiplicação matriz-vetor, conhecida como FORMA IJ, e multiplicação matriz-matriz, conhecida como FORMA IJK, e depois converter para a forma adequada a uma dada arquitetura.

3 - SISTEMAS TRANSFORMACIONAIS

3.1 - INTRODUÇÃO

Várias técnicas foram criadas nos últimos anos com a finalidade de dar apoio ao processo de desenvolvimento de software, facilitando assim o trabalho de escrever programas.

Com o passar dos anos, chegou-se à conclusão de que programas "grandes" geralmente se tornam complexos, e o pior, consomem muito tempo na fase de manutenção.

Surgiu a idéia de estudar as várias técnicas que um programador geralmente aplicava durante a fase de manutenção, e juntá-las para serem manipuladas por um software como um sistema automático de transformação de programas, que realizaria as transformações necessárias. O sistema de transformação poderia então, realizar otimizações a nível fonte ao invés das otimizações serem feitas manualmente.

Na década de 70 foram criados os primeiros Sistemas de Desenvolvimento de Software, pelos quais o interesse é cada vez mais crescente na área de programação de computadores. Dentre os Sistemas de Desenvolvimento de Software destacam-se os Sistemas Transformacionais.

Em 1974, Knuth [16] cita que os sistemas de transformação de programas prometiam ser uma ferramenta de futuro promissor ajudando os programadores a melhorarem seus programas.

Os Sistemas Transformacionais vêm se tornando assim, ferramentas importantes no processo de desenvolvimento de software.

3.2 - APLICAÇÃO / VANTAGENS

A meta mais comum de um Sistema Transformacional é de servir de apoio na modificação de programas. Esse apoio facilita a otimização de programas, a produção de versões de programas em uma nova precisão distinta daquela versão existente e a adaptação de programas para ser executado em um novo ambiente de computação.

Além de apoio à modificação de programas, outros fatores como redução de custos de desenvolvimento de software, maior confiabilidade dos programas produzidos e considerável economia de tempo, impulsionaram a utilização dos Sistemas Transformacionais no processo de desenvolvimento de software.

Uma outra aplicação dos Sistemas Transformacionais consiste em converter programas escritos em uma linguagem para programas equivalentes em uma outra linguagem. É comum nos dias de hoje o interesse pela conversão de programas. O processo de conversão consome tempo quando feito manualmente. A conversão de programas se processa da seguinte forma: o Sistema Transformacional aceita como entrada um programa fonte em uma linguagem e automaticamente executa todas as traduções, retornando como saída um programa em uma outra linguagem. Portanto, em vez de gerar um programa objeto, o Sistema Transformacional retorna um programa fonte na linguagem destino.

A utilização do computador assim facilita o trabalho de conversão. A tarefa de conversão era considerada como uma das tarefas de programação mais tediosas, embora comum.

Wolberg & Rafal [28] afirmam que experiências no

desenvolvimento de uma variedade de Sistemas Transformacionais os levaram a concluir que um primeiro passo necessário é de transformar os programas a um formato padrão antes de serem submetidos aos Sistemas Transformacionais. Sistemas para colocar programas em um determinado padrão são uma outra aplicação dos Sistemas Transformacionais.

Wolberg & Rafal [28] desenvolveram o Conversor CONVERT que tem como finalidade transformar códigos fontes para formatos padrão.

Os Sistemas Transformacionais também são ferramentas úteis para facilitar o desenvolvimento de programas portáteis ou transportáveis.

Segundo Ryder [23], um software deve ser portátil para assegurar o seu uso por um grande grupo de pessoas. Um dos métodos para atacar o problema de portabilidade é definir cuidadosamente um subconjunto de alguma linguagem de programação disponível em uma variedade de computadores. Ela criou o Verificador PFORT, um sistema que verifica se um programa FORTRAN só utiliza construções do subconjunto portátil do American National Standard FORTRAN também chamado PFORT.

Segundo Medeiros [19], para solucionar os problemas de portabilidade, uma solução frequentemente adotada consiste em efetuar manualmente todas as adaptações até que o software produza os resultados desejados. Uma outra solução, modernamente adotada para o mesmo problema, é a utilização de Sistemas Transformacionais que, em geral, transformam parcialmente programas FORTRAN previamente padronizados.

O problema de portabilidade é muito comum em áreas como

Matemática Computacional. Em Matemática Computacional é de extrema importância que os programas produzidos possam ser executados em diversas máquinas sem nenhuma alteração, conseguindo assim satisfazer o atributo portabilidade, ou mesmo que possam ser executados com um mínimo de alterações.

Maher & Sleeman [18] resumem muito bem a utilidade dos Sistemas Transformacionais afirmando que eles constituem uma excelente ferramenta que têm várias aplicações em potencial. Estas aplicações incluem:

1) transformar automaticamente programas escritos em uma linguagem para outros programas em uma linguagem melhor estruturada. Por exemplo, FORTRAN para RATFOR;
2) transformar programas para serem executados em uma instalação padrão;
3) otimizar programas.

Standish et al. [25] reforçam a idéia de Maher & Sleeman [18] citando que também vêem diversas vantagens no uso de Sistemas Transformacionais, tais como

- acoplando um determinado programa a um programa para avaliar desempenho, podemos isolar e então transformar aquelas partes do programa cuja eficiência é mais crítica.
- uma transformação mecânica tenta demonstrar a equivalência de programas e transmitir confiança na transformação que estaria sujeito a erros se fosse executada manualmente.
- transformações mecânicas permitem experimentar alterações nas representações básicas dos programas, o que seria proibitivamente dispendioso ou não confiável se feitas manualmente.

Não podemos esquecer que os Sistemas Transformacionais podem enfrentar dificuldades no processo de transformação, devido à grande complexidade de muitos programas.

3.3 - UM EXEMPLO DE SISTEMA TRANSFORMACIONAL

Apresentamos nesta seção uma descrição de um conhecido Sistema Transformacional na tentativa de melhor mostrar as vantagens em utilizar Sistemas Transformacionais.

O sistema que vamos apresentar é o Sistema TAMPR, descrito por Boyle [4], Boyle et al. [5], Boyle & Matz [6], Dritz [8], Hague [10], e Hattori & Queiroz [14].

3.3.1 - TAMPR

Ambiente de Desenvolvimento

O Sistema TAMPR (Transformation-Assisted Multiple Program Realization) foi desenvolvido a partir de 1973 por Boyle, Dritz e outros no Argonne National Laboratory, Argonne, Illinois, Estados Unidos, um ambiente não comercial de pesquisa sobre metodologias e ferramentas para desenvolvimento de software numérico.

O Sistema TAMPR foi concebido pela necessidade de aperfeiçoamento do seu predecessor que foi o Sistema Especializador-Generalizador.

O Sistema Especializador-Generalizador foi projetado para automatizar a tarefa de conversão de precisão de um conjunto de programas escritos usando aritmética de precisão dupla em

FORTRAN para programas equivalentes em precisão simples e para manter uma única versão mestre dos programas em forma executável, que continha comandos de controle bastante rudimentares.

O Sistema Especializador-Generalizador em geral funcionava bem, mas apresentava algumas limitações. Além de só converter a precisão dos programas de dupla para simples, se o Sistema fosse utilizado para converter um programa arbitrário que não tivesse sido escrito obedecendo a determinados padrões, ele poderia efetuar algumas mudanças indevidas. O projeto do Sistema não permitia a adição de novas funções.

Como resultado das limitações do Sistema Especializador-Generalizador e do interesse no desenvolvimento de pacotes em áreas mais complexas da Análise Numérica (do que Álgebra Linear), e no desenvolvimento descentralizado de software, fortaleceu a idéia de desenvolvimento de um novo software mais compacto e poderoso, que apresentasse novas características como formatação automática de programas e capacidade para especificação e execução de mudanças (de manutenção) que devessem ser efetuadas na maioria dos programas de um projeto. Como consequência dessas novas idéias, surgiu o Sistema TAMPR.

O Sistema TAMPR foi assim desenvolvido para produzir múltiplas realizações de uma simples versão protótipo de um programa. É um Sistema de programação automática concebido para assistir a área de análise numérica na preparação, teste e refinamento de subrotinas numéricas para diferentes computadores.

Objetivos

1 - o Sistema deveria reduzir tanto quanto possível, a quantidade de programas armazenados na forma mestre; para tanto, deveriam ser automatizadas a derivação de algumas versões dependentes de máquina e a derivação de versões em precisões simples e dupla;

2 - deveria permitir a um programador produzir programas bem estruturados. Para escrever programas transportáveis, o Sistema não deveria exigir do programador um conhecimento detalhado do ambiente destino;

3 - deveria aceitar programas FORTRAN comuns, executáveis em alguma máquina, e convertê-los para a forma mestre.

4 - deveria executar a derivação das versões e outras operações de maneira mais confiável possível;

5 - deveria ser flexível o suficiente para satisfazer requisitos ainda desconhecidos como a formatação de programas de acordo com especificações ainda desconhecidas.

Abordagem

No Sistema TAMPR existe uma versão (não executável) dos programas chamada de "forma canônica abstrata". Além disso, o Sistema também contém um conjunto de processadores (funções de produção) que operam sobre a forma canônica abstrata para produzir as várias versões de um mesmo programa.

A operação do Sistema TAMPR consiste então na aplicação de uma função de produção a um protótipo de programa para obter uma versão particular daquele programa. As funções de produção são genéricas e podem ser aplicadas em sequência para a obtenção de transformações compostas confiáveis.

O fluxo de informação no Sistema pode ser resumido pelo seguinte diagrama:

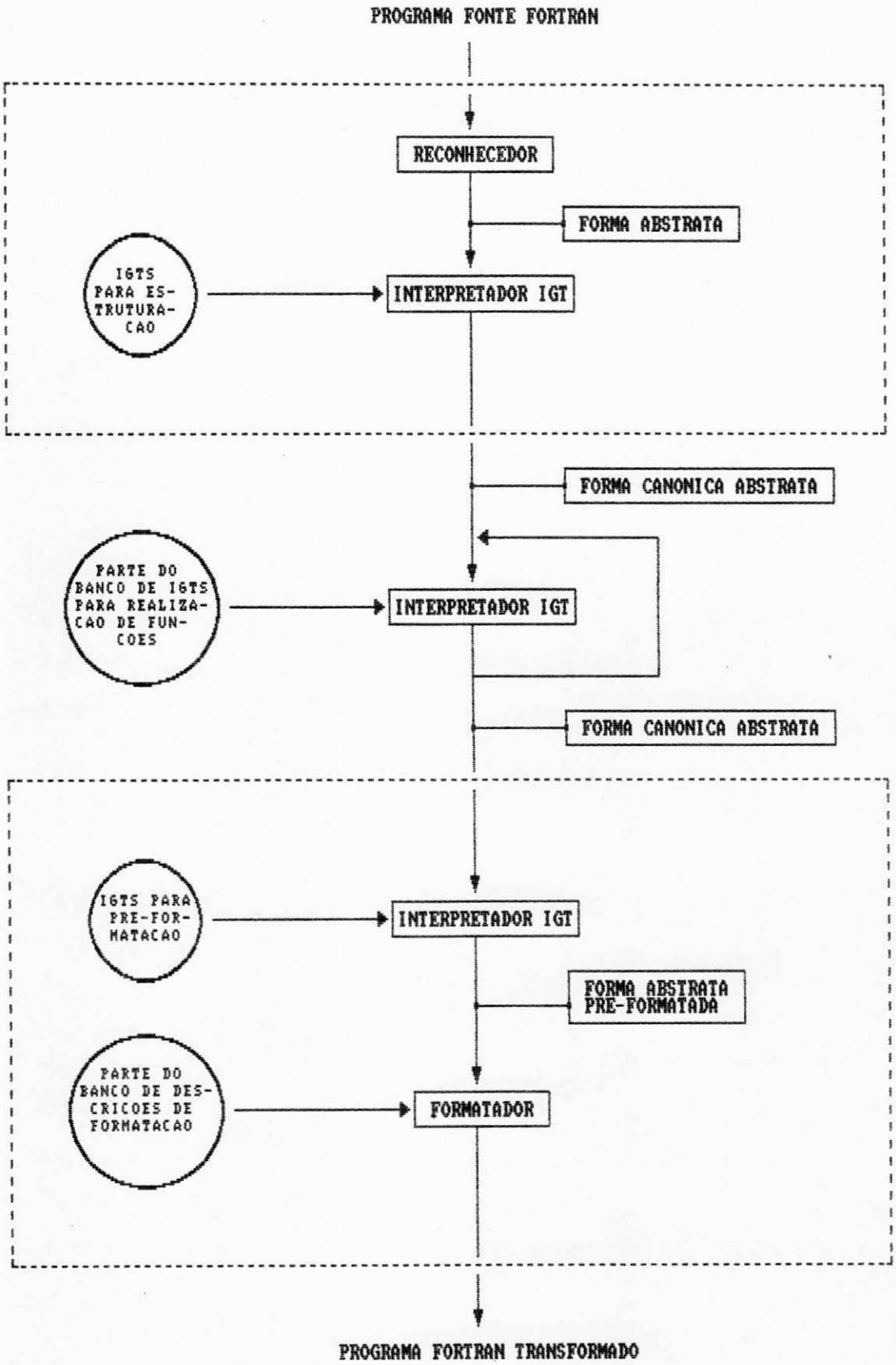
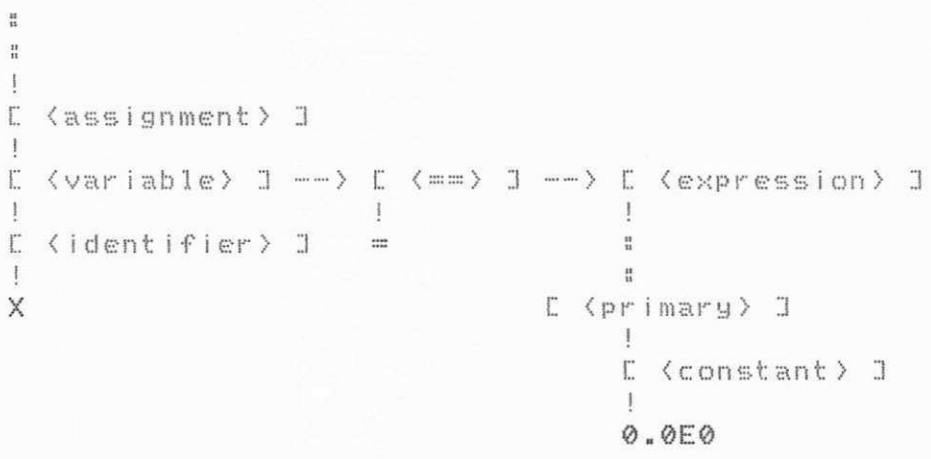


FIG. 1 - DIAGRAMA DE FLUXO DE INFORMAÇÃO DO SISTEMA TAMPR

Um programa fonte entra no Sistema e é processado pelo "Reconhecedor" que converte o programa na forma concreta para a forma abstrata. A forma abstrata é essencialmente a árvore de **parsing** do programa, que é uma representação sintática explícita do programa na forma de árvore construída de acordo com uma gramática BNF fornecida.

Suponha que estamos reconhecendo um programa FORTRAN que contém o seguinte comando de atribuição: X = 0.0E0. O correspondente fragmento da árvore de parse para este comando de acordo com a gramática poderia ser:



onde [<x>] é um nó da árvore de parse de tipo gramatical <x>.

Assim formada, a forma abstrata é imediatamente processada pelo componente de Transformações Intra-Gramaticais (IGTs) do Sistema, que utiliza um conjunto de transformações para clarificação de estruturas. Isto produz a forma canônica abstrata.

O uso da forma canônica pode ser ilustrado pelo exemplo que vamos apresentar a seguir. Considere o seguinte fragmento de programa:

```
      DO 10 I = 1,N
      ¦
      ¦
      DO 10 J = 1,N
      ¦
      ¦
      IF(A.LT.B) GOTO 10
      ¦
      ¦
   10 X(I,J) = X(I,J) * B
```

que é inteiramente equivalente ao fragmento:

```
      DO 30 I = 1,N
         ¦
         ¦
         DO 20 J = 1,N
            ¦
            ¦
            IF(A.LT.B) GOTO 10
            ¦
            ¦
   10       CONTINUE
            X(I,J) = X(I,J) * B
   20    CONTINUE
   30 CONTINUE
```

O rótulo 10 no primeiro fragmento dá uma idéia de dupla função, parecendo representar o começo do comando X(I,J)=X(I,J)*B quando ele ocorre no comando IF, e o final do comando de atribuição quando ele ocorre no comando DO. Os dois fragmentos de programas são representados a seguir pelo simples fragmento na forma canônica abstrata, usando ":" para delimitar definições de rótulos e ";" para comandos, é:

```
      DO I = 1,N;
         ¦
         DO J = 1,N;
            ¦
            ¦
            IF(A.LT.B) GOTO 10;
            ¦
            ¦
         10:CONTINUE;
            X(I,J) = X(I,J) * B;
      END;
   END;
```

Além disso, se possível, o fragmento:

```
IF(A.LT.B) GOTO 10;
|
|
10:CONTINUE;
```

seria convertido para:

```
IF(A.GE.B) THEN;
|
|
END;
```

Uma vez que o programa está na forma canônica abstrata, ele deve ser processado zero ou mais vezes pelo IGT. A forma canônica abstrata final é então processada pelo "Formatador" e reconstruída na forma executável.

Com vistas à obtenção de flexibilidade (objetivo (5)), o Sistema TAMPR foi projetado para possuir alguns processadores programáveis. Portanto, o Componente Transformacional, o Formatador e o Reconhecedor são programáveis.

O Reconhecedor, programável na gramática BNF usada para gerá-lo, converte, por exemplo, um programa FORTRAN para a forma abstrata.

O Componente Transformacional é dirigido por um conjunto de transformações Intra-Gramaticais (IGTs).

O Interpretador de Transformações é o responsável pelas passagens das formas abstratas para outras formas, também abstratas, de acordo com as especificações escritas na linguagem dos IGTs, pois garantem a correção sintática do programa transformado.

Para simplificar a descrição e aplicação de funções de produção como IGTs, a forma abstrata é inicialmente convertida para a forma abstrata canônica pela simplificação de

transformações de clarificação de estruturas. Um efeito destas transformações é reduzir a complexidade de programas substituindo construções complexas por construções equivalentes mais simples e preservando a semântica da construção. Um exemplo é o reconhecimento de um ninho de DOs que têm o mesmo número de comando de fim de escopo. Estes DOs são assim substituídos por DOs equivalentes, onde cada comando DO tem seu próprio comando de fim de escopo. Um outro exemplo típico é a substituição de algumas utilizações do comando GOTO por construções equivalentes melhor estruturadas e sem rótulos. Vale ressaltar que para alcançar essas finalidades, a gramática básica usada é a gramática de uma extensão estruturada de FORTRAN.

Já o Formatador é um processador programável em Linguagem de Controle de Formato (FCL - Format Control Language) que converte da forma canônica abstrata para FORTRAN executável. Ele especifica o formato (espaçamento, margens, continuação de linhas) para o programa FORTRAN. Permite ainda ações como renomear variáveis.

O desenvolvimento do Sistema TAMPR trouxe como uma das principais contribuições a idéia de funções de produção. As funções de produção, entre outras vantagens, tornam possível o transporte de programas para máquinas cujas implementações de FORTRAN violam o padrão e permitem derivar mais de um programa a partir de uma única versão mestre.

Hattori & Queiroz [14] resumem a ação do Sistema TAMPR citando que o Sistema preserva a correção dos programas e para isso, constrói a árvore de parsing do programa permitindo ao usuário analisar e transformar a árvore e finalmente convertê-la

de volta para um programa equivalente em FORTRAN.

Ainda segundo Hattori & Queiroz [14], TAMPR examina as regras de transformações para garantir que as transformações que ela especifica não altera a função do programa objeto de transformação. Além disso, por causa da liberdade do usuário em analisar e transformar a árvore, o Sistema tem praticamente recursos ilimitados.

Os maiores defeitos do Sistema TAMPR, segundo Hattori & Queiroz [14], são a sua complexidade e a exigência que o usuário seja habilidoso em tratar com o formalismo matemático.

4 - O FORTRAN-77

4.1 - INTRODUÇÃO

A dificuldade de programar em linguagem que o computador entendia nos primórdios da computação, motivou a criação das chamadas linguagens de programação de alto nível. Estas linguagens vieram oferecer maior facilidade de programação aos usuários, tornando-se assim um dos fatores chaves para a disseminação do uso dos computadores.

Em meados de 1956, surgiu a primeira linguagem de programação de alto nível, a linguagem FORTRAN, que se tornou uma das mais populares e difundidas linguagens de programação na comunidade de ciência e tecnologia.

Desde sua criação até nossos dias, a linguagem FORTRAN sofreu consideráveis transformações para atender da melhor forma possível às necessidades de seus usuários, oferecendo-lhes cada vez mais recursos avançados de programação.

Hehl [15] cita que inicialmente a linguagem FORTRAN ficou conhecida como FORTRAN-I. Em 1960, surgiu o chamado FORTRAN-II e em 1964 foi lançado o FORTRAN-IV. Com o surgimento de várias versões da linguagem, o American National Standards Institute (ANSI) padronizou em 1966 duas versões oficiais do FORTRAN chamadas de FORTRAN IV Básico (Basic FORTRAN IV) e FORTRAN IV Avançado (Advanced FORTRAN IV). Apesar dessas versões representarem diferentes estágios de desenvolvimento da linguagem, as versões são muito semelhantes.

Com a introdução do conceito de programação estruturada, o FORTRAN sofreu uma nova modificação. O ANSI estabeleceu um novo

padrão para a linguagem FORTRAN, conhecido como FORTRAN-77, também chamado FORTRAN estruturado. Hehl [15] cita que apesar do FORTRAN-77 ser a versão mais moderna da linguagem FORTRAN, ele mantém todas as características das versões anteriores.

Desde meados dos anos 60 quando um grupo de fabricantes de computadores estabeleceu uma definição da linguagem FORTRAN chamada o ANSI FORTRAN na tentativa de padronização da linguagem, a indústria de computadores tem aderido aos padrões do ANSI. Mesmo assim surgiram várias implementações com extensões da nova linguagem introduzidas por alguns fabricantes. Hehl [15] cita como principais extensões da linguagem FORTRAN-77: VS FORTRAN (IBM), FORTRAN-77 (DIGITAL), FORTRAN 5 (CDC) e FORTRAN 77 (BURROUGHS). Todas essas versões implementam o FORTRAN padronizado e adiciona extensões.

O FORTRAN-77 trouxe a flexibilidade de utilização da estrutura IF-THEN-ELSE, um dos recursos que facilita a programação estruturada, e outros recursos de programação que serão comentados na seção seguinte. Isso motivou o interesse em transformar programas escritos em FORTRAN-66 para FORTRAN-77, tendo em vista que a versão mais antiga impôs, por falta de recursos, construções condenadas pela programação estruturada.

4.2 - NOVAS CARACTERÍSTICAS DO FORTRAN

Apresentamos nesta seção algumas características da linguagem FORTRAN incluídas na nova versão, o FORTRAN-77, e mostramos na seção seguinte como explorar algumas destas características transformando programas FORTRAN-66 para FORTRAN-77.

Hehl [15] descreve os elementos, as regras de programação e os comandos da linguagem de acordo com o documento X3.9-1978, conhecido como padrão FORTRAN-77. Ele também descreve algumas características das principais implementações da linguagem. Portanto, a descrição de algumas das novas características da linguagem que vamos apresentar a seguir é extraída de Hehl [15].

As principais novidades da nova versão foram o aparecimento dos comandos IF-THEN-ELSE, OPEN, INQUIRE, CLOSE, PRINT e PARAMETER, uma nova forma do comando READ e alterações no comando DO.

Começamos mostrando alguns recursos avançados de operações de entrada e saída de dados contidos na nova versão padrão que facilitam a manipulação e o uso de arquivos.

Os comandos OPEN, INQUIRE e CLOSE são usados para abrir, indagar condições correntes de arquivos e unidades, e fechar certos atributos de arquivos usados nas operações de entrada ou saída de dados, respectivamente. Não é necessário utilizar estes comandos quando se trabalha com unidades padrões de entrada e saída; só é necessário quando se trabalha com arquivos em outros dispositivos como fita magnética ou disco magnético.

O comando PRINT tem a mesma função do comando WRITE, com a diferença de só ser permitido utilizar com arquivos sequenciais.

O comando READ agora pode ser utilizado na forma READ * [,list], onde * é o especificador de unidade padrão de entrada. Portanto, não é mais necessário especificar a unidade de entrada

quando se está trabalhando com unidades que são usadas sob o modo de acesso sequencial tais como leitora de cartões, terminais e fitas magnéticas.

Outra novidade foi o aparecimento do comando PARAMETER. O comando PARAMETER é utilizado para representar uma constante simbolicamente em vez de representá-la por um valor. Uma vez definido o valor de uma constante simbólica, não se pode mais redefiní-la. Por exemplo, o comando PARAMETER (PI = 3.14159) declara PI como uma constante simbólica tendo o valor 3.14159. Tem a mesma função do comando CONST na linguagem Pascal.

O comando DO na nova versão ficou mais flexível. A forma geral do comando continuou sendo:

DO n [,] v = e1, e2 [, e3] onde:

- o que estiver entre colchetes é opcional;
- n é o número de um comando executável chamado comando terminal do laço de DO. Geralmente este comando é o CONTINUE;
- v é, usualmente, uma variável inteira, chamada variável de controle.
- e1, e2, e3 são os valores da variável de controle do DO e usualmente são obtidos da avaliação de expressões inteiras.
- e1 é o valor inicial de v;
- e2 é o valor final de v;
- e3 é o valor de incremento de v. Se omitido, é assumido o valor 1.

As novidades quanto ao comando DO são: v, a variável de controle do DO, agora pode ser do tipo inteiro, real ou dupla precisão; e1, e2 e e3, parâmetros da variável de controle do DO, podem também ser expressões do tipo real ou dupla precisão; e o

incremento de e3 pode ser negativo (na verdade ocorre um decremento). Exemplos de comandos DO são:

```
DO 100 J = 1, 50, 2
DO 200 L = 50, -2, -2
DO 300 M = 0.5, 10.0, 0.5
DO 400 N = 1, 20
```

Outra novidade da nova versão é a maior flexibilidade na passagem de parâmetros que são conjuntos (arrays). Por exemplo:

```
SUBROUTINE MATRIX (A,NL)
REAL  A(NL,*)
```

O valor da dimensão não é passado como um argumento, porém é determinado pelo número de elementos declarado na unidade ativadora.

A mudança mais radical e importante na versão FORTRAN-77 foi o aparecimento da estrutura condicional IF-THEN-ELSE. A estrutura condicional veio facilitar a construção de programas mais legíveis. Na próxima seção exemplificamos o uso da estrutura IF-THEN-ELSE.

A nova versão também adotou a filosofia de não trabalhar com recursividade e apontadores e continuou com a pobreza de estruturas de controle pela ausência de uma estrutura do tipo WHILE.

Uma solução para quem deseja utilizar o comando WHILE na nova versão é simular a sua estrutura da seguinte forma:

```
n IF (condição) THEN
     comandos
     GOTO n
  ENDIF
```

Como essa é uma estrutura não comum e obscura, sua utilização pode tornar os programas bem menos legíveis.

A pobreza de estruturas de controle em FORTRAN, segundo Hattori & Queiroz [14], justifica a tentativa de utilizar extensões da linguagem. Por outro lado a transportabilidade impõe a utilização de um subconjunto de construções que represente uma intersecção das implementações disponíveis.

Um subconjunto conhecido da linguagem FORTRAN é o PFORT. O subconjunto PFORT foi definido por Ryder [23], que criou um verificador, também chamado PFORT, que testa a aderência de um programa às suas restrições.

Hague [10] apresenta algumas características do RATFOR (Rational FORTRAN) que se tornou uma conhecida extensão da linguagem FORTRAN. RATFOR tem como objetivos principais facilitar a escrita e produzir programas mais legíveis e melhor estruturados. Esses objetivos são atingidos porque o RATFOR fornece melhores estruturas de controle que a linguagem FORTRAN padrão.

Algumas características do RATFOR são: adotar a filosofia de formato livre, ou seja, comandos podem aparecer em qualquer parte de uma linha de comando; usar como caráter de continuação o símbolo # que pode estar em qualquer parte de uma linha de comando, só que antes do comando desta linha; e apresentar comandos tais como: IF (condição) comandos ELSE comandos (a palavra THEN é opcional); WHILE (condição) comandos; FOR (valor inicial; condição; incremento) comandos; e REPEAT comandos UNTIL (condição) que facilitam a produção de programas

mais legíveis. A prática comum é manter os programas escritos em RATFOR porque facilita a manipulação pelo programador. Antes de compilar, esses programas são pré-processados e transformados em PFORT para manter a portabilidade ou transportabilidade.

4.3 - MODELAGEM DO FLUXO DE CONTROLE

Programas em FORTRAN-66 eram obrigados a usar frequentemente o comando GOTO porque a linguagem não tinha comandos de iteração como WHILE nem estruturas do tipo IF-THEN-ELSE. Com a nova versão do FORTRAN apareceu a estrutura IF-THEN-ELSE e assim, tornou-se interessante reestruturar programas FORTRAN-66 transformando os trechos desses programas onde a estrutura IF-THEN-ELSE poderia ser usada, a fim de melhorar a sua legibilidade.

A pobreza de estruturas de controle da nova versão do FORTRAN-77, algumas vezes, torna complicado eliminar o comando GOTO na transformação de programas FORTRAN-66 para FORTRAN-77.

Para resolver o problema de eliminação do comando GOTO tentamos encontrar uma ferramenta de modelagem que nos respondesse uma questão concreta: quando uma construção IF (...) GOTO pode ser transformada e quando não pode ser transformada para uma estrutura IF-THEN ou IF-THEN-ELSE com a eliminação do GOTO.

A ferramenta de modelagem testada para resolver os problemas relacionados com a eliminação do comando GOTO foi o grafo. A planaridade do grafo parecia ser a resposta à questão acima.

Escolhida a ferramenta de modelagem, passamos a combinar

a ferramenta com simbologias para representar o fluxo de controle de um programa na linguagem FORTRAN e assim facilitar uma melhor modelagem do fluxo dos programas.

As simbologias utilizadas para representar os comandos da linguagem FORTRAN foram:

○ para representar o comando DO; □ para representar um comando IF (...) GOTO ou um comando GOTO "puro"; — para representar um comando CONTINUE; ↙ ou ↘ para indicar o destino de um comando IF (...) GOTO ou um comando GOTO "puro"; ↗ ou ↖ para indicar um ciclo de um comando DO; e ↓ para representar os demais comandos disponíveis na linguagem FORTRAN. Assim, um trecho de programa escrito em FORTRAN-66 da forma:

```
      .....

      IF (ERRSUM .LE. ERRBND) GOTO 15
      RLIST2(1) = RESULT
      .....
      DO 10 K = 1,LAST
         RESULT = RESULT + RLIST(K)
   10 CONTINUE
      .....
      LIMIT = 1
   15 CONTINUE
      EXTRAP = .FALSE.
      .....
```

assumiria a seguinte forma quando modelado:

Considerando aqui que o teste de fim de um comando DO é feito no fim do seu escopo.

Algumas restrições foram impostas quando da transformação dos programas FORTRAN-66 para programas FORTRAN-77, na tentativa de obter uma maior uniformidade dos programas transformados. As restrições impostas foram as seguintes:

- os programas transformados devem manter a integridade da linguagem FORTRAN;
- o número de IFs do programa transformado deve ser igual ou inferior ao número de IFs do programa escrito em FORTRAN-66;
- os programas a serem transformados devem satisfazer uma disciplina de programação previamente definida que proibe certas construções como desvio de volta com GOTO;
- durante o processo de transformação, pode-se usar repetição de blocos de código fonte já existentes, mas fica proibida a criação de novas linhas de código; e
- a ferramenta de modelagem é o grafo.

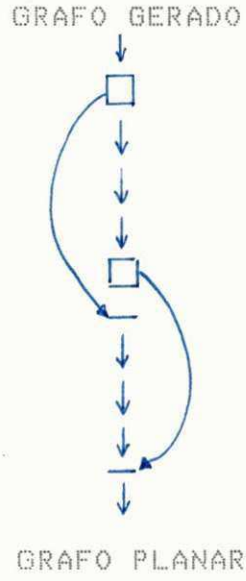
Consideradas as restrições impostas, um outro tipo de construção que não pode acontecer é apresentado a seguir, pois a disciplina de programação adotada não permite empregar dois comandos DO com mesmo rótulo. Alguns detalhes das disciplinas de programação são apresentados no capítulo 5.

```
      ......
      DO 20 I = 1,NINT
         IP1 = I + 1
         DO 20 J = IP1,NINTP1
           IF (PTS(I) .LE. PTS(J) GOTO 20
           TEMP = PTS(I)
           PTS(I) = PTS(J)
           PTS(J) = TEMP
   20 CONTINUE
      ......
```

Unificado o processo de modelagem e levadas em consideração as restrições impostas, partimos para a análise da modelagem por grafo. Numa primeira análise pensávamos que se os grafos para trechos de programas que utilizavam construções IF (...) GOTO fossem planares, os trechos de programas poderiam ser sempre transformados para estruturas IF-THEN ou IF-THEN-ELSE com a eliminação do comando GOTO.

Mostramos a seguir, um exemplo de construção FORTRAN-66 para a qual é gerado um grafo planar e a eliminação do GOTO é possível. A transformação do exemplo abaixo para FORTRAN-77 é mostrado na subseção 4.4.2, nos exemplos (8) e (8.1).

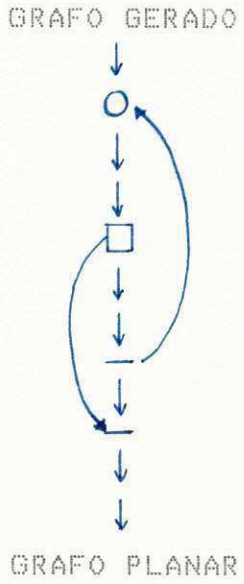
```
      .....
      IF (EXTRAP) GOTO 100
      ALIST(LAST) = A2
      BLIST(MAXERR) = B1
      BLIST(LAST) = B2
      GOTO 110
  100 CONTINUE
      ALIST(MAXERR) = A2
      ALIST(LAST) = A1
      BLIST(LAST) = B1
  110 CONTINUE
      KSGN = -1
      .....
```



Um outro exemplo de construção FORTRAN-66 para a qual também é gerado um grafo planar e a eliminação do GOTO também é possível, é apresentado a seguir:

```
      ......
      DO 130 K = ID, JUPBND
         MAXERR = IORD(K)
         ERRMAX = ERRMAX + ELIST(MAXERR)
         IF (KTMIN .LE. 5) GOTO 160
         ......
         NRMAX = NRMAX + 1
130   CONTINUE
      ......

160   CONTINUE
      KTMIN = KTMIN + NEV
      ......
```



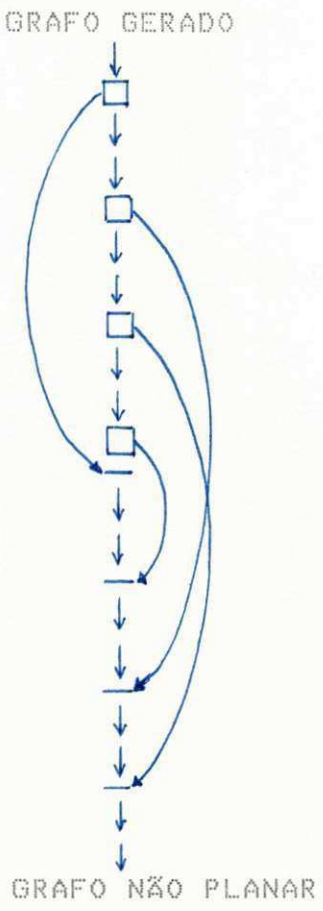
Neste exemplo, a substituição do comando IF (...) GOTO 160 no trecho de programa por uma estrutura IF-THEN é facilmente conseguida ao utilizar a técnica de remoção de comandos invariantes e negação de relações lógicas. Este exemplo de transformação pode ser encontrado também na subseção 4.4.2, no exemplo (12).

Outros exemplos de construção para os quais são gerados grafos planares e todo comando GOTO é passível de transformação são apresentados na subseção 4.4.2.

Por outro lado pensávamos que quando os grafos gerados fossem não planares, os trechos de programas que utilizavam o comando GOTO não poderiam ser transformados para estruturas IF-THEN ou IF-THEN-ELSE com a eliminação do comando GOTO. Um exemplo dessa situação é apresentado a seguir.

```
      .....

      IF (LEVCUR + 1 .LE. LEVMAX) GOTO 100
      SMALL = SMALL * 5.0E-01
      .....
      IF (NUMRL2 .LE. 2) GOTO 120
      NUMRL2 = NUMRL2 + 1
      .....
      IF (AREA .EQ. 0.0E+00) GOTO 130
      ERTEST = ERRBND
      .....
      GOTO 110
  100 CONTINUE
      ERLARG = ERRSUM
      .....
  110 CONTINUE
      CORREC = ERLARG
      .....
  120 CONTINUE
      NRMAX = 2
      .....
  130 CONTINUE
      KTMIN = 0
      .....
```



A transformação do trecho de programa escrito em FORTRAN-66 para um trecho equivalente em FORTRAN-77 com a eliminação do comando GOTO poderia ser algo da forma:

```
.....
IF (LEVCUR + 1 .LE. LEVMAX) THEN
   ERLARG = ERRSUM
   .....
   CORREC = ERLARG
   .....
   NRMAX = 2
   .....
ELSE
   SMALL = SMALL * 5.0E-01
   .....
   IF (NUMRL2 .LE. 2) THEN
      NRMAX = 2
      .....
   ELSE
      NUMRL2 = NUMRL2 + 1
      .....
      IF (AREA .NE. 0.0E+00) THEN
         ERTEST = ERRBND
         .....
         CORREC = ERLARG
         .....
         NRMAX = 2
         .....
      ENDIF
   ENDIF
ENDIF
KTMIN = 0
.....
```

Observamos que a modelagem por grafo não nos fornece nenhuma resposta concreta, pois existem casos em que o grafo gerado para um trecho de programa é não planar, como no exemplo anterior, e a construção IF (...) GOTO é possível ser transformada para a estrutura IF-THEN ou IF-THEN-ELSE com a eliminação de GOTO.

As investigações nos permitiram ainda observar que se levadas em consideração as restrições impostas quando da transformação dos programas escritos em FORTRAN-66 para programas equivalentes em FORTRAN-77, toda construção IF (...) GOTO pode

ser transformada para uma estrutura IF-THEN ou IF-THEN-ELSE utilizando algumas técnicas de transformação apresentadas no capítulo 2, principalmente a técnica de repetição de código.

O emprego da técnica de repetição de código e de outras técnicas de transformação para a transformação da construção IF (...) GOTO para uma estrutura IF-THEN ou IF-THEN-ELSE será apresentado em detalhes na subseção 4.4.2.

4.4 - TRANSFORMAÇÕES DE CONSTRUÇOES FORTRAN-66 PARA FORTRAN-77

4.4.1 - Transformações do Ciclo de DO

Uma técnica de transformação que podemos aplicar sobre os programas escritos em FORTRAN-66 diz respeito ao comando DO. Como foi visto na seção anterior o comando DO trouxe como novidades na nova versão, variáveis de controle do tipo real ou dupla precisão e expressões arbitrárias para os valores inicial, final e variação dessas variáveis. Assim, trechos de programa da forma:

```
      ......
      X = 0.0
      DO 10 I = 1, 50
        X = X + 0.5
        Y = (Y + X) / 2.5
        ......
   10 CONTINUE
      ......
```

podem ser reescritos na forma:

```
      ......
      DO 10 X = 0.5, 100.0, 0.5
        Y = (Y + X) / 2.5
        ......
   10 CONTINUE
      ......
```

Trechos de programas que contêm decrementos como:

```
      ......
      J = N + 1
      DO 10 I = 1,N
        J = J - 1
        B = B * A(J)
        ......
   10 CONTINUE
      ......
```

podem ser transformados para:

```
      ......
      DO 10 J = N, 1, -1
        B = B * A(J)
        ......
   10 CONTINUE
      ......
```

4.4.2 - Transformações Para Eliminação do GOTO

Algumas técnicas de transformação que vamos aplicar nos programas escritos em FORTRAN-66 consiste em substituir trechos de programas que utilizam GOTO por estruturas do tipo IF-THEN ou IF-THEN-ELSE.

Apresentamos em seguida técnicas de transformação utilizadas para eliminar o uso do comando GOTO. Algumas destas técnicas já foram discutidas no capítulo 2 e outras técnicas serão agora apresentadas por se tratarem de técnicas que exploram características próprias da linguagem FORTRAN. As técnicas serão sempre ilustradas com exemplos retirados de Piessens et al. [21].

Para todos os exemplos que vamos apresentar a seguir, são gerados grafos planares.

Começamos apresentando uma das transformações mais simples aplicada a um comando IF. A transformação torna mais uniforme a utilização do IF.

(1)

FORTRAN-66

```
      ......
      IF (LIMIT .EQ. 1) IER = 1
      ......
```

FORTRAN-77

```
      ......
      IF (LIMIT .EQ. 1) THEN
         IER = 1
      ENDIF
      ......
```

Mostramos no exemplo (2) o uso da técnica de simplificação de condicionais triviais apresentada na subseção 2.2.8, uma técnica de transformação simples cuja aplicação resulta em uma estrutura mais legível e mais eficiente pela redução de um teste de IF.

(2)

FORTRAN-66

```
      ......
      IF (EXTRAP) IROFF2 = IROFF2 + 1
      IF (.NOT. EXTRAP) IROFF1 = IROFF1 +
     X     1
      ......
```

FORTRAN-77

```
      ......
      IF (EXTRAP) THEN
         IROFF2 = IROFF2 + 1
      ELSE
         IROFF1 = IROFF1 + 1
      ENDIF
      ......
```

Outra técnica de transformação usada para abolir o uso do comando GOTO é a técnica de repetição de código, apresentada na subseção 2.5.2, como ilustra o exemplo (3).

(3)

FORTRAN-66

```
      ......
      IF (IER .EQ. 6) GOTO 999
      IERRO = 0
      ......

      ......
  999 CONTINUE
      RETURN
      END
```

FORTRAN-77

```
      ......
      IF (IER .EQ. 6) THEN
         RETURN
      ENDIF
      IERRO = 0
      ......

      ......
  999 CONTINUE
      RETURN
      END
```

De acordo com o exemplo (3), podemos pensar em transformar o **IF (...) GOTO 999** para a estrutura **IF-THEN**. Não é interessante converter o **IF (...) GOTO 999** quando o rótulo 999 estiver muito distante do IF porque pode dificultar a visualização do escopo do comando IF. É muito mais interessante repetir as duas últimas linhas de código para fazer parte do escopo de um novo IF e obter uma estrutura do tipo apresentado na coluna FORTRAN-77.

Vale ainda ressaltar que só podemos anular a linha cujo rótulo é 999 se este rótulo não for referenciado em outro local do programa. No exemplo (3), partimos do princípio de que existem outras referências ao rótulo 999 no programa.

(4)

FORTRAN-66

```
      ......
      IF (IER .NE .0) GOTO 140
      ERRMAX = ABSERR
      ......

      ......
  140 CONTINUE
      NEVAL = 42 * LAST - 21
  999 CONTINUE
      RETURN
      END
```

FORTRAN-77

```
      ......
    IF (IER .NE. 0) THEN
      NEVAL = 42 * LAST - 21
      RETURN
    ENDIF
    ERRMAX = ABSERR
    ......

    ......
140 CONTINUE
    NEVAL = 42 * LAST - 21
999 CONTINUE
    RETURN
    END
```

Os comentários para o exemplo (4) são os mesmos do exemplo anterior. A diferença entre este exemplo e o exemplo (3) é a repetição do comando de atribuição que, neste caso, também é copiado para fazer parte do escopo do IF transformado para a estrutura IF-THEN.

(5) FORTRAN-66

```
      ......
      IF(DOMEGA .GT. 2.0E+00) GOTO 15
      NUMRL2 = 1
      EXTALL = .TRUE.
      RLIST2(1) = RESULT
15    CONTINUE
      KSGN = -1
      ......
```

FORTRAN-77

```
      ......
      IF(DOMEGA .GT. 2.0E+00) THEN
      ELSE
         NUMRL2 = 1
         EXTALL = .TRUE.
         RLIST2(1) = RESULT
      ENDIF
      KSGN = -1
      ......
```

Uma solução para excluir o **GOTO 15** na estrutura condicional e a linha de referência **15 CONTINUE** é conservar a condição da estrutura condicional e colocar em seguida a palavra THEN, formar um comando vazio colocando a palavra ELSE na próxima linha, inserir os comandos que irão fazer parte do corpo do ELSE, e depois colocar a palavra ENDIF conforme apresentado na coluna FORTRAN-77 do exemplo (5). Uma solução mais elegante e de acordo com os padrões do FORTRAN-77 para o exemplo (5) é empregar a técnica de negação de relações lógicas (subseção 2.2.17) e simplificação de condicionais triviais (subseção 2.2.8), conforme podemos observar no exemplo (5.1).

(5.1) FORTRAN-66

```
      ......
      IF(DOMEGA .GT. 2.0E+00) GOTO 15
      NUMRL2 = 1
      EXTALL = .TRUE.
      RLIST2(1) = RESULT
15    CONTINUE
      KSGN = -1
      ......
```

FORTRAN-77

```
      ......
      IF(DOMEGA .LE. 2.0E+00) THEN
         NUMRL2 = 1
         EXTALL = .TRUE.
         RLIST2(1) = RESULT
      ENDIF
      KSGN = -1
      ......
```

Outros casos de transformação de GOTO são:

(6) FORTRAN-66

```
      .......
      LST = 0
      IF (OMEGA .NE. 0.0E+00) GOTO 10
      RSLST(1) = RESULT
      LST = 1
      GOTO 999
   10 CONTINUE
      L = ABS(OMEGA)
      DL = 2 * L + 1
      .......

  999 CONTINUE
      RETURN
      END
```

FORTRAN-77

```
      .......
      LST = 0
      IF (OMEGA .EQ. 0.0E+00) THEN
         RSLST(1) = RESULT
         LST = 1
         RETURN
      ENDIF
      L = ABS(OMEGA)
      DL = 2 * L + 1
      .......

  999 CONTINUE
      RETURN
      END
```

Alguns comentários que já foram feitos nos exemplos anteriores são aplicáveis também ao exemplo (6). A diferença é que neste exemplo temos ao invés do GOTO associado a um IF, um GOTO "puro" fazendo referência a uma determinada linha de código. A repetição de linhas de código elimina o **GOTO 999**.

O processo de transformação do **IF (...) GOTO 10** para a estrutura **IF-THEN** no exemplo (6) é o mesmo citado para os exemplos anteriores.

(7) FORTRAN-66

```
      .......
      DO 20 J = 1, NINT
         IF(PTS(I).LE.PTS(J)) GOTO 20
         TEMP = PTS(I)
         PTS(I) = PTS(J)
         PTS(J) = TEMP
   20 CONTINUE
      .......
```

FORTRAN-77

```
      .......
      DO 20 J = 1, NINT
         IF(PTS(I).GT.PTS(J)) THEN
            TEMP = PTS(I)
            PTS(I) = PTS(J)
            PTS(J) = TEMP
         ENDIF
   20 CONTINUE
      .......
```

Temos no exemplo (7), dentro do domínio do comando DO, o comando IF (...) GOTO 20 e vários comandos de atribuições. Ao utilizarmos a técnica de simplificação de condicionais triviais,

transformamos como explicado em exemplos anteriores, a estrutura **IF (...) GOTO 20** para a estrutura **IF-THEN**, e observamos que os comandos de atribuições passam a fazer parte do escopo do IF e não mais diretamente do escopo do DO, conforme apresentado na coluna FORTRAN-77 do exemplo (7).

(8)

FORTRAN-66

```
      ......
      IF (EXTRAP) GOTO 100
      ALIST(LAST) = A2
      BLIST(MAXERR) = B1
      BLIST(LAST) = B2
      GOTO 110
  100 CONTINUE
      ALIST(MAXERR) = A2
      ALIST(LAST) = A1
      BLIST(LAST) = B1
  110 CONTINUE
      KSGN = -1
      ......
```

FORTRAN-77

```
      ......
      IF (EXTRAP) THEN
         ALIST(MAXERR) = A2
         ALIST(LAST) = A1
         BLIST(LAST) = B1
      ELSE
         ALIST(LAST) = A2
         BLIST(MAXERR) = B1
         BLIST(LAST) = B2
      ENDIF
      KSGN = -1
      ......
```

O exemplo (8) é transformável para a estrutura IF-THEN-ELSE utilizando a técnica de simplificação de condicionais triviais. O processo de transformação é: anulamos a linha **GOTO 110** colocando a palavra **ELSE** e excluímos os rótulos **100** e **110**, invertemos a posição dos blocos de comandos correspondentes aos escopos do **THEN** e **ELSE** do **IF**, e assim conseguimos obter o trecho apresentado na coluna FORTRAN-77 da tabela acima.

Uma solução mais elegante e simples, não havendo necessidade de troca de posição de blocos de comandos mas apenas mudança da condição do comando IF, é empregar a técnica de negação de relações lógicas e simplificação de condicionais triviais, cuja solução final é apresentada a seguir no exemplo (8.1).

(8.1) FORTRAN-66

```
      ......
      IF (EXTRAP) GOTO 100
      ALIST(LAST) = A2
      BLIST(MAXERR) = B1
      BLIST(LAST) = B2
      GOTO 110
  100 CONTINUE
      ALIST(MAXERR) = A2
      ALIST(LAST) = A1
      BLIST(LAST) = B1
  110 CONTINUE
      KSGN = -1
      ......
```

FORTRAN-77

```
      ......
      IF (.NOT.EXTRAP) THEN
         ALIST(LAST) = A2
         BLIST(MAXERR) = B1
         BLIST(LAST) = B2
      ELSE
         ALIST(MAXERR) = A2
         ALIST(LAST) = A1
         BLIST(LAST) = B1
      ENDIF
      KSGN = -1
      ......
```

Mais casos de transformação de GOTO são apresentados a seguir.

(9) FORTRAN-66

```
      ......
      IF(IERRO .NE. 0) GOTO 110
      ......
      IF(IER .NE. 0) GOTO 105
      ......
      GOTO 110
  105 CONTINUE
      KTMIN = 5
  110 CONTINUE
      KSGN = IER + IERRO
      ......
```

FORTRAN-77

```
      ......
      IF(IERRO .EQ. 0) THEN
        ......
        IF(IER .EQ. 0) THEN
          ......
        ELSE
          KTMIN = 5
        ENDIF
      ENDIF
      KSGN = IER + IERRO
      ......
```

Todos os comandos **IF (...) GOTO** no exemplo (9) são transformáveis para a estrutura **IF-THEN** com eliminação do GOTO. Colocamos o exemplo apenas para ilustrar as diversas situações em que podemos converter estruturas **IF (...) GOTO** para **IF-THEN** ou **IF-THEN-ELSE**.

(10) FORTRAN-66

```
      ......
      IF(LST .GT. 1) GOTO 20
      PSUM(1) = RSLST(1)
      GOTO 30
   20 CONTINUE
      PSUM(NUMRL2) = PSUM(LL) + A
      IF (LST .EQ. 2) GOTO 30
      KTMIN = KTMIN + 1
      ABSERR = ABSEPS
   30 CONTINUE
      DRL = 5.0E+00
      ......
```

FORTRAN-77

```
      ......
      IF(LST .LE. 1) THEN
         PSUM(1) = RSLST(1)
      ELSE
         PSUM(NUMRL2) = PSUM(LL)
     X     + A
         IF (LST .NE. 2) THEN
            KTMIN = KTMIN + 1
            ABSERR = ABSEPS
         ENDIF
      ENDIF
      DRL = 5.0E+00
      ......
```

Os comentários do exemplo (9) são aplicáveis ao exemplo (10).

(11) FORTRAN-66

```
      ......
      IF (ABSERR .GE. ERRBND) GOTO 150
      KTMIN = 0
      ABSERR = ABSEPS
      RESULT = RESEPS
      IF (A1 .LT. 2.0E+00) GOTO 170
  150 CONTINUE
      NRMAX = 1
      ......

  170 CONTINUE
      KTMIN = KTMIN + 1
      ......
```

FORTRAN-77

```
      ......
      IF (EPS .GE. ERR) THEN
         NRMAX = 1
         ......

      ELSE
         KTMIN = 0
         ABSERR = ABSEPS
         RESULT = RESEPS
         IF (A1 .GE. 2.0E+00) THEN
            NRMAX = 1
            ......

         ENDIF
      ENDIF
      KTMIN = KTMIN + 1
      ......
```

O **IF (...) GOTO 150** no exemplo (11) é transformável para a estrutura padrão **IF-THEN**. O processo de transformação, como podemos observar na tabela acima, é o mesmo comentado para casos anteriores.

Já o IF (...) GOTO 170 parece não poder ser eliminado porque a princípio fica difícil aplicar alguma técnica que possibilite uma transformação para uma estrutura equivalente no padrão FORTRAN-77. Mas como podemos observar na coluna FORTRAN-77 da tabela (11), ao utilizar também a técnica de repetição de código (subseção 2.5.2), conseguimos facilmente adaptar o trecho de programa FORTRAN-66 para o padrão FORTRAN-77.

Apresentamos no apêndice I outros exemplos de construções FORTRAN-66 que empregam a estrutura IF (...) GOTO com suas respectivas transformações para estruturas IF-THEN ou IF-THEN-ELSE em FORTRAN-77 após a utilização de técnicas de transformação abordadas no capítulo 2.

Todos os exemplos apresentados anteriormente envolvem o comando IF (...) GOTO isolado. Os casos mais complicados de transformação são justamente os casos onde temos o comando IF (...) GOTO X com o rótulo X fazendo parte do escopo de um comando DO com rótulo Y, com X > Y (recorde-se que os rótulos aparecem sempre em ordem crescente).

Mostramos a seguir exemplos de construções FORTRAN-66 que utilizam a estrutura IF (...) GOTO dentro do escopo de um comando DO com suas respectivas transformações para estruturas semelhantes no FORTRAN-77.

(12) FORTRAN-66

```
      ......
      DO 130 K = ID, JUPBND
         MAXERR = IORD(K)
         ERRMAX = ERRMAX + ELIST(K)
         IF (KTMIN .LE. 5) GOTO 160
         ......
         NRMAX = NRMAX + 1
130   CONTINUE
      ......
160   CONTINUE
      KTMIN = KTMIN + NEV
      ......
```

FORTRAN-77

```
      ......
      IF (KTMIN .GT. 5) THEN
         DO 130 K = ID, JUPBND
            MAXERR = IORD(K)
            ERRMAX = ERRMAX +
     X           ELIST(K)
            NRMAX = NRMAX + 1
130      CONTINUE
         ......
      ENDIF
      KTMIN = KTMIN + NEV
      ......
```

Para eliminar o uso do comando **IF (...) GOTO 160** no exemplo (12) basta empregar a técnica de remoção de comandos invariantes (subseção 2.3.3) e em seguida utilizar a técnica de negação de relações lógicas (subseção 2.2.17). Após utilizar as referidas técnicas de transformação, obtemos uma nova sequência de código como podemos observar na coluna FORTRAN-77 da tabela acima.

(13) FORTRAN-66

```
      ......
      DO 160 K = 1, LIMIT
         DRES = DRES + ELIST(K)
         IF (ALIST(K) .LT. A) GOTO 170
         A1 = A1 + ALIST(K)
         B1 = B1 + BLIST(K)
160   CONTINUE
170   CONTINUE
      MAXERR = MAXERR + 1
      ......

      RETURN
      END
```

FORTRAN-77

```
      ......
      DO 160 K = 1, LIMIT
         DRES = DRES + ELIST(K)
         IF (ALIST(K) .LT. A)
     X      THEN
            K = LIMIT + 5
         ELSE
            A1 = A1 + ALIST(K)
            B1 = B1 + BLIST(K)
         ENDIF
160   CONTINUE
      MAXERR = MAXERR + 1
      ......

      RETURN
      END
```

Para eliminar o GOTO 170 no exemplo (13), uma solução é atribuir à variável de controle do DO um valor maior que o

limite superior que conforme o padrão FORTRAN-77, torna o comando DO automaticamente inativo passando a execução para a linha seguinte ao CONTINUE, que é a linha referenciada pelo GOTO 170. A transformação poderia ser algo da forma apresentado na coluna FORTRAN-77 da tabela (13).

A solução apresentada para o exemplo (13) não é interessante por dois motivos:

- ao criar uma nova linha de código fonte para redefinir a variável de controle do comando DO, a linha K = LIMIT + 5, esta solução infringe a restrição imposta de não poder criar novas linhas de código fonte quando da transformação dos programas FORTRAN-66 para FORTRAN-77.
- por outro lado, o fato de redefinir a variável de controle do DO para alcançarmos nossos objetivos nos parece um truque sutil de programação e assim, contraria a meta principal da transformação que é de melhorar a legibilidade. Para amenizar o problema seria interessante por exemplo, colocar comentários no novo trecho de programa que alertassem para a ação de interrupção do comando DO.

Uma solução mais apropriada para converter a construção IF (...) GOTO da tabela (13) para uma estrutura IF-THEN em FORTRAN-77, é empregar a técnica de repetição de código como podemos observar na tabela (13.1).

(13.1) FORTRAN-66

```
    .......
    DO 160 K = 1, LIMIT
       DRES = DRES + ELIST(K)
       IF (ALIST(K) .LT. A) GOTO 170
       A1 = A1 + ALIST(K)
       B1 = B1 + BLIST(K)
160 CONTINUE
170 CONTINUE
    MAXERR = MAXERR + 1
    .......
    RETURN
    END
```

FORTRAN-77

```
    .......
    DO 160 K = 1, LIMIT
       DRES = DRES + ELIST(K)
       IF (ALIST(K) .LT. A)
          MAXERR = MAXERR + 1
          .......
          RETURN
       ENDIF
       A1 = A1 + ALIST(K)
       B1 = B1 + BLIST(K)
160 CONTINUE
    MAXERR = MAXERR + 1
    .......
    RETURN
    END
```

O problema de utilizar a técnica de repetição de código, principalmente quando os programas objetos de transformação são grandes e utilizam vários comandos da forma IF (...) GOTO, é o aumento significatico do número de linhas de código fonte. Como consequência do aumento de código fonte, o tempo de processamento pode aumentar e memória extra será necessária. Neste caso, pode-se criar subrotinas para substituir os blocos de códigos comuns e reduzir os gastos com memória.

Outro problema decorrente da utilização da técnica de repetição de código é a possibilidade de tornar os programas transformados menos legíveis. Já a criação de subrotinas para armazenar as linhas de código repetidas vezes infringe as restrições impostas na seção anterior. Assim, a implementação desta técnica na transformação automática de programas se torna complexa.

Trechos de programas em FORTRAN-66 como

(14)

```
      ......

      IF (.NOT. EXTALL) GOTO 50
      IF (ABS (B1 - A1) .GT. SMALL) ERLARG = ERLARG + ERRO12
      IF (EXTRAP) GOTO 70
   50 CONTINUE
      WIDTH = ABS (BLIST(MAXERR) - ALIST(MAXERR))
      IF (WIDTH .GT. SMALL) GOTO 140
      IF (EXTALL) GOTO 60
      SMALL = SMALL * 5.0E+00
      IF (2.5E-01 * WIDTH * DOMEGA .GT. 2.0E+00) GOTO 140
      EXTALL = .TRUE.
      GOTO 130
   60 CONTINUE
      EXTRAP = .TRUE.
      NRMAX = 2
   70 CONTINUE
      IF (IERRO .EQ. 3 .OR. ERLARG .LE. ERTEST) GOTO 90
      ......

   90 CONTINUE
      ......

  130 CONTINUE
      ......

  140 CONTINUE
      ......
```

são extremamentes complicados e requerem um maior esforço computacional para torná-los mais claros e adaptá-los a estruturas IF-THEN ou IF-THEN-ELSE do FORTRAN-77.

Algumas das dificuldades de transformação da construção IF (...) GOTO para uma construção IF-THEN ou IF-THEN-ELSE são consequência das próprias condições impostas pelo FORTRAN-77 que só tem a estrutura IF-THEN-ELSE e o comando DO como estruturas de controle.

O fato do FORTRAN-77 também apresentar pobreza de estruturas de controle dificulta a eliminação do comando GOTO, e essa situação vem comprovar a teoria de Bohm & Jacopini [02] de que sem o comando WHILE a estrutura dos programas pode se tornar

complexa. Bohm & Jacopini [02] mostram que IF-THEN-ELSE, WHILE e uma sequência de código fonte são suficientes para expressar qualquer programa sem o GOTO.

A dificuldade de transformar alguns comandos GOTO presentes nos programas também reforça a idéia de que programas bem escritos quando submetidos aos sistemas transformacionais, resultarão também em bons programas transformados. Por outro lado, quando programas com estruturas obscuras e complicadas são transformados, podemos esperar que os programas resultantes também apresentem estruturas complicadas.

5 - PROTÓTIPO DE CONVERSOR

O estudo de técnicas de transformação de programas fontes reforçou a idéia de criarmos um protótipo de conversor para aplicá-lo sobre programas implementados em FORTRAN-66 para convertê-los em programas em FORTRAN-77 equivalentes.

Apresentamos neste capítulo o projeto de um protótipo de sistema conversor automático de programas que chamaremos de Sistema Conversor de Programas FORTRAN-66 para FORTRAN-77 (SISCO). É apenas um sistema experimental.

Este capítulo está assim dividido: na seção 5.1 descrevemos o ambiente de desenvolvimento do protótipo; na seção 5.2 apresentamos os objetivos estabelecidos para o sistema; na seção 5.3 detalhamos a abordagem de desenvolvimento; na seção 5.4 apresentamos as disciplinas de programação empregadas para o FORTRAN-66 e o FORTRAN-77; na seção 5.5 citamos as restrições impostas pelo sistema; e na seção 5.6 exemplificamos as técnicas de transformações manipuláveis pelo sistema.

5.1 - COMENTÁRIOS GERAIS

O Sistema está sendo desenvolvido para transformar inicialmente a Biblioteca Transportável de Análise Numérica (BITAN), uma biblioteca de rotinas numéricas implementadas em FORTRAN-66, existente em uma máquina IBM 4381 no Núcleo de Processamento de Dados da Universidade Federal da Paraíba - Campus II - Campina Grande - PB, para a obtenção de uma nova versão da biblioteca em FORTRAN-77 (adaptação de Medeiros [19]).

Na BITAN, cerca de 260 algoritmos matemáticos estão implementados utilizando a linguagem FORTRAN-66.

De acordo com Hattori & Pequeno [13], recentemente 80% da BITAN foi transportada para microcomputadores compatíveis com IBM-PC - 54.000 das quase 70.000 linhas de programa fonte.

O papel do conversor neste estágio inicial é traduzir as possíveis construções transformáveis no FORTRAN-66 para FORTRAN-77, mantendo a integridade lógica do programa e a tentativa de deixá-lo estruturado.

A versão FORTRAN-77 trouxe a flexibilidade de poder trabalhar com a estrutura IF-THEN-ELSE, um dos recursos que facilitam a programação estruturada em FORTRAN, e isso despertou o interesse em converter os programas escritos em FORTRAN-66 para FORTRAN-77 tendo em vista a versão mais antiga não oferecer recursos de programação estruturada.

A conversão automática de programas FORTRAN-66 para FORTRAN-77 é, portanto, o principal objetivo a ser alcançado pelo conversor. Mas a necessidade de transformar as várias rotinas da BITAN é justificada por vários outros objetivos a serem alcançados, e que foram previamente estabelecidos e esperados para o Sistema a ser desenvolvido, dentre os quais podemos citar:

1 - produzir programas mais claros, mais legíveis, conforme os padrões atuais de programação;

2 - converter programas a um custo relativamente baixo;

3 - praticidade: construir um conversor de fácil manuseio.

Passamos agora a enfocar os vários aspectos considerados durante o desenvolvimento do protótipo de conversor SISCO.

O SISCO foi implementado em linguagem de programação Pascal. Hoje, o Sistema contém aproximadamente 800 linhas de código fonte e funciona em qualquer microcomputador compatível com IBM-PC com TURBO-PASCAL.

O SISCO foi projetado para converter programas escritos em FORTRAN-66 para programas equivalentes em FORTRAN-77. Como as versões FORTRAN-66 e FORTRAN-77 são muito semelhantes, várias transformações são executadas diretamente.

Mesmo com a ausência de estruturas como WHILE e REPEAT na nova versão, o Conversor nesta fase inicial tenta reescrever programas FORTRAN eliminando sempre que possível o comando GOTO e introduzindo a estrutura condicional IF-THEN ou IF-THEN-ELSE. O comando GOTO é conservado quando nenhuma outra estrutura disponível na versão FORTRAN-77 possibilite sua eliminação.

A continuação do emprego do comando GOTO em diversos casos se dá em decorrência da não utilização da técnica de repetição de código nesta fase inicial, pois como já foi comentado no capítulo anterior, a técnica de repetição de código se torna muito complexa em vários casos durante a etapa de transformação de construções FORTRAN-66 para FORTRAN-77.

O Conversor atualmente não melhora programas no sentido de, por exemplo, criar código, criar subrotinas, alterar nomes de variáveis, adicionar novas variáveis e comentários, efetuar transformações padrões tais como eliminar variáveis redundantes, atribuições inúteis.

5.2 - DISCIPLINAS DE PROGRAMAÇÃO EMPREGADAS

Bibliotecas de subprogramas existentes, empregaram e passaram a adotar um conjunto de regras de codificação durante o processo de desenvolvimento dos subprogramas para facilitar a sua leitura e compreensão e assim, reduzir as dificuldades na fase de manutenção.

Smith [24] apresenta uma disciplina de programação para se trabalhar com a linguagem FORTRAN-66. Ele reforça a importância do emprego de uma disciplina na conversão de programas FORTRAN-66, principalmente porque a linguagem FORTRAN-66 não contém construções e estruturas de controle bem adaptados a algoritmos numéricos.

A vantagem de utilizar uma disciplina de programação consiste em obter melhor legibilidade dos programas e facilidade de manutenção, além de facilitar uma conversão automática de programas.

5.2.1 - Disciplina de Programação Para a Versão FORTRAN-66

A BITAN foi desenvolvida em FORTRAN já obedecendo a algumas padronizações quanto ao estilo de programação. Portanto, o Sistema deve, tanto quanto possível, ser aplicável à biblioteca existente, levando em consideração estas padronizações (Medeiros [19]).

O Conversor, portanto, exige que os programas estejam dentro dos padrões adotados pela disciplina de programação.

Hattori [11], inspirado no trabalho de Smith [24],

apresenta de forma detalhada os padrões sugeridos para a BITAN.

Apresentamos a seguir, extraído de Hattori [11], um resumo dos tópicos da disciplina de programação empregada na BITAN. Selecionamos os tópicos da disciplina de programação que apresentam uma maior afinidade com o nosso trabalho.

A disciplina de programação impõe restrições, estabelece regras e padrões:

1. Restrição no conjunto de caracteres
2. Construções do FORTRAN restritas a aquelas definidas pelo PFORT (Ryder [23])
3. Regras para obtenção de uma boa estrutura
   a) Declaração explícita de tipos de todas as variáveis
   b) Regras para a utilização do comando DO
   c) Proibição do IF aritmético
   d) Números de comando em ordem crescente
   e) Ordenação dos comandos de especificação
   f) Convenções de formatação
   g) Convenções de nomenclatura
4. Padrões de documentação interna.

5.2.2 - Disciplina de Programação Para a Versão FORTRAN-77

O surgimento da nova versão, o FORTRAN-77, exigiu uma revisão e atualização na disciplina de programação empregada para as rotinas da BITAN.

Hattori [12] detalha os padrões a serem empregados na disciplina de programação em FORTRAN-77. O trabalho de Hattori [12] é uma atualização de seu próprio trabalho, Hattori [11].

Apresentamos a seguir, apenas as diferenças mais importantes da nova disciplina comparada com a anterior. A maior diferença ocorre no subconjunto do FORTRAN que agora inclui os novos comandos PARAMETER, PROGRAM, IF-THEN-ELSE e os comandos de entrada e saída, OPEN, CLOSE, ENDFILE, INQUIRE e PRINT em suas formas padronizadas, e a melhoria do comando DO. O que maior reflexo tem na disciplina é a utilização do IF-THEN-ELSE.

Com a adição da estrutura IF-THEN-ELSE na versão FORTRAN-77, o tópico convenções de formatação sofre alteração, passando agora a ser definida a seguinte regra:

- Os comandos correspondentes a THEN num IF devem ser deslocados três posições para a direita em relação à coluna do IF, o ELSE deve ficar na mesma coluna do IF e os comandos correspondentes ao ELSE devem ser deslocados três posições para a direita em relação à coluna do ELSE e finalmente, o ENDIF deve começar na mesma margem do IF como se mostra a seguir,

```
      ......
      IF ( . . . ) THEN
         C1
         C2
         ......
      ELSE
         S1
         S2
         ......
      ENDIF
      ......
```

Os comandos Ci's e Si's podem ser IF's, casos em que a regra deve ser aplicada recursivamente.

5.3 - RESTRIÇÕES

Além de algumas restrições já apresentadas quando da discussão das disciplinas de programação, relacionamos outras restrições que devem estar presentes nos programas fontes das rotinas da BITAN quando submetidos ao processo de conversão. Algumas restrições impostas são:

(1) - As colunas de 73 a 80 não serão utilizadas em nenhuma situação;

(2) - Quando tivermos comandos dos tipos **IF (EXTRAP)** e **IF (.NOT. EXTRAP)**, sempre a estrutura condicional que tem o **.NOT.** deve vir em seguida à estrutura normal, ou seja, devem vir na sequência:

```
IF (EXTRAP) comando
IF (.NOT. EXTRAP) comando
```

(3) - Em comandos do tipo **IF (condição) ***, o " * " deve ser um comando do tipo: GOTO, CALL ou atribuição.

(4) - Em regras para obter uma boa estrutura, no item convenções de formatação, há uma alteração quanto ao ponto de quebra de uma linha de comando. Além dos pontos de quebra já conhecidos da disciplina, o ponto de quebra de uma linha de comando pode se dar também depois de um operador relacional (.EQ., .NE., .LT., .LE., .GT. e .GE.).

(5) - A disciplina empregada obriga que se utilizem rótulos em ordem crescente, sempre fazendo referências a rótulos na ordem ascendente. Portanto, fica proibido fazer referências a rótulos que estão definidos antes das linhas de comandos que os referenciam, e sendo assim, não podemos ter construções do tipo:

```
      ......

   15 NUMRL2 = NUMRL2 + 1
      ALIST(ID) = JUPBND(NUMRL2)
      NRMAX(NUMRL2) = LAST(NUMRL2) + KTMIN
      IF (NUMRL2 .LT. ID) GOTO 15
      ......
```

5.4 - CONSTRUÇÕES TRANSFORMÁVEIS

Apresentamos nesta seção, exemplos de construções FORTRAN-66 que serão manipuladas e que ilustram a ação do Conversor.

O conjunto de construções que vamos apresentar representa um resumo das construções mais comuns a muitas rotinas da BITAN.

O nosso estudo basicamente se concentrou nos seguintes comandos da linguagem FORTRAN: atribuição; declaração de variáveis; chamadas às subrotinas; RETURN; END; IF's do tipo IF (condição) GOTO, IF (condição) atribuição e IF (condição) CALL, e comentário.

Construções do tipo:

```
(1)       SUBROUTINE DQAGS (F, A, B, EPSABS, EPSREL, RESULT,
         X                  ABSERR, NEVAL, IER)
(2)   C   PROGRAMA PARA CÁLCULO DE INTEGRAIS
(3)       INTEGER I, ID, IER, IERRO
(4)       LOGICAL EXTRAP, NOEXT
(5)       REAL A, ABSEPS, ABSERR, AREA
(6)       DOUBLE PRECISION ALIST(500), BLIST(500), ELIST(500)
(7)       EXTERNAL F
(8)       DATA LIMIT /500/
(9)       CALL QMACO (EPMACH, UFLOW, OFLOW)
(10)      DO 100 LAST = 1,LIMIT
(11) 100  CONTINUE
(12)      IER = 0
(13)      RETURN
(14)      END
```

serão automaticamente copiadas para a nova versão sem nenhuma alteração, pois as construções já se enquadram no padrão da nova versão.

Quanto ao comando **IF (...) GOTO** do FORTRAN-66, o Conversor nesta fase inicial consegue convertê-lo, na maioria das vezes, para estruturas IF-THEN ou IF-THEN-ELSE do FORTRAN-77. O Conversor nesta fase ainda não consegue empregar algumas técnicas de transformação devido à complexidade de suas implementações como a técnica de repetição de código que facilita a eliminação do comando GOTO, e por esse motivo, o comando GOTO ainda é utilizado.

Os trechos de programas FORTRAN-66 apresentados no capítulo 4 são totalmente manipuláveis pelo Conversor, sendo que ao invés de utilizar algumas técnicas de transformação como a técnica de repetição de código, o Conversor conserva a presença do comando GOTO. Um exemplo dessa situação é apresentado a seguir.

FORTRAN-66

```
      ......
      IF (LAST .EQ. 2) GOTO 120
      ERLARG = ERLARG - ERLAST
      IF (EXTALL) GOTO 60
      SMALL = SMALL * 5.0E-01
      GOTO 130
   60 CONTINUE
      EXTRAP = .TRUE.
      NRMAX = 2
  120 CONTINUE
      SMALL = SMALL * 3.0E-01
      RLIST(NUMRL2) = AREA
  130 CONTINUE
      ERTEST = ERRBND
      ......
```

FORTRAN-77

```
      ......
      IF (LAST .NE. 2) THEN
         ERLARG = ERLARG - ERLAST
         IF (EXTALL) THEN
            SMALL = SMALL * 5.0E-01
            GOTO 130
         ENDIF
         EXTRAP = .TRUE.
         NRMAX = 2
      ENDIF
      SMALL = SMALL * 3.0E-01
      RLIST(NUMRL2) = AREA
  130 CONTINUE
      ERTEST = ERRBND
      ......
```

Para um melhor entendimento da ação do Conversor, listagens completas de dois pacotes de software que utilizam rotinas existentes na BITAN, os quais inicialmente empregam algumas dessas rotinas em FORTRAN-66, compilados e executados, e listagens dos mesmos pacotes, utilizando as rotinas que foram convertidas para FORTRAN-77 pelo SISCO, também compilados e executados, são apresentadas nos apêndices III e V.

Apresentamos nos apêndices II e IV resultados obtidos quanto ao tempo de transformação das rotinas escritas em FORTRAN-66 para FORTRAN-77, quanto ao tempo de compilação e tempo de execução dos pacotes utilizando rotinas em FORTRAN-66 e dos mesmos pacotes utilizando as rotinas derivadas da transformação.

6 - CONCLUSÕES

Os aspectos relacionados com o ciclo de desenvolvimento de software tem variado ao longo dos anos.

Inicialmente os programadores enfatizavam construções que permitiam obter eficiência em detrimento da clareza. Em seguida, surgiu o desejo de desenvolver e manter programas bem estruturados sacrificando às vezes a eficiência dos programas. A solução foi tentar atingir um compromisso entre programas eficientes e legíveis.

Já nos dias atuais existe um consenso definido: o objetivo deve ser sempre desenvolver programas bem estruturados para facilitar o entendimento, a depuração, modificação e manutenção. Um programa legível é mais importante que um programa eficiente, pois como os programas geralmente são atualizados por pessoas que não os desenvolveram, os programas legíveis são mais fáceis de serem depurados, modificados e mantidos. A preocupação com a clareza dos programas pode também melhorar a sua eficiência porque essa clareza pode revelar pontos críticos que poderão ser analisados e melhorados.

Com o advento dos supercomputadores, a preocupação com a eficiência do software, atualmente considerada não prioritária, volta a ser muito importante no processo de desenvolvimento de software. Ao desenvolver software para supercomputadores, procura-se explorar o máximo de sua velocidade de execução, adequando um programa às características de operação do equipamento.

Quanto ao polêmico problema de eliminação do comando GOTO, comprovamos que a modelagem por grafo não nos oferece nenhuma resposta concreta. Esse resultado foi um tanto surpreendente já que esperávamos que existisse alguma ligação entre as propriedades dos grafos e a possibilidade de eliminação do GOTO. Mesmo assim, o resultado se torna interessante porque ninguém vai mais tentar solucionar o problema de eliminar o comando GOTO por esse caminho. Alguma outra maneira de modelar o fluxo de controle de programas precisa ser investigada.

As investigações nos permitiram ainda observar que se levadas em consideração as restrições impostas no capítulo 4, subseção 4.3, quando da transformação dos programas escritos em FORTRAN-66 para programas equivalentes em FORTRAN-77, toda construção IF (...) GOTO pode ser transformada para uma estrutura IF-THEN ou IF-THEN-ELSE utilizando algumas técnicas de transformação apresentadas no capítulo 2, principalmente a técnica de repetição de código.

Algumas das dificuldades de transformação da construção IF (...) GOTO para uma estrutura IF-THEN ou IF-THEN-ELSE são consequência das próprias condições impostas pelo FORTRAN-77 por só possuir a estrutura IF-THEN-ELSE e o comando DO como estruturas de controle. O comando WHILE resolveria alguns problemas dos casos de transformações. Essa observação vem reforçar a teoria de Bohm & Jacopini [02] segundo a qual as estruturas de controle IF-THEN-ELSE e WHILE mais a sequência de código fonte são suficientes para expressar qualquer programa, portanto sem o GOTO. Contudo, mesmo que WHILE estivesse presente,

a técnica de repetição de código para eliminar GOTO's poderia aumentar o código excessivamente. Entre a teoria e a prática existem dificuldades imprevisíveis.

A dificuldade de transformar alguns comandos GOTO presentes nos programas também reforça a idéia de que programas bem escritos quando submetidos aos sistemas transformacionais, resultarão também em programas transformados bem estruturados.

Quanto ao SISCO é esperado que se torne uma ferramenta útil na manutenção de programas FORTRAN existentes. Nesta fase inicial, o SISCO consegue substituir vários comandos GOTO utilizados em programas escritos em FORTRAN-66 por estruturas IF-THEN ou IF-THEN-ELSE em FORTRAN-77. Um passo seguinte seria fazer com que o SISCO utilizasse outras técnicas de transformação que também facilitam a eliminação do comando GOTO como, por exemplo, a técnica de repetição de código.

Para resolver os problemas decorrentes da utilização da técnica de repetição de código, poderia criar subrotinas especiais para substituir os blocos de códigos comuns. Assim em princípio, toda construção IF (...) GOTO seria transformada para uma estrutura IF-THEN ou IF-THEN-ELSE.

O não emprego de algumas técnicas de transformação de programas fontes que facilitam a transformação da construção IF (...) GOTO para a estrutura IF-THEN ou IF-THEN-ELSE pelo SISCO é ainda consequência de não ter sido encontrada uma ferramenta de modelagem que respondesse formalmente questões básicas como quando pode e quando não pode uma construção IF (...) GOTO ser transformada para uma estrutura IF-THEN ou IF-THEN-ELSE com

eliminação do GOTO.

Uma solução para resolver os casos mais complicados de transformação da construção IF (...) GOTO para a estrutura IF-THEN ou IF-THEN-ELSE seria transformar o SISCO num sistema interativo de conversão de programas, onde os casos mais simples de transformações seriam convertidos automaticamente pelo SISCO, enquanto que os casos mais complicados de transformações seriam efetuados com uma interação do SISCO com o próprio usuário.

Um outro passo no trabalho com SISCO seria concentrar esforços na exploração de outros recursos surgidos com a versão FORTRAN-77 como a maior flexibilidade de utilização da variável de controle do comando DO.

Após a exploração dos recursos sofisticados da versão FORTRAN-77, um outro passo seria trabalhar no emprego de técnicas de transformação de programas fontes padrões tais como eliminação de variáveis e atribuições redundantes; eliminação de variáveis e atribuições inúteis; remoção de comandos invariantes e outras.

E por fim, seriam analisadas as transformações que visam explorar o desempenho dos softwares em supercomputadores.

Uma outra abordagem seria empregar o SISCO para traduzir programas escritos em FORTRAN-66 para programas equivalentes em linguagens que oferecem mais recursos avançados de programação (melhores comandos de iteração e estruturas de controle, apontadores, recursão) que o FORTRAN-77 como Pascal e C. Seria interessante transformar programas FORTRAN-66 para programas em C principalmente pela portabilidade de programas em C.

Com a aprovação do FORTRAN-90 pelo American National Standards Institute (ANSI) e International Standards Organization

(ISO), uma abordagem interessante seria concentrar esforços na transformação dos programas FORTRAN-66 ou mesmo dos programas em FORTRAN-77 para FORTRAN-90, já que esta última versão do FORTRAN apresenta novidades como o comando de iteração WHILE, recursividade, apontadores e operações diretas com conjuntos.

7 - REFERÊNCIAS BIBLIOGRÁFICAS

[1] Baker, Brenda S. - An algorithm for Structuring Flowgraphs, Journal of the Association for Computing Machinery, 24 (1977), pp. 98-120.

[2] Bohm, Corrado & Jacopini, Giuseppe - Flow-diagrams, Turing Machines and Languages with only Two Formation Rules, Comm. ACM, 9, 5 (may 1966), pp. 336-371.

[3] Bondy, J. A. & Murty, U. S. R. - Graph Theory with Applications, American Elsevier Publishing Co., Inc, New York, N. Y., U.S.A., 1977.

[4] Boyle, James M. - Mathematical Software Transportability Systems - Have the Variations a Theme?, Portability of Numerical Software, W. Cowell (ed.), Oak Brook, Illinois, Jun/1976, Springer-Verlag, New York, 1977, pp. 305-360.

[5] Boyle, James M., K. W. Dritz, O. B. Arushanian & Y. V. Kuchevskiy - Program Generation and Transformation - Tools for Mathematical Software Development in Information Processing 77, North-Holland Publishing Company (1977).

[6] Boyle, J. & Matz, M. - Automatic Multiple Program Realizations, Symposium on Computer Software Engineering, Polytechnic Institute of New York - Apr/1976, pp. 421-456.

[7] Dongarra, J. J. & Gustavson, F. G. & Karp, A. - Implementing Linear Algebra Algorithms for Dense Matrices on a Vector Pipeline Machine, Siam Review, 1 (1984), pp. 91-112.

[8] Dritz, Kenneth W. - Multiple Program Realizations Using the TAMPR System, Portability of Numerical Software, W. Cowell (ed.), Oak Brook, Illinois, Jun/1976, Springer-Verlag, New York, 1977, pp. 405-423.

[9] Feather, Martin S. - A System for Assisting Program Transformation, ACM Trans. Program. Lang. Syst., 4 (1982), pp. 1-20.

[10] Hague, S. J. - Software Tools, Numerical Software = Needs and Avaliability, D. A. H. Jacobs (ed.), Academic Press, London, 1978, pp. 57-79.

[11] Hattori, Mário T. - Uma Disciplina de Programação em FORTRAN-66, Relatório Técnico, Departamento de Sistemas e Computação, Universidade Federal da Paraíba, Campina Grande - PB, Mai/1984.

[12] Hattori, Mário T. - Uma Disciplina de Programação em FORTRAN-77, Relatório Técnico, Departamento de Sistemas e Computação, Universidade Federal da Paraíba, Campina Grande - PB, Jun/1989.

[13] Hattori, Mário T. & Pequeno, Mauro C., O Projeto Micro-BITAN, XIII CNMAC, Águas de Lindóia - SP, Nov/1990.

[14] Hattori, Mário T. & Queiroz, Bruno C. N. - Manuscrito - Departamento de Sistemas e Computação, Universidade Federal da Paraíba (UFPB), Campina Grande -PB, Set/1990.

[15] Hehl, Maxmilian Emil - Linguagem de Programação Estruturada FORTRAN-77, McGraw-Hill, São Paulo, 1986.

[16] Knuth, Donald E. - Structured Programming with GOTO Statements, ACM Computing Surveys, 6 (1974), pp. 261-301.

[17] Loveman, David B. - Program Improvement by Source to Source Transformation,3rd ACM Symposium on Principles of Programming Languages, Atlanta, Georgia, Jan/1976, pp. 140-152; Também em Journal of the Association for Computing Machinery, 24 (1977), pp. 121-145.

[18] Maher, B. & Sleeman, D. M. - Automatic Program Improvement: Variable Usage Transformations, ACM Trans. Program. Lang. Syst., 5 (1983), pp. 236-264.

[19] Medeiros, Camilo de L. G. - Projeto e Implementação de um Sistema Transformador de Programas FORTRAN, Dissertação de Mestrado, Departamento de Sistemas e Computação, Universidade Federal da Paraíba, Campina Grande - PB, 1986.

[20] Partsch, H. & Steinbrueggen, R. - Program Transformation Systems, Computing Surveys, 15 (1983), pp. 199-236.

[21] Piessens, R., Doncker-Kapenga, E. de, Uberhuber, W. & Kahaner, D. K., QUADPACK - A Subroutine Package for Automatic Integration, Springer-Verlag, Berlin Heidelberg, 1983.

[22] Reingold, E. M., J. Nievergelt & N. Deo - Combinational Algorithms - Theory and Practice, Prentice-Hall, Inc, Englewood Cliffs, New Jersey, U.S.A., 1977.

[23] Ryder, B. G. - The PFORT Verifier, Software = Practice and Experience, 4 (1974), pp. 359-377.

[24] Smith, Brian T. - FORTRAN Poisoning and Antidotes , Portability of Numerical Software, W. Cowell (ed.), Oak Brook, Illinois, Jun/1976, Springer-Verlag, New York, 1977, pp. 178-256.

[25] Standish, Thomas A., D. F. Kibbler & J. M. Neighbors - Improving and Refining Programs by Program Manipulation. Proceedings ACM Annual Conf., Houston, Texas, 1976, pp. 509-516.

[26] Standish, Thomas A., D. Harriman, D. F. Kibbler & J. M. Neighbors - The Irvine Program Transformation Catalogue. Dep. Inform. and Comptr. Sci., University of California at Irvine, Irvine, California, Jan/1976.

[27] Tassel, Dennie Van - Program Style, Design, Efficiency, Debugging and Testing, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, U.S.A., 1974.

[28] Wolberg, John R. & Rafal, M. - Using Convert to Transform Source Code, Software=Practice and Experience, 9, (1979), pp. 881-890.

APÊNDICE I

CONSTRUÇÕES FORTRAN-66 TRANSFORMADAS PARA FORTRAN-77

(1)

FORTRAN-66

```
      ......

      IF (LAST. EQ. 2) GOTO 180
      ABSERR = ABSEPS
      RESULT = RESEPS
      IF (NOEXT) GOTO 180
      NRMAX = 1
      EPSABS = RESEPS
      GOTO 190
180   CONTINUE
      ERLARG = ERRSUM
      ERTEST = ERRBND
190   CONTINUE
      RESULT = KTMIN
      SMALL = 5.0E+00
      ......
```

FORTRAN-77

```
      ......

      IF (LAST .EQ. 2) THEN
         ERLARG = ERRSUM
         ERTEST = ERRBND
      ELSE
         ABSERR = ABSEPS
         RESULT = RESEPS
         IF (NOEXT) THEN
            ERLARG = ERRSUM
            ERTEST = ERRBND
         ELSE
            NRMAX = 1
            EPSABS = RESEPS
         ENDIF
      ENDIF
      RESULT = KTMIN
      SMALL = 5.0E+00
      ......
```

(2)

FORTRAN-66

```
      ......

      IF (ID .EQ. 2) GOTO 80
      IF (EXTALL) GOTO 90
      ERLARG = ERRBND
      ......
      GOTO 90
   80 CONTINUE
      SMALL = ABS (B - A)
      ERLARG = ERRSUM
      ERTEST = ERRBND
   90 CONTINUE
      ......
```

FORTRAN-77

```
      ......

      IF (ID .EQ. 2) THEN
         SMALL = ABS (B - A)
         ERLARG = ERRSUM
         ERTEST = ERRBND
      ELSE
         IF (.NOT. EXTALL) THEN
            ERLARG = ERRBND
            ......
         ENDIF
      ENDIF
      ......
```

(3)

FORTRAN-66

```
      ......
      IF(ABSERR .EQ. OFLOW) GOTO 115
      EXTRAP = .FALSE.
      SMALL = SMALL * 5.0E-01
      IF(IER .EQ. 0) GOTO 130
      ERLARG = ERRSUM
      GOTO 130
  115 CONTINUE
      ERTEST = ERRBND
      ERLARG = ERRMAX
  130 CONTINUE
      NRMAX = 1
      ......
```

FORTRAN-77

```
      ......
      IF(ABSERR .EQ. OFLOW) THEN
         ERTEST = ERRBND
         ERLARG = ERRMAX
      ELSE
         EXTRAP = .FALSE.
         SMALL = SMALL * 5.0E-01
         IF (IER .NE. 0) THEN
            ERLARG = ERRSUM
         ENDIF
      ENDIF
      NRMAX = 1
      ......
```

(4)

FORTRAN-66

```
      ......
      IF (LAST .EQ. 2) GOTO 120
      ERLARG = ERLARG - ERLAST
      EXTALL = .TRUE.
      IF (NOEXT) GOTO 130
      ID = NRMAX
      NUMRL2 = NUMRL2 + 1
      GOTO 140
  120 CONTINUE
      IER = ID - 1
      JUPBND = LAST
  130 CONTINUE
      SMALL = SMALL * 5.0E-01
      KTMIN = KTMIN + 1
  140 CONTINUE
      KSGN = 1
      ......
```

FORTRAN-77

```
      ......
      IF (LAST .NE. 2) THEN
         ERLARG = ERLARG - ERLAST
         EXTALL = .TRUE.
         IF (NOEXT) THEN
            SMALL = SMALL *
     X              5.0E-01
            KTMIN = KTMIN + 1
         ELSE
            ID = NRMAX
            NUMRL2 = NUMRL2 + 1
         ENDIF
      ELSE
         IER = ID - 1
         JUPBND = LAST
         SMALL = SMALL * 5.0E-01
         KTMIN = KTMIN + 1
      ENDIF
      KSGN = 1
      ......
```

(5)

FORTRAN-66

```
      ......
      IF (LAST .EQ. 2) GOTO 120
      ERLARG = ERLARG - ERLAST
      IF (EXTALL) GOTO 60
      SMALL = SMALL * 5.0E-01
      GOTO 130
   60 CONTINUE
      EXTRAP = .TRUE.
      NRMAX = 2
  120 CONTINUE
      SMALL = SMALL * 3.0E-01
      RLIST(NUMRL2) = AREA
  130 CONTINUE
      ERTEST = ERRBND
      ......
```

FORTRAN-77

```
      ......
      IF (LAST .NE. 2) THEN
         ERLARG = ERLARG - ERLAST
         IF (EXTALL) THEN
            EXTRAP = .TRUE.
            NRMAX = 2
            SMALL = SMALL * 3.0E-01
            RLIST(NUMRL2) = AREA
         ELSE
            SMALL = SMALL * 5.0E-01
         ENDIF
      ELSE
         SMALL = SMALL * 3.0E-01
         RLIST(NUMRL2) = AREA
      ENDIF
      ERTEST = ERRBND
      ......
```

APÊNDICE II

RESULTADOS OBTIDOS COM O PACOTE DAQWFT DA BITAN

- NOME DO PACOTE: DAQWFT

- NOME DA ROTINA EM FORTRAN-66 UTILIZADA PARA CONVERSÃO EM FORTRAN-77: DQFOUR

- NÚMERO DE LINHAS DA ROTINA EM FORTRAN-66: 472

- TEMPO MÉDIO DE TRANSFORMAÇÃO DA ROTINA PARA O FORTRAN-77 GASTO PELO SISCO: 20.01 segundos

- NÚMERO DE LINHAS DA ROTINA EM FORTRAN-77: 547

	TEMPO DE COMPILACAO		TEMPO DE EXECUCAO	
ORDEM	PACOTE COM ROTINA EM FORTRAN-66	PACOTE COM ROTINA EM FORTRAN-77	PACOTE COM ROTINA EM FORTRAN-66	PACOTE COM ROTINA EM FORTRAN-77
1	28.73	30.54	2.36	2.36
2	29.32	27.37	2.42	2.35
3	24.88	28.56	2.36	2.35
4	24.66	32.50	2.36	2.35
5	22.46	30.70	2.37	2.35
6	25.92	25.87	2.42	2.35
7	25.21	29.64	2.36	2.36
8	24.99	25.65	2.36	2.35
9	25.54	33.40	2.42	2.35
10	24.28	26.64	2.37	2.35
MEDIA	25.60	29.09	2.38	2.35

Podemos observar que o tempo médio de compilação é menor para o pacote que utiliza a rotina escrita em FORTRAN-66, enquanto que o tempo médio de execução é praticamente o mesmo para as duas versões do pacote.

APÊNDICE III

LISTAGENS DO PACOTE DAQWFT

LISTAGEM DO PACOTE DAQWFT UTILIZANDO DQFOUR EM FORTRAN-66

```
FOR-77 V3.0 Copyright WATCOM Systems Inc. 1984,1988  92/06/12 21:06:50
ions: list,disk,xtype,terminal,extensions,warnings,check,arraycheck

    1         INTEGER IER, INTEGR, NEVAL
    2         DOUBLE PRECISION A, ABSERR, B, EPSABS, ERRABS, F, OMEGA,
             X                    PI, RESULT
    3         DOUBLE PRECISION DATAN
    4         EXTERNAL F
      C
    5         A = 0.D0
    6         PI = DATAN (1.D0) * 4.D0
      C
    7         NPTS2 = 4
    8         OMEGA = .5D0 * PI
    9         INTEGR = 1
   10         EPSABS = 1.D-3
      C
   11         CALL DQAWF (F, A, OMEGA, INTEGR, EPSABS, RESULT,
             X             ABSERR, NEVAL, IER)
      C
   12         ERRABS = DABS (1.D0 - RESULT)
   13         WRITE (*,900) RESULT, ABSERR, ERRABS, NEVAL, IER
   14     900 FORMAT( ' APROXIMACAO DA INTEGRAL =', D24.16//
             X         ' ESTIMATIVA DO ERRO ABSOLUTO =', D9.2//
             X         ' ERRO ABSOLUTO REAL =', D9.2//
             X         ' NUMERO DA AVALIACOES DO INTEGRANDO =', I5//
             X         ' CODIGO DE RETORNO =', I2)
   15         STOP
   16         END

   17         DOUBLE PRECISION FUNCTION F(X)
   18         DOUBLE PRECISION X
      C
   19         DOUBLE PRECISION DSQRT
      C
   20         F = 0.D0
   21         IF (X .GT. 0.D0) F = 1.D0 / DSQRT(X)
   22         RETURN
   23         END

   24         SUBROUTINE DQAWF (F, A, OMEGA, INTEGR, EPSABS, RESULT,
             X                  ABSERR, NEVAL, IER)
      C
   25         INTEGER INTEGR, NEVAL, IER
   26         DOUBLE PRECISION F, A, OMEGA, EPSABS, RESULT, ABSERR
      C
      C      CALCULA UMA APROXIMACAO DE
      C           I = INTEGRAL DE F(X)*W(X) SOBRE (A,INFINITO)
      C      EM QUE
      C           W(X) = COS(OMEGA*X) OU
      C           W(X) = SIN(OMEGA*X),
      C      ESPERANDO QUE O ERRO COMETIDO SATISFACA
      C           ABS (I - RESULT) .LE. EPSABS.
      C
      C      PARAMETROS
      C
      C      NA CHAMADA
      C         F       DOUBLE PRECISION FUNCTION
```

```
C                E A FUNCAO INTEGRANDO DEFINIDA PELO USUARIO EM UM
C                SUBPROGRAMA FUNCAO. O NOME REAL DE F PRECISA SER
C                DECLARADO NUM EXTERNAL NO PROGRAMA ATIVADOR.
C         A      DOUBLE PRECISION
C                LIMITE INFERIOR DO INTERVALO DE INTEGRACAO
C         OMEGA  DOUBLE PRECISION
C                PARAMETRO DA FUNCAO PESO
C         INTEGR INTEGER
C                INDICA QUAL A FUNCAO PESO A SER UTILIZADA. SE INTEGR
C                = 1   W(X) = COS(OMEGA*X)
C                = 2   W(X) = SIN(OMEGA*X)
C                QUALQUER OUTRO VALOR DE INTEGR SINALIZARA ERRO.
C         EPSABS DOUBLE PRECISION
C                TOLERANCIA DE ERRO ABSOLUTO SOLICITADA
C         RESULT DOUBLE PRECISION
C                APENAS DECLARADO
C         ABSERR DOUBLE PRECISION
C                APENAS DECLARADO
C         NEVAL  INTEGER
C                APENAS DECLARADO
C         IER    INTEGER
C                APENAS DECLARADO
C
C      NO RETORNO
C         RESULT APROXIMACAO DA INTEGRAL
C         ABSERR ESTIMATIVA DO MODULO DO ERRO ABSOLUTO QUE DEVE
C                SER MAIOR OU IGUAL A ABS(I - RESULT).
C         NEVAL  O NUMERO DE AVALIACOES DO INTEGRANDO.
C         IER    SE O SEU VALOR FOR
C                = 0 TERMINO NORMAL DA ROTINA. SUPOE-SE QUE A
C                    TOLERANCIA DE ERRO SOLICITADA FOI SATISFEITA
C                .GT. 0 SIGNIFICA TERMINO ANORMAL. SUPOE-SE QUE
C                    A TOLERANCIA DE ERRO SOLICITADA FOI ALCANCADA.
C
C                SE OMEGA .NE. 0 E SE O VALOR DE IER
C                = 6 ALGUM OU ALGUNS PARAMETROS DE ENTRADA INVALIDO.
C                = 7 MAU COMPORTAMENTO DO INTEGRANDO OCORRE EM UM OU
C                    MAIS CICLOS. A LOCALIZACAO E O TIPO DE DIFICULDADE
C                    PODEM SER DETERMINADOS A PARTIR DO VETOR IERLST.
C                = 8 O MAXIMO NUMERO DE CICLOS PERMITIDOS FOI ATINGIDO,
C                    ISTO E, DE SUBINTERVALOS (A + (K-1)C, A + KC), EM
C                    QUE C = (2 * INT(ABS(OMEGA))+1)*PI/ABS(OMEGA), PARA
C                    K = 1, 2, ..., LST. PODE-SE PERMITIR MAIS CICLOS
C                    AUMENTANDO-SE O VALOR DE LIMLST E AJUSTANDO AS
C                    DIMENSOES DOS VETORES ENVOLVIDOS.
C                    EXAMINE O VETOR IERLST QUE CONTEM INFORMACOES SOBRE
C                    ERROS NOS CICLOS A FIM DE VERIFICAR DIFICULDADES DE
C                    INTEGRACAO.
C                    SE A POSICAO DE UMA DIFICULDADE LOCAL PUDER SER
C                    DETERMINADA (POR EXEMPLO, SINGULARIDADE OU DESCON-
C                    TINUIDADE NO INTERVALO), UMA POSSIVEL SOLUCAO SERIA
C                    DESDOBRAR O INTERVALO DE INTEGRACAO NESSE PONTO E
C                    CHAMAR ROTINAS APROPRIADAS EM CADA UM DOS
C                    SUBINTERVALOS.
C                = 9 A TABELA DE EXTRAPOLACAO CONSTRUIDA PARA A ACELE-
C                    RACAO DA CONVERGENCIA DA SERIE FORMADA PELAS
C                    CONTRIBUICOES DE INTEGRAIS SOBRE CADA CICLO
C                    CONVERGE COM A TOLERANCIA DE ERRO SOLICITADA.
C                    COMO NO CASO IER = 8, ACONSELHA-SE EXAMINAR O
C                    VETOR IERLST QUE CONTEM SINALIZACOES DE ERRO SOBRE
```

```
C                A RSLST(K).
C        LST     NUMERO DE SUBINTEGRAIS NECESSARIO PARA A INTEGRACAO.
C                SE OMEGA = 0, LST = 1.
C
C     SUBPROGRAMAS UTILIZADOS
C        DA BIBLIOTECA - DQAWFE
C        DO USUARIO - F
C
C     REFERENCIA
C        PIESSENS, DE DONCKER-KAPENGA, UBERHUBER, KAHANER, QUADPACK - A
C          SUBROUTINE PACKAGE FOR AUTOMATIC INTEGRATION, SPRINGER-
C          VERLAG, BERLIN HEIDELBERG N. YORK TOKYO, 1983
C
C                UNIVERSIDADE FEDERAL DA PARAIBA
C                DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C                PROJETO BITAN
C                MARIO T. HATTORI  ABR/90
C
      INTEGER LAST, LIMIT, LIMLST, IORD(500), IERLST(500), NNLOG(500),
     X        LST, MAXP1
      DOUBLE PRECISION ALIST(500), BLIST(500), CHEBMO(21,25),
     X                 ELIST(500), ERLST(500), RLIST(500), RSLST(500),
     X                 ZERO
      EXTERNAL F
      DATA ZERO /0.D0/
C
      IER = 0
      NEVAL = 0
      LAST = 0
      RESULT = ZERO
      ABSERR = ZERO
C
      MAXP1 = 21
      LIMLST = 50
      LIMIT = 500
C
C     VERIFICA VALIDADE DOS PARAMETROS
C
      IF (LIMLST .LT. 3 .OR. MAXP1 .LT. 1) RETURN
C
      CALL DQAWFE (F, A, OMEGA, INTEGR, EPSABS, LIMLST, LIMIT, MAXP1,
     X             RESULT, ABSERR, NEVAL, IER, RSLST, ERLST, IERLST,
     X             LST, ALIST, BLIST, RLIST, ELIST, IORD, NNLOG,
     X             CHEBMO)
C
      RETURN
      END

      SUBROUTINE DQAWFE (F, A, OMEGA, INTEGR, EPSABS, LIMLST,
     X                   LIMIT, MAXP1, RESULT, ABSERR, NEVAL,
     X                   IER, RSLST, ERLST, IERLST, LST,
     X                   ALIST, BLIST, RLIST, ELIST, IORD, NNLOG,
     X                   CHEBMO)
C
      INTEGER INTEGR, LIMLST, LIMIT, MAXP1, NEVAL, IER, IERLST(LIMLST),
     X        LST, IORD(LIMIT), NNLOG(LIMIT)
      DOUBLE PRECISION F, A, OMEGA, EPSABS, RESULT, ABSERR,
     X                 RSLST(LIMLST), ERLST(LIMLST),
     X                 ALIST(LIMIT), BLIST(LIMIT), RLIST(LIMIT),
     X                 ELIST(LIMIT), CHEBMO(MAXP1,25)
```

```
C
C      CALCULA UMA APROXIMACAO DE
C           I = INTEGRAL DE F(X)*W(X) SOBRE (A,INFINITO)
C      EM QUE
C           W(X) = COS(OMEGA*X) OU
C           W(X) = SIN(OMEGA*X),
C      ESPERANDO QUE O ERRO COMETIDO SATISFACA
C           ABS (I - RESULT) .LE. EPSABS.
C      A ROTINA E CHAMADA POR DQAWO E DQAWF. CONTUDO, PODE SER
C      CHAMADO DIRETAMENTE PELO USUARIO.
C
C      PARAMETROS
C
C      NA CHAMADA
C         F        DOUBLE PRECISION FUNCTION
C                  E A FUNCAO INTEGRANDO DEFINIDA PELO USUARIO EM UM
C                  SUBPROGRAMA FUNCAO. O NOME REAL DE F PRECISA SER
C                  DECLARADO NUM EXTERNAL NO PROGRAMA ATIVADOR.
C         A        DOUBLE PRECISION
C                  LIMITE INFERIOR DO INTERVALO DE INTEGRACAO
C         OMEGA    DOUBLE PRECISION
C                  PARAMETRO DA FUNCAO PESO
C         INTEGR   INTEGER
C                  INDICA QUAL A FUNCAO PESO A SER UTILIZADA. SE INTEGR
C                  = 1   W(X) = COS(OMEGA*X)
C                  = 2   W(X) = SIN(OMEGA*X)
C                  QUALQUER OUTRO VALOR DE INTEGR SINALIZARA ERRO.
C         EPSABS   DOUBLE PRECISION
C                  TOLERANCIA DE ERRO ABSOLUTO SOLICITADA
C         LIMLST   INTEGER
C                  DA UM LIMITE SUPERIOR NO NUMERO DE CICLOS.
C                  SE LIMLST MENOR QUE 3, A ROTINA TERMINA COM ERRO DE
C                  PARAMETRO.
C         LIMIT    INTEGER
C                  E O LIMITE SUPERIOR NO NUMERO DE SUBDIVISOES DO
C                  INTERVALO (A,B). DEVE SER MAIOR OU IGUAL 1.
C         MAXP1    INTEGER
C                  FORNECE UM LIMITE SUPERIOR NO NUMERO DE MOMENTOS DE
C                  CHEBYSHEV QUE PODE SER ARMAZENADO, ISTO E, PARA
C                  INTERVALOS DE TAMANHOS ABS(B-A)*2**(-L),
C                  L = 0, 1, 2, ... , MAXP1-2 COM MAXP1 MAIOR OU
C                  IGUAL 1. SE MAXP1 FOR MENOR QUE 1 DQAWFE TERMINARA
C                  SINALIZANDO OCORRENCIA DE ERRO.
C         RESULT   DOUBLE PRECISION
C                  APENAS DECLARADO
C         ABSERR   DOUBLE PRECISION
C                  APENAS DECLARADO
C         NEVAL    INTEGER
C                  APENAS DECLARADO
C         IER      INTEGER
C                  APENAS DECLARADO
C         RSLST    DOUBLE PRECISION(LIMLST)
C                  APENAS DECLARADO
C         ERLST    DOUBLE PRECISION(LIMLST)
C                  APENAS DECLARADO
C         IERLST   INTEGER (LIMLST)
C                  APENAS DECLARADO
C         LST      INTEGER
C                  APENAS DECLARADO
C         ALIST    DOUBLE PRECISION(LIMIT)
```

```
C                APENAS DECLARADO
C         BLIST  DOUBLE PRECISION(LIMIT)
C                APENAS DECLARADO
C         RLIST  DOUBLE PRECISION(LIMIT)
C                APENAS DECLARADO
C         ELIST  DOUBLE PRECISION(LIMIT)
C                APENAS DECLARADO
C         IORD   INTEGER(LIMIT)
C                APENAS DECLARADO
C         NNLOG  INTEGER(LIMIT)
C                APENAS DECLARADO
C         CHEBMO DOUBLE PRECISION(MAXP1,25)
C                APENAS DECLARADO NA PRIMEIRA CHAMADA
C
C     NO RETORNO
C         RESULT APROXIMACAO DA INTEGRAL
C         ABSERR ESTIMATIVA DO MODULO DO ERRO ABSOLUTO QUE DEVE
C                SER MAIOR OU IGUAL A ABS(I - RESULT).
C         NEVAL  O NUMERO DE AVALIACOES DO INTEGRANDO.
C         IER    SE O SEU VALOR FOR
C                = 0 TERMINO NORMAL DA ROTINA. SUPOE-SE QUE A
C                    TOLERANCIA DE ERRO SOLICITADA FOI SATISFEITA
C                .GT. 0 SIGNIFICA TERMINO ANORMAL. SUPOE-SE QUE
C                    A TOLERANCIA DE ERRO SOLICITADA FOI ALCANCADA.
C                SE OMEGA .NE. 0 E SE O VALOR DE IER
C                = 6 ALGUM OU ALGUNS PARAMETROS DE ENTRADA INVALIDO.
C                = 7 MAU COMPORTAMENTO DO INTEGRANDO OCORRE EM UM OU
C                    MAIS CICLOS. A LOCALIZACAO E O TIPO DE DIFICULDADE
C                    PODEM SER DETERMINADOS A PARTIR DO VETOR IERLST.
C                = 8 O MAXIMO NUMERO DE CICLOS PERMITIDOS FOI ATINGIDO,
C                    ISTO E, DE SUBINTERVALOS (A + (K-1)C, A + KC), EM
C                    QUE C = (2 * INT(ABS(OMEGA))+1)*PI/ABS(OMEGA), PARA
C                    K = 1, 2, ..., LST. PODE-SE PERMITIR MAIS CICLOS
C                    AUMENTANDO-SE O VALOR DE LIMLST E AJUSTANDO AS
C                    DIMENSOES DOS VETORES ENVOLVIDOS.
C                    EXAMINE O VETOR IERLST QUE CONTEM INFORMACOES SOBRE
C                    ERROS NOS CICLOS A FIM DE VERIFICAR DIFICULDADES DE
C                    INTEGRACAO.
C                    SE A POSICAO DE UMA DIFICULDADE LOCAL PUDER SER
C                    DETERMINADA (POR EXEMPLO, SINGULARIDADE OU DESCON-
C                    TINUIDADE NO INTERVALO), UMA POSSIVEL SOLUCAO SERIA
C                    DESDOBRAR O INTERVALO DE INTEGRACAO NESSE PONTO E
C                    CHAMAR ROTINAS APROPRIADAS EM CADA UM DOS
C                    SUBINTERVALOS.
C                = 9 A TABELA DE EXTRAPOLACAO CONSTRUIDA PARA A ACELE-
C                    RACAO DA CONVERGENCIA DA SERIE FORMADA CONTRIBUI-
C                    COES DE INTEGRAIS SOBRE CADA CICLO NAO CONVERGE
C                    COM A TOLERANCIA DE ERRO SOLICITADA.
C                    COMO NO CASO IER = 8, ACONSELHA-SE EXAMINAR O
C                    VETOR IERLST QUE CONTEM SINALIZACOES DE ERRO SOBRE
C                    OS CICLOS. SE O VALOR DE IERLST(K) FOR
C                    = 1 O MAXIMO NUMERO DE SUBDIVISOES (=LIMIT) FOI
C                        ATINGIDO NO K-EESIMO CICLO.
C                    = 2 OCORRENCIA DE ERRO DE ARREDONDAMENTO FOI
C                        DETETADA QUE IMPEDE QUE A TOLERANCIA DE ERRO
C                        SOLICITADA SEJA ALCANCADA NO K-ESIMO CICLO.
C                    = 3 MAU COMPORTAMENTO DO INTEGRANDO OCORRE EM
C                        ALGUNS PONTOS DO K-ESIMO CICLO.
C                    = 4 O PROCEDIMENTO DE INTEGRACAO NO K-ESIMO CICLO
C                        NAO CONVERGE DEVIDO AO ERRO DE ARREDONDAMENTO
```

```
C                       NA ROTINA DE EXTRAPOLACAO INVOCADA NO CICLO.
C                       SUPOE-SE QUE O RESULTADO E O MELHOR QUE SE
C                       POSSA ALCANCAR.
C                   = 5 A INTEGRAL NO K-ESIMO CICLO E PROVAVELMENTE
C                       DIVERGENTE OU LENTAMENTE CONVERGENTE.
C
C               SE OMEGA = 0 E INTEGR = 1, A INTEGRAL E CALCULADA PELA
C               ROTINA DQAGI E IER TERA OS SIGNIFICADOS DESCRITOS EM
C               DQAGI QUE SAO OS SEGUINTES. SE O VALOR DE IER FOR
C               = 1 O MAXIMO NUMERO SOLICITADO DE SUBDIVISOES  DO
C                   INTERVALO FOI ATINGIDO. PODE-SE PERMITIR MAIS
C                   SUBDIVISOES AUMENTANDO-SE O VALOR DE  LIMIT
C                   (PRECISA AJUSTAR DIMENSOES). SE ESSE AUMENTO NAO
C                   ACARRETAR MELHORIA NO RESULTADO ACONSELHA-SE
C                   ANALISAR O INTEGRANDO A FIM DE AVERIGUAR AS
C                   DIFICULDADES DE INTEGRACAO. SE A DIFICULDADE PUDER
C                   SER LOCALIZADA (POR EXEMPLO, SINGULARIDADE, DES-
C                   CONTINUIDADE DENTRO DO INTERVALO) VALERA A PENA
C                   SUBDIVIDIR O INTERVALO NESSE PONTO E CHAMAR O
C                   DQAGI EM CADA SUBINTERVALO. SE POSSIVEL, ESCOLHER
C                   UMA ROTINA ESPECIALIZADA QUE SEJA CAPAZ DE CONTOR-
C                   NAR A DIFICULDADE DETETADA.
C               = 2 FOI DETETADA A OCORRENCIA DE ERRO DE ARREDONDAMENTO
C                   QUE NAO PERMITE ATINGIR A TOLERANCIA DE ERRO
C                   SOLICITADA.
C               = 3 UM MAU COMPORTAMENTO EXTREMO DO INTEGRANDO OCORRE
C                   EM ALGUNS PONTOS NO INTERVALO DE INTEGRACAO.
C               = 4 O ALGORITMO NAO CONVERGE. ERRO DE ARREDONDAMENTO
C                   E DETETADO NA TABELA DE EXTRAPOLACAO. PRESUME-SE
C                   QUE A TOLERANCIA SOLICITADA NAO PODE SER ALCANCADA
C                   DEVIDO AO ERRO DE ARREDONDAMENTO NA TABELA E QUE
C                   O RESULTADO FORNECIDO E O MELHOR QUE SE PODE OBTER.
C               = 5 A INTEGRAL E PROVAVELMENTE DIVERGENTE OU LENTAMEN-
C                   TE CONVERGENTE. DEVE-SE OBSERVAR QUE A DIVERGENCIA
C                   PODE OCORRER COM QUALQUER OUTRO VALOR DE IER DIFE-
C                   RENTE DE ZERO.
C               = 6 HOUVE ERRO NOS DADOS DE ENTRADA (EPSABS E EPSREL
C                   NEGATIVOS,OU INTEGR NAO E 1 OU 2, OU ICALL MENOR
C                   QUE 1 OU AINDA MAXP1 MENOR QUE 1). OS VALORES DE
C                   RESULT, ABSERR, NEVAL, LAST, RLIST(1), ELIST(1),
C                   IORD(1) E NNLOG(1) SAO NULOS.
C       RSLST   CONTEM A CONTRIBUICAO DA INTEGRAL NO INTERVALO
C               (A + (K-1)C, A + KC), EM QUE
C                 . C = (2*INT(ABS(OMEGA))+1)*PI/ABS(OMEGA),
C               K = 1, 2, ... , LST.
C               OBSERVE QUE SE OMEGA = 0, RSLST(1) CONTERA O VALOR
C               TOTAL DA INTEGRAL, SOBRE (A, INFINITO).
C       ERLST   ERLST(K) CONTEM A ESTIMATIVA DO ERRO CORRESPONDENTE
C               A INTEGRAL EM RSLST(K).
C       IERLST  IERLST(K) CONTEM SINALIZACAO DE ERRO CORRESPONDENTE
C               RSLST(K).
C       LST     NUMERO DE SUBINTEGRAIS NECESSARIO PARA A INTEGRACAO.
C               SE OMEGA = 0, LST = 1.
C
C     SUBPROGRAMAS UTILIZADOS
C        DO FORTRAN - DABS, DATAN, DBLE, DMAX1, FLOAT
C        DA BIBLIOTECA - DMAQ, DQAGI, DQEXT, DQFOUR
C        DO USUARIO - F
C
C     REFERENCIA
```

```
C         PIESSENS, DE DONCKER-KAPENGA, UBERHUBER, KAHANER, QUADPACK - A
C           SUBROUTINE PACKAGE FOR AUTOMATIC INTEGRATION, SPRINGER-
C           VERLAG, BERLIN HEIDELBERG N. YORK TOKYO, 1983
C
C                  UNIVERSIDADE FEDERAL DA PARAIBA
C                  DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C                  PROJETO BITAN
C                  MARIO T. HATTORI  ABR/90
C
      INTEGER KTMIN, L, LL, MOMCOM, NEV, NRES, NUMRL2
      INTEGER IDINT
      REAL FLOAT
      DOUBLE PRECISION ABSEPS, CORREC, CYCLE, C1, C2, DL, DLA,
     X                 DRL, EP, EPMACH, EPS, EPSA, ERRSUM, FACT,
     X                 FIFTY, FOUR, OFLOW, ONE,
     X                 PI, P1, P9, PSUM(52),
     X                 RESEPS, RES3LA(3), TEN, TENM3, TENP3,
     X                 UFLOW, ZERO
      DOUBLE PRECISION DABS, DBLE, DMAQ, DMAX1, DATAN
      EXTERNAL F
      DATA TENM3 /1.D-3/, P9 /.9D0/
      DATA ZERO /0.D0/, ONE /1.D0/, FOUR /4.D0/,
     X     TEN /10.D0/, FIFTY /5.D1/, TENP3 /1.D3/
C
      PI = DATAN (ONE) * FOUR
      EPMACH = DMAQ (4)
      UFLOW = DMAQ (1)
      OFLOW = DMAQ (2)
C
C     VERIFICA VALIDADE DOS PARAMETROS
C
      RESULT = ZERO
      ABSERR = ZERO
      NEVAL = 0
      LST = 0
      IER = 0
      IF ((INTEGR .NE. 1 .AND. INTEGR .NE. 2) .OR. EPSABS .LT. ZERO
     X     .OR. LIMLST .LT. 3) IER = 6
C
C     ERRO NO(S) PARAMETRO(S), RETORNA.
C
      IF (IER .EQ. 6) RETURN
C
      IF (OMEGA .EQ. ZERO) THEN
C
C        INTEGRACAO POR DQAGI SE OMEGA ZERO
C
         IF (INTEGR .EQ. 1) THEN
            CALL DQAGI (F, ZERO, 1, EPSABS, ZERO, RESULT, ABSERR,
     X                  NEVAL, IER)
         ENDIF
         RSLST(1) = RESULT
         ERLST(1) = ABSERR
         IERLST(1) = IER
         LST = 1
         RETURN
      ENDIF
C
C     DEFINE CONDICOES INICIAIS
C
```

```
      L = IDINT (DABS (OMEGA))
      DL = DBLE (FLOAT (2*L+1))
      CYCLE = DL * PI / DABS (OMEGA)
      IER = 0
      KTMIN = 0
      NEVAL = 0
      NUMRL2 = 0
      NRES = 0
      C1 = A
      C2 = CYCLE + A
      P1 = ONE - P9
      EPS = EPSABS
      IF (EPSABS .GT. UFLOW / P1) EPS = EPSABS * P1
      EP = EPS
      FACT = ONE
      CORREC = ZERO
      ERRSUM = ZERO
      ABSERR = ZERO
C     MOMCOM = 0
C
C     CICLO PRINCIPAL
C
      DO 50 LST = 1,LIMLST
C
C        INTEGRA SOBRE O INTERVALO CORRENTE
C
         DLA = DBLE (FLOAT (LST))
         EPSA = EPS * FACT
         CALL DQFOUR (F, C1, C2, OMEGA, INTEGR, EPSA, ZERO,
     X                LIMIT, LST, MAXP1, RSLST(LST), ERLST(LST),
     X                NEV, IERLST(LST), ALIST, BLIST, RLIST,
     X                ELIST, IORD, NNLOG, MOMCOM, CHEBMO)
         NEVAL = NEVAL + NEV
         FACT = FACT * P9
         ERRSUM = ERRSUM + ERLST(LST)
         DRL = FIFTY * DABS (RSLST(LST))
C
C        TESTA EXATIDAO NA SOMA PARCIAL
C
         IF ((ERRSUM + DRL) .LE. EPSABS .AND. LST .GE. 6) THEN
C
C           RETORNA
C
            RESULT = PSUM(NUMRL2)
            ABSERR = ERRSUM + DRL
            RETURN
         ENDIF
         CORREC = DMAX1 (CORREC, ERLST(LST))
         IF (IERLST(LST) .NE. 0) THEN
            EPS = DMAX1 (EP, CORREC * P1)
            IER = 7
         ENDIF
         IF (IER .EQ. 7 .AND. (ERRSUM + DRL) .LE. CORREC * TEN .AND.
     X        LST .GT. 5) THEN
C
C           RETORNA
C
            RESULT = PSUM(NUMRL2)
            ABSERR = ERRSUM + DRL
            RETURN
```

```
115           ENDIF
    C
116           NUMRL2 = NUMRL2 + 1
117           IF (LST .LE. 1) THEN
118              PSUM(1) = RSLST(1)
119              GOTO 40
120           ENDIF
121           PSUM(NUMRL2) = PSUM(LL) + RSLST(LST)
    C
122           IF (LST .NE. 2) THEN
    C
    C              VERIFICA SE ATINGIU O MAXIMO NUMERO DE SUBINTERVALOS
    C              IF (LST .EQ. LIMLST) IER = 8
    C
    C              EXECUTA NOVA EXTRAPOLACAO
    C
123              CALL DQEXT (NUMRL2, PSUM, RESEPS, ABSEPS, RES3LA, NRES)
    C
    C              VERIFICA SE O RESULTADO EXTRAPOLADO E INFLUENCIADO PELO
    C              ARREDONDAMENTO
    C
124              KTMIN = KTMIN + 1
125              IF (KTMIN .GE. 15 .AND. ABSERR .LE. TENM3 *
       X                (ERRSUM + DRL)) IER = 9
126              IF (ABSEPS .LE. ABSERR .OR. LST .EQ. 3) THEN
127                 ABSERR = ABSEPS
128                 RESULT = RESEPS
129                 KTMIN = 0
    C
    C                 CASO IER NAO SEJA 0, VERIFICA SE O RESULTADO DIRETO OU
    C                 O RESULTADO EXTRAPOLADO FORNECE A MELHOR APROXIMACAO
    C                 DA INTEGRAL
    C
130                 IF ((ABSERR + TEN * CORREC) .LE. EPSABS .OR.
       X                    (ABSERR .LE. EPSABS .AND. TEN * CORREC .GE.
       X                     EPSABS)) GOTO 60
131              ENDIF
132              IF (IER .NE. 0 .AND. IER .NE. 7) GOTO 60
133           ENDIF
134       40  CONTINUE
135           LL = NUMRL2
136           C1 = C2
137           C2 = C2 + CYCLE
138       50 CONTINUE
    C
139       60 CONTINUE
    C
    C     DEFINE RESULTADO E ESTIMATIVA DE ERRO FINAIS
    C
140        ABSERR = ABSERR + TEN * CORREC
141        IF (IER .EQ. 0) RETURN
142        IF (RESULT .EQ. ZERO .OR. PSUM(NUMRL2) .EQ. ZERO) THEN
143           IF (ABSERR .GT. ERRSUM) THEN
144              RESULT = PSUM(NUMRL2)
145              ABSERR = ERRSUM + DRL
146              RETURN
147           ENDIF
148        ENDIF
149        IF (ABSERR / DABS(RESULT) .LE. (ERRSUM + DRL) /
       X     DABS (PSUM(NUMRL2))) THEN
```

```
150           RESULT = PSUM(NUMRL2)
151           ABSERR = ERRSUM + DRL
152           RETURN
153         ENDIF
154         IF (IER .GE. 1 .AND. IER .NE. 7) ABSERR = ABSERR + DRL
155         RETURN
156         END

157         SUBROUTINE DQAGI (F, BOUND, INF, EPSABS, EPSREL, RESULT,
           X                  ABSERR, NEVAL, IER)
      C
158         INTEGER INF, NEVAL, IER
159         DOUBLE PRECISION F, BOUND, EPSABS, EPSREL, RESULT,
           X                 ABSERR
      C
      C     CALCULA UMA APROXIMACAO DE
      C          I = INTEGRAL DE F(X) SOBRE (BOUND, +INFINITO)
      C       OU I = INTEGRAL DE F(X) SOBRE (-INFINITO, +INFINITO)
      C       OU I = INTEGRAL DE F(X) SOBRE (-INFINITO, BOUND)
      C     ESPERANDO QUE O ERRO COMETIDO SATISFACA
      C          ABS (I - RESULT) .LE. MAX (EPSABS, EPSREL*ABS(I)).
      C
      C     PARAMETROS
      C
      C     NA CHAMADA
      C        F        DOUBLE PRECISION FUNCTION
      C                 E A FUNCAO INTEGRANDO DEFINIDA PELO USUARIO EM UM
      C                 SUBPROGRAMA FUNCAO. O NOME REAL DE F PRECISA SER
      C                 DECLARADO NUM EXTERNAL NO PROGRAMA ATIVADOR.
      C        BOUND    DOUBLE PRECISION
      C                 LIMITE FINITO DO INTERVALO DE INTEGRACAO
      C                 NAO TEM SIGNIFICADO SE O INTERVALO FOR
      C                 (-INFINITO, +INFINITO)
      C        INF      INTEGER
      C                 INDICA O TIPO DE INTERVALO DE INTEGRACAO.
      C                  INF = 1 CORRESPONDE A (BOUND,+INFINITO)
      C                  INF = 2 CORRESPONDE A (-INFINITO, BOUND)
      C                  INF = 3 CORRESPONDE A (-INFINITO, +INFINITO)
      C        EPSABS   DOUBLE PRECISION
      C                 TOLERANCIA DE ERRO ABSOLUTO SOLICITADA
      C        EPSREL   DOUBLE PRECISION
      C                 TOLERANCIA DE ERRO RELATIVO SOLICITADA. SE EPSABS E
      C                 EPSREL FOREM NEGATIVOS DQFOUR SINALIZARA A OCORRENCIA
      C                 DE ERRO.
      C        RESULT   DOUBLE PRECISION
      C                 APENAS DECLARADO
      C        ABSERR   DOUBLE PRECISION
      C                 APENAS DECLARADO
      C        NEVAL    INTEGER
      C                 APENAS DECLARADO
      C        IER      INTEGER
      C                 APENAS DECLARADO
      C
      C     NO RETORNO
      C        RESULT   APROXIMACAO DA INTEGRAL
      C        ABSERR   ESTIMATIVA DO MODULO DO ERRO ABSOLUTO QUE DEVE
      C                 SER MAIOR OU IGUAL A ABS(I - RESULT).
      C        NEVAL    O NUMERO DE AVALIACOES DO INTEGRANDO.
      C        IER      SE O SEU VALOR FOR
      C                 = 0 TERMINO NORMAL DA ROTINA. SUPOE-SE QUE A
```

```
C                    TOLERANCIA DE ERRO SOLICITADA FOI SATISFEITA
C                = 1 O MAXIMO NUMERO SOLICITADO DE SUBDIVISOES  DO
C                    INTERVALO FOI ATINGIDO. PODE-SE PERMITIR MAIS
C                    SUBDIVISOES AUMENTANDO-SE O VALOR DE  LIMIT
C                    (PRECISA AJUSTAR DIMENSOES). SE ESSE AUMENTO NAO
C                    ACARRETAR MELHORIA NO RESULTADO ACONSELHA-SE
C                    ANALISAR O INTEGRANDO A FIM DE AVERIGUAR AS
C                    DIFICULDADES DE INTEGRACAO. SE A DIFICULDADE PUDER
C                    SER LOCALIZADA (POR EXEMPLO, SINGULARIDADE, DES-
C                    CONTINUIDADE DENTRO DO INTERVALO) VALERA A PENA
C                    SUBDIVIDIR O INTERVALO NESSE PONTO E CHAMAR O
C                    DQFOUR EM CADA SUBINTERVALO. SE POSSIVEL, ESCOLHER
C                    UMA ROTINA ESPECIALIZADA QUE SEJA CAPAZ DE CONTOR-
C                    NAR A DIFICULDADE DETETADA.
C                = 2 FOI DETETADA A OCORRENCIA DE ERRO DE ARREDONDAMENTO
C                    QUE NAO PERMITE ATINGIR A TOLERANCIA DE ERRO
C                    SOLICITADA.
C                = 3 UM MAU COMPORTAMENTO EXTREMO DO INTEGRANDO OCORRE
C                    EM ALGUNS PONTOS NO INTERVALO DE INTEGRACAO.
C                = 4 O ALGORITMO NAO CONVERGE. ERRO DE ARREDONDAMENTO
C                    E DETETADO NA TABELA DE EXTRAPOLACAO. PRESUME-SE
C                    QUE A TOLERANCIA SOLICITADA NAO PODE SER ALCANCADA
C                    DEVIDO AO ERRO DE ARREDONDAMENTO NA TABELA E QUE
C                    O RESULTADO FORNECIDO E O MELHOR QUE SE PODE OBTER.
C                = 5 A INTEGRAL E PROVAVELMENTE DIVERGENTE OU LENTAMEN-
C                    TE CONVERGENTE. DEVE-SE OBSERVAR QUE A DIVERGENCIA
C                    PODE OCORRER COM QUALQUER OUTRO VALOR DE IER DIFE-
C                    RENTE DE ZERO.
C                = 6 HOUVE ERRO NOS DADOS DE ENTRADA (EPSABS E EPSREL
C                    NEGATIVOS) OS VALORES DERESULT, ABSERR E NEVAL
C                    SERAO NULOS.
C
C     SUBPROGRAMAS UTILIZADOS
C        DO FORTRAN - DABS, DMAX1
C        DA BIBLIOTECA - DMAQ, DQK15I, DQSORT, DQEXT
C        DO USUARIO - F
C
C     REFERENCIA
C        PIESSENS, DE DONCKER-KAPENGA, UBERHUBER, KAHANER, QUADPACK - A
C          SUBROUTINE PACKAGE FOR AUTOMATIC INTEGRATION, SPRINGER-
C          VERLAG, BERLIN HEIDELBERG N. YORK TOKYO, 1983
C
C                  UNIVERSIDADE FEDERAL DA PARAIBA
C                  DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C                  PROJETO BITAN
C                  MARIO T. HATTORI  MAI/90
C
      INTEGER ID, IERRO, IORD(500), IROFF1, IROFF2, IROFF3,
     X        JUPBND, K, KSGN, KTMIN, LAST, LIMIT, MAXERR,
     X        NRES, NRMAX, NUMRL2
      LOGICAL EXTRAP, NOEXT
      DOUBLE PRECISION ABSEPS, ALIST(500), AREA, AREA1, AREA12, AREA2,
     X                 A1, A2, BLIST(500), BOUN, B1, B2, CORREC, DEFAB1,
     X                 DEFAB2, DEFABS, DRES, ELIST(500), EPMACH, ERLARG,
     X                 ERLAST, ERRBND, ERRMAX, ERROR1, ERR012,
     X                 ERROR2, ERRSUM, ERTEST, FIFTY, HUNDRD, OFLOW,
     X                 ONE, P375, P99, RESABS, RESEPS, RES3LA(3),
     X                 RLIST(500), RLIST2(52), SMALL,TENM2, TENM3,
     X                 TENM5, TENP3, TWO, UFLOW, ZERO
      DOUBLE PRECISION DABS, DMAQ, DMAX1
```

```
      EXTERNAL F
      DATA LIMIT /500/
      DATA TENM5 /1.D-5/, TENM3 /1.D-3/, TENM2 /1.D-2/, P99 /.99D0/,
     X     P375 /.375D0/
      DATA ZERO /0.D0/, ONE /1.D0/, TWO /2.D0/, FIFTY /5.D1/,
     X     HUNDRD /100.D0/, TENP3 /1.D3/
C
      EPMACH = DMAQ (4)
      OFLOW = DMAQ (2)
      UFLOW = DMAQ (1)
C
C     TESTA VALIDADE DOS PARAMETROS
C
      IER = 0
      NEVAL = 0
      LAST = 0
      RESULT = ZERO
      ABSERR = ZERO
      ALIST(1) = ZERO
      BLIST(1) = ONE
      RLIST(1) = ZERO
      ELIST(1) = ZERO
      IORD(1) = 0
      IF (EPSABS .LT. ZERO .AND. EPSREL .LT. ZERO) IER = 6
      IF (IER .EQ. 6) THEN
         RETURN
      ENDIF
C
C     PRIMEIRA APROXIMACAO DA INTEGRAL
C
C     DETERMINA O INTERVALO A SER MAPEADO EM (0,1). SE INF = 2 A
C     INTEGRAL E COMPUTADA COMO I = I1 + I2, EM QUE
C        I1 = INTEGRAL DE F SOBRE (-INFINITO, 0)
C        I2 = INTEGRAL DE F SOBRE (0, +INFINITO)
C
      BOUN = BOUND
      IF (INF .EQ. 2) BOUN = ZERO
      CALL DQK15I (F, BOUN, INF, ZERO, ONE, RESULT, ABSERR,
     X             DEFABS, RESABS)
C
C     TESTA EXATIDAO
C
      LAST = 1
      RLIST(1) = RESULT
      ELIST(1) = ABSERR
      IORD(1) = 1
      DRES = DABS (RESULT)
      ERRBND = DMAX1 (EPSABS, EPSREL * DRES)
      IF (ABSERR .LE. HUNDRD * EPMACH * DEFABS .AND.
     X    ABSERR .GT. ERRBND) IER = 2
      IF (LIMIT .EQ. 1) IER = 1
      IF (IER .NE. 0 .OR. (ABSERR .LE. ERRBND .AND. ABSERR .NE. RESABS)
     X    .OR. ABSERR .EQ. ZERO) THEN
C
C        TOLERANCIA DE ERRO SATISFEITA OU OCORREU ERRO. RETORNA.
C
         NEVAL = 30 * LAST - 15
         IF (INF .EQ. 2) NEVAL = 2 * NEVAL
         IF (IER .GT. 2) IER = IER - 1
         RETURN
```

```
201         ENDIF
    C
    C       DEFINE CONDICOES INICIAIS
    C
202         RLIST2(1) = RESULT
203         ERRMAX = ABSERR
204         MAXERR = 1
205         AREA = RESULT
206         ERRSUM = ABSERR
207         ABSERR = OFLOW
208         NRMAX = 1
209         NRES = 0
210         KTMIN = 0
211         NUMRL2 = 2
212         EXTRAP = .FALSE.
213         NOEXT = .FALSE.
214         IERRO = 0
215         IROFF1 = 0
216         IROFF2 = 0
217         IROFF3 = 0
218         KSGN = -1
219         IF (DRES .GE. (ONE - FIFTY * EPMACH) * DEFABS) KSGN = 1
    C
    C       CICLO PRINCIPAL
    C
220         DO 90 LAST = 2,LIMIT
    C
    C           BISSECCIONA O SUBINTERVALO COM A ESTIMATIVA DE ERRO MAIOR
    C
221             A1 = ALIST(MAXERR)
222             B1 = (ALIST(MAXERR) + BLIST(MAXERR)) / TWO
223             A2 = B1
224             B2 = BLIST(MAXERR)
225             ERLAST = ERRMAX
226             CALL DQK15I (F, BOUN, INF, A1, B1, AREA1, ERROR1,
        X                        RESABS, DEFAB1)
227             CALL DQK15I (F, BOUN, INF, A2, B2, AREA2, ERROR2,
        X                        RESABS, DEFAB2)
    C
    C           MELHORA AS APROXIMACOES ANTERIORES DA INTEGRAL E DO ERRO E FAZ
    C           TESTE DE EXATIDAO
    C
228             AREA12 = AREA1 + AREA2
229             ERRO12 = ERROR1 + ERROR2
230             ERRSUM = ERRSUM + ERRO12 - ERRMAX
231             AREA = AREA + AREA12 - RLIST(MAXERR)
    C
232             IF (DEFAB1 .NE. ERROR1 .AND. DEFAB2 .NE. ERROR2) THEN
233                IF (DABS (RLIST(MAXERR) - AREA12) .LE. TENM5 * DABS (AREA12)
        X                .AND. ERRO12 .GE. P99 * ERRMAX) THEN
234                   IF (EXTRAP) THEN
235                      IROFF2 = IROFF2 + 1
236                   ELSE
237                      IROFF1 = IROFF1 + 1
238                   ENDIF
239                ENDIF
240                IF (LAST .GT. 10 .AND. ERRO12 .GT. ERRMAX)
        X                                          IROFF3 = IROFF3 + 1
 41             ENDIF
    C
```

```
242               RLIST(MAXERR) = AREA1
243               RLIST(LAST) = AREA2
244               ERRBND = DMAX1 (EPSABS, EPSREL * DABS (AREA))
      C
      C           TESTA ERRO DE ARREDONDAMENTO E EVENTUALMENTE SINALIZA
      C           ERRO
      C
245               IF (IROFF1 + IROFF2 .GE. 10 .OR. IROFF3 .GE. 20) IER = 2
246               IF (IROFF2 .GE. 5) IERRO = 3
      C
      C           NUMERO DE SUBINTERVALOS ATINGIU LIMIT
      C
247               IF (LAST .EQ. LIMIT) IER = 1
      C
      C           INTEGRANDO TEM MAU COMPORTAMENTO EM UM PONTO DO
      C           INTERVALO DE INTEGRACAO
      C
248               IF (DMAX1 (DABS (A1), DABS (B2)) .LE. (ONE +
     X                    TENP3 * EPMACH) * (DABS (A2) + TENP3 * UFLOW))
     X                                          IER = 4
      C
      C           ANEXA OS INTERVALOS RECEM-CRIADOS A LISTA
      C
249               IF (ERROR2. LE. ERROR1) THEN
250                  ALIST(LAST) = A2
251                  BLIST(MAXERR) = B1
252                  BLIST(LAST) = B2
253                  ELIST(MAXERR) = ERROR1
254                  ELIST(LAST) = ERROR2
255               ELSE
256                  ALIST(MAXERR) = A2
257                  ALIST(LAST) = A1
258                  BLIST(LAST) = B1
259                  RLIST(MAXERR) = AREA2
260                  RLIST(LAST) = AREA1
261                  ELIST(MAXERR) = ERROR2
262                  ELIST(LAST) = ERROR1
263               ENDIF
      C
      C           MANTEM A ORDENACAO DESCENDENTE NA LISTA DE ESTIMATIVAS DE ERRO
      C           E SELECIONA O SUBINTERVALO COM ESTIMATIVA DE ERRO MAIOR
      C           (PARA SER BISSECCIONADA EM SEGUIDA)
      C
264               CALL DQSORT (LIMIT, LAST, MAXERR, ERRMAX, ELIST,
     X                       IORD, NRMAX)
265               IF (ERRSUM .LE. ERRBND) GOTO 115
266               IF (IER .NE. 0) GOTO 100
267               IF (LAST .EQ. 2) THEN
268                  SMALL = P375
269                  ERLARG = ERRSUM
270                  ERTEST = ERRBND
271                  RLIST2(2) = AREA
272               ELSE
273                  IF (.NOT. NOEXT) THEN
274                     ERLARG = ERLARG - ERLAST
275                     IF (DABS (B1 - A1) .GT. SMALL) ERLARG = ERLARG + ERRO12
276                     IF (.NOT. EXTRAP) THEN
      C
      C                    TESTA SE O PROXIMO INTERVALO A SER BISSECCIONADO E
      C                    O MENOR
```

```
C
277                     IF (DABS (BLIST(MAXERR) - ALIST(MAXERR)) .GT. SMALL)
     X                        GOTO 90
278                     EXTRAP = .TRUE.
279                     NRMAX = 2
280                  ENDIF
281                  IF (IERRO .NE. 3 .AND. ERLARG .GT. ERTEST) THEN
C
C                       O MENOR INTERVALO TEM  O MAIOR ERRO. ANTES DE
C                       BISSECCIONAR DIMINUI A SOMA DOS ERROS SOBRE OS
C                       INTERVALOS MAIORES (ERLARG) E EXECUTA EXTRAPOLACAO.
C
282                     ID = NRMAX
283                     JUPBND = LAST
284                     IF (LAST .GT. (2 + LIMIT / 2)) JUPBND = LIMIT +
     X                      3 - LAST
285                     DO 50 K = ID,JUPBND
286                        MAXERR = IORD(NRMAX)
287                        ERRMAX = ELIST(MAXERR)
288                        IF (DABS (BLIST(MAXERR) - ALIST(MAXERR)) .GT.
     X                         SMALL) GOTO 90
289                        NRMAX = NRMAX + 1
290                50      CONTINUE
291                  ENDIF
C
C                    EXECUTA EXTRAPOLACAO
C
292                  NUMRL2 = NUMRL2 + 1
293                  RLIST2(NUMRL2) = AREA
294                  CALL DQEXT (NUMRL2, RLIST2, RESEPS, ABSEPS, RES3LA, NRES)
295                  KTMIN = KTMIN + 1
296                  IF (KTMIN .GT. 5 .AND. ABSERR .LT. TENM3 * ERRSUM)
     X                        IER = 5
297                  IF (ABSEPS .LT. ABSERR) THEN
298                     KTMIN = 0
299                     ABSERR = ABSEPS
300                     RESULT = RESEPS
301                     CORREC = ERLARG
302                     ERTEST = DMAX1 (EPSABS, EPSREL * DABS (RESEPS))
303                     IF (ABSERR .LE. ERTEST) GOTO 100
C
C                       PREPARA BISSECCAO DO MENOR INTERVALO
C
304                  ENDIF
305                  IF (NUMRL2 .EQ. 1) NOEXT = .TRUE.
306                  IF (IER .EQ. 5) GOTO 100
307                  MAXERR = IORD(1)
308                  ERRMAX = ELIST(MAXERR)
309                  NRMAX = 1
310                  EXTRAP = .FALSE.
311                  SMALL = SMALL / TWO
312                  ERLARG = ERRSUM
313               ENDIF
314            ENDIF
315    90 CONTINUE
C
C      COMPUTA O RESULTADO FINAL
C
316   100 CONTINUE
 17       IF (ABSERR .EQ. OFLOW) GOTO 115
```

```
318         IF (IER + IERRO .EQ. 0) GOTO 110
319         IF (IERRO .EQ. 3) ABSERR = ABSERR + CORREC
320         IF (IER .EQ. 0) IER = 3
321         IF (RESULT .NE. ZERO .AND. AREA .NE. ZERO) GOTO 105
322         IF (ABSERR .GT. ERRSUM) GOTO 115
323         IF (AREA .EQ. ZERO) GOTO 130
324         GOTO 110
325   105 CONTINUE
326         IF (ABSERR / DABS (RESULT) .GT. ERRSUM / DABS (AREA)) GOTO 115
    C
    C     TESTA DIVERGENCIA
    C
327   110 CONTINUE
328         IF (KSGN .EQ. (-1) .AND. DMAX1 (DABS (RESULT), DABS (AREA)) .LE.
          X    DEFABS * TENM2) GOTO 130
329         IF (TENM2 .GT. (RESULT / AREA) .OR. (RESULT / AREA) .GT. HUNDRD
          X     .OR. ERRSUM .GT. DABS (AREA)) IER = 6
330         GOTO 130
331   115 CONTINUE
    C
    C     COMPUTA A INTEGRAL
    C
332         RESULT = ZERO
333         DO 120 K = 1,LAST
334            RESULT = RESULT + RLIST(K)
335   120 CONTINUE
336         ABSERR = ERRSUM
337   130 CONTINUE
338         NEVAL = 30 * LAST - 15
339         IF (IER .GT. 2) IER = IER - 1
340         RETURN
341         END
    C

342         SUBROUTINE DQFOUR (F, A, B, OMEGA, INTEGR, EPSABS, EPSREL, LIMIT,
          X                    ICALL, MAXP1, RESULT, ABSERR, NEVAL, IER,
          X                    ALIST, BLIST, RLIST, ELIST, IORD, NNLOG,
          X                    MOMCOM, CHEBMO)
    C
343         INTEGER INTEGR, LIMIT, ICALL, MAXP1, NEVAL, IER, IORD(LIMIT),
          X         NNLOG(LIMIT), MOMCOM
344         DOUBLE PRECISION F, A, B, OMEGA, EPSABS, EPSREL, RESULT, ABSERR,
          X                 ALIST(LIMIT), BLIST(LIMIT), RLIST(LIMIT),
          X                 ELIST(LIMIT), CHEBMO(MAXP1,25)
    C
    C     CALCULA UMA APROXIMACAO DE
    C          I = INTEGRAL DE F(X)*W(X) SOBRE (A,B),
    C     EM QUE
    C          W(X) = COS(OMEGA*X) OU
    C          W(X) = SIN(OMEGA*X),
    C     ESPERANDO QUE O ERRO COMETIDO SATISFACA
    C          ABS (I - RESULT) .LE. MAX (EPSABS, EPSREL*ABS(I)).
    C     A ROTINA E CHAMADA POR DQAWO E DQAWF. CONTUDO, PODE SER
    C     CHAMADO DIRETAMENTE PELO USUARIO.
    C
    C     PARAMETROS
    C
    C     NA CHAMADA
    C        F       DOUBLE PRECISION FUNCTION
    C                E A FUNCAO INTEGRANDO DEFINIDA PELO USUARIO EM UM
```

```
C               SUBPROGRAMA FUNCAO. O NOME REAL DE F PRECISA SER
C               DECLARADO NUM EXTERNAL NO PROGRAMA ATIVADOR.
C         A     DOUBLE PRECISION
C               LIMITE INFERIOR DO INTERVALO DE INTEGRACAO
C         B     DOUBLE PRECISION
C               LIMITE SUPERIOR DO INTERVALO DE INTEGRACAO
C         OMEGA DOUBLE PRECISION
C               PARAMETRO DA FUNCAO PESO
C         INTEGR INTEGER
C               INDICA QUAL A FUNCAO PESO A SER UTILIZADA. SE INTEGR
C               = 1  W(X) = COS(OMEGA*X)
C               = 2  W(X) = SIN(OMEGA*X)
C               QUALQUER OUTRO VALOR DE INTEGR SINALIZARA ERRO.
C         EPSABS DOUBLE PRECISION
C               TOLERANCIA DE ERRO ABSOLUTO SOLICITADA
C         EPSREL DOUBLE PRECISION
C               TOLERANCIA DE ERRO RELATIVO SOLICITADA. SE EPSABS E
C               EPSREL FOREM NEGATIVOS DQFOUR SINALIZARA A OCORRENCIA
C               DE ERRO.
C         LIMIT INTEGER
C               E O LIMITE SUPERIOR NO NUMERO DE SUBDIVISOES DO
C               INTERVALO (A,B). DEVE SER MAIOR OU IGUAL A 1.
C         ICALL INTEGER
C               SE DQFOUR FOR SER USADO UMA UNICA VEZ, ICALL DEVE TER
C               VALOR 1. SUPONHA QUE DURANTE ESSA CHAMADA UNICA OS
C               MOMENTOS DE CHEBYSHEV (PARA O METODO DE INTEGRACAO
C               DE CLENSHAW-CURTIS DE GRAU 24) FORAM CALCULADOS PARA
C               INTERVALOS DE COMPRIMENTOS (ABS(B-A))*2**(-L),
C               L = 0, 1, 2, ... , MOMCOM-1. OS MOMENTOS DE CHEBYSHEV
C               JA CALCULADOS PODEM SER SER RE-UTILIZADOS EM CHAMADAS
C               SUBSEQUENTES, SE DQFOUR DEVE SER CHAMADO DUAS OU MAIS
C               VEZES NO INTERVALO DE MESMO TAMANHO ABS(B-A). A PARTIR
C               DA SEGUNDA CHAMADA DEVE-SE USAR ICALL MAIOR QUE 1. SE
C               ICALL FOR MENOR QUE 1 A ROTINA SINALIZARA UMA
C               OCORRENCIA DE ERRO.
C         MAXP1 INTEGER
C               FORNECE UM LIMITE SUPERIOR NO NUMERO DE MOMENTOS DE
C               CHEBYSHEV QUE PODE SER ARMAZENADO, ISTO E, PARA
C               INTERVALOS DE TAMANHOS ABS(B-A)*2**(-L),
C               L = 0, 1, 2, ... , MAXP1-2 COM MAXP1 MAIOR OU
C               IGUAL A 1. SE MAXP1 FOR MENOR QUE 1 DQFOUR TERMINARA
C               SINALIZANDO OCORRENCIA DE ERRO.
C         RESULT DOUBLE PRECISION
C               APENAS DECLARADO
C         ABSERR DOUBLE PRECISION
C               APENAS DECLARADO
C         NEVAL INTEGER
C               APENAS DECLARADO
C         IER   INTEGER
C               APENAS DECLARADO
C         ALIST DOUBLE PRECISION(LIMIT)
C               APENAS DECLARADO
C         BLIST DOUBLE PRECISION(LIMIT)
C               APENAS DECLARADO
C         RLIST DOUBLE PRECISION(LIMIT)
C               APENAS DECLARADO
C         ELIST DOUBLE PRECISION(LIMIT)
C               APENAS DECLARADO
C         IORD  INTEGER(LIMIT)
C               APENAS DECLARADO
```

```
C          NNLOG   INTEGER(LIMIT)
C                  APENAS DECLARADO
C          MOMCOM  INTEGER
C                  NA PRIMEIRA CHAMADA DEVE TER VALOR ZERO
C          CHEBMO  DOUBLE PRECISION(MAXP1,25)
C                  APENAS DECLARADO NA PRIMEIRA CHAMADA
C
C      NO RETORNO
C          RESULT  APROXIMACAO DA INTEGRAL
C          ABSERR  ESTIMATIVA DO MODULO DO ERRO ABSOLUTO QUE DEVE
C                  SER MAIOR OU IGUAL A ABS(I - RESULT).
C          NEVAL   O NUMERO DE AVALIACOES DO INTEGRANDO.
C          IER     SE O SEU VALOR FOR
C                  = 0 TERMINO NORMAL DA ROTINA. SUPOE-SE QUE A
C                      TOLERANCIA DE ERRO SOLICITADA FOI SATISFEITA
C                  = 1 O MAXIMO NUMERO SOLICITADO DE SUBDIVISOES DO
C                      INTERVALO FOI ATINGIDO. PODE-SE PERMITIR MAIS
C                      SUBDIVISOES AUMENTANDO-SE O VALOR DE LIMIT
C                      (PRECISA AJUSTAR DIMENSOES). SE ESSE AUMENTO NAO
C                      ACARRETAR MELHORIA NO RESULTADO ACONSELHA-SE
C                      ANALISAR O INTEGRANDO A FIM DE AVERIGUAR AS
C                      DIFICULDADES DE INTEGRACAO. SE A DIFICULDADE PUDER
C                      SER LOCALIZADA (POR EXEMPLO, SINGULARIDADE, DES-
C                      CONTINUIDADE DENTRO DO INTERVALO) VALERA A PENA
C                      SUBDIVIDIR O INTERVALO NESSE PONTO E CHAMAR O
C                      DQFOUR EM CADA SUBINTERVALO. SE POSSIVEL, ESCOLHER
C                      UMA ROTINA ESPECIALIZADA QUE SEJA CAPAZ DE CONTOR-
C                      NAR A DIFICULDADE DETETADA.
C                  = 2 FOI DETETADA A OCORRENCIA DE ERRO DE ARREDONDAMENTO
C                      QUE NAO PERMITE ATINGIR A TOLERANCIA DE ERRO
C                      SOLICITADA.
C                  = 3 UM MAU COMPORTAMENTO EXTREMO DO INTEGRANDO OCORRE
C                      EM ALGUNS PONTOS NO INTERVALO DE INTEGRACAO.
C                  = 4 O ALGORITMO NAO CONVERGE. ERRO DE ARREDONDAMENTO
C                      E DETETADO NA TABELA DE EXTRAPOLACAO. PRESUME-SE
C                      QUE A TOLERANCIA SOLICITADA NAO PODE SER ALCANCADA
C                      DEVIDO AO ERRO DE ARREDONDAMENTO NA TABELA E QUE
C                      O RESULTADO FORNECIDO E O MELHOR QUE SE PODE OBTER.
C                  = 5 A INTEGRAL E PROVAVELMENTE DIVERGENTE OU LENTAMEN-
C                      TE CONVERGENTE. DEVE-SE OBSERVAR QUE A DIVERGENCIA
C                      PODE OCORRER COM QUALQUER OUTRO VALOR DE IER DIFE-
C                      RENTE DE ZERO.
C                  = 6 HOUVE ERRO NOS DADOS DE ENTRADA (EPSABS  E EPSREL
C                      NEGATIVOS, OU INTEGR NAO 1 OU 2, OU ICALL MENOR QUE
C                      1 OU AINDA MAXP1 MENOR QUE 1). OS VALORES DE
C                      RESULT, ABSERR, NEVAL, LAST, RLIST(1), ELIST(1),
C                      IORD(1) E NNLOG(1) SAO NULOS. ALIST(1) E BLIST(1)
C                      CONTERAO OS VALORES DE A E DE B, RESPECTIVAMENTE.
C          ALIST   OS PRIMEIROS LAST ELEMENTOS SAO OS LIMITES INFERIORES
C                  DOS SUBINTERVALOS NA PARTICAO DO INTERVALO (A,B) DE
C                  INTEGRACAO.
C          BLIST   OS PRIMEIROS LAST ELEMENTOS SAO OS LIMITES SUPERIORES
C                  DOS SUBINTERVALOS NA PARTICAO DO INTERVALO (A,B) DE
C                  INTEGRACAO.
C          RLIST   OS PRIMEIROS LAST ELEMENTOS CONTEM APROXIMACOES
C                  DAS INTEGRAIS NOS SUBINTERVALOS.
C          ELIST   OS PRIMEIROS LAST ELEMENTOS CONTEM OS MODULOS DAS
C                  ESTIMATIVAS DOS ERROS ABSOLUTOS NOS SUBINTERVALOS.
C          IORD    OS PRIMEIROS K ELEMENTOS SAO APONTADORES PARA AS
C                  ESTIMATIVAS DE ERROS NOS SUBINTERVALOS, DE MODO QUE
```

```
C                ELIST(IORD(1)), ... , ELIST(IORD(K)) FORMAM UMA
C                SEQUENCIA DECRESCENTE COM K = LAST SE LAST FOR
C                MENOR OU IGUAL LIMIT/2 + 2 E K = LIMIT + 1 - LAST
C                EM OUTROS CASOS.
C          NNLOG   INDICA OS NIVEIS DE SUBDIVISAO, OU SEJA, SE
C                NNLOG(I) = L SIGNIFICA QUE O SUBINTERVALO DE NUMERO I
C                TEM TAMANHO ABS(B-A)*2**(-L).
C          MOMCOM  INDICA QUE OS MOMENTOS DE CHEBYSHEV FORAM CALCULADOS
C                PARA INTERVALOS DE TAMANHO (ABS(B-A)*2**(-L),
C                L = 0, 1, ... , MOMCOM-1.
C          CHEBMO  CONTEM OS ELEMENTOS DE CHEBYSHEV.
C
C       SUBPROGRAMAS UTILIZADOS
C          DO FORTRAN - DABS, DMAX1
C          DA BIBLIOTECA - DQEXT, DQC25O, DQSORT
C          DO USUARIO - F
C
C       REFERENCIA
C          PIESSENS, DE DONCKER-KAPENGA, UBERHUBER, KAHANER, QUADPACK - A
C            SUBROUTINE PACKAGE FOR AUTOMATIC INTEGRATION, SPRINGER-
C            VERLAG, BERLIN HEIDELBERG N. YORK TOKYO, 1983
C
C                     UNIVERSIDADE FEDERAL DA PARAIBA
C                     DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C                     PROJETO BITAN
C                     MARIO T. HATTORI  ABR/90
C
345         INTEGER ID, IERRO, IROFF1, IROFF2, IROFF3, JUPBND, K,
           X         KSGN, KTMIN, LAST, MAXERR, NEV,
           X         NRES, NRMAX, NRMOM, NUMRL2
346         LOGICAL EXTRAP, NOEXT, EXTALL
347         DOUBLE PRECISION ABSEPS, AREA, AREA1, AREA12, AREA2,
           X                 A1, A2, B1, B2, CORREC, DEFAB1, DEFAB2,
           X                 DEFABS, DOMEGA, DRES, EPMACH, ERLARG,
           X                 ERLAST, ERRBND, ERRMAX, ERROR1, ERRO12,
           X                 ERROR2, ERRSUM, ERTEST, OFLOW, RESABS, RESEPS,
           X                 RES3LA(3), RLIST2(52), SMALL, UFLOW, WIDTH
348         DOUBLE PRECISION DABS, DMAX1, DMAQ, DQWGTO
349         EXTERNAL F
350         EPMACH = DMAQ (4)
351         UFLOW = DMAQ (1)
352         OFLOW = DMAQ (2)
C
C     VERIFICA VALIDADE DOS PARAMETROS
C
353         IER = 0
354         NEVAL = 0
355         LAST = 0
356         RESULT = 0.0E+00
357         ABSERR = 0.0E+00
358         ALIST(1) = A
359         BLIST(1) = B
360         RLIST(1) = 0.0E+00
361         ELIST(1) = 0.0E+00
362         IORD(1) = 0
363         NNLOG(1) = 0
C
364         IF ((INTEGR .NE. 1 .AND. INTEGR .NE. 2) .OR. (EPSABS .LT. 0.0E+00
           X   .AND. EPSREL .LT. 0.0E+00) .OR. ICALL .LT. 1 .OR. MAXP1 .LT. 1)
           X   IER = 6
```

```
      C
      C   ERRO NO(S) PARAMETRO(S), RETORNA.
      C
365         IF (IER .EQ. 6) GOTO 999
      C
      C   PRIMEIRA APROXIMACAO DA INTEGRAL
      C
366         DOMEGA = DABS (OMEGA)
367         NRMOM = 0
368         IF (ICALL .GT. 1) GOTO 5
369         MOMCOM = 0
370       5 CONTINUE
371         CALL DQC250 (F, A, B, DOMEGA, INTEGR, NRMOM, MAXP1, 0, RESULT,
           X             ABSERR, NEVAL, DEFABS, RESABS, MOMCOM, CHEBMO)
      C
      C   COMPARA ERRO COM TOLERANCIA DE ERRO
      C
372         DRES = DABS (RESULT)
373         ERRBND = DMAX1 (EPSABS, EPSREL * DRES)
374         RLIST(1) = RESULT
375         ELIST(1) = ABSERR
376         IORD(1) = 1
377         IF (ABSERR .LE. 1.0E+02 * EPMACH * DEFABS .AND.
           X    ABSERR .GT. ERRBND) IER = 2
378         IF (LIMIT .EQ. 1) IER = 1
      C
      C   RETORNA SE TOLERANCIA DE ERRO SATISFEITA
      C
379         IF (IER .NE. 0 .OR. ABSERR .LE. ERRBND) GOTO 200
      C
      C   ESTABELECE CONDICOES INICIAIS
      C
380         ERRMAX = ABSERR
381         MAXERR = 1
382         AREA = RESULT
383         ERRSUM = ABSERR
384         ABSERR = OFLOW
385         NRMAX = 1
386         EXTRAP = .FALSE.
387         NOEXT = .FALSE.
388         IERRO = 0
389         IROFF1 = 0
390         IROFF2 = 0
391         IROFF3 = 0
392         KTMIN = 0
393         SMALL = DABS (B - A) * 7.5E-01
394         NRES = 0
395         NUMRL2 = 0
396         EXTALL = .FALSE.
397         IF (5.0E-01 * DABS (B - A) * DOMEGA .GT. 2.0E+00) GOTO 10
398         NUMRL2 = 1
399         EXTALL = .TRUE.
400         RLIST2(1) = RESULT
401      10 CONTINUE
402         IF (2.5E-01 * DABS (B - A) * DOMEGA .LE. 2.0E+00) EXTALL = .TRUE.
403         KSGN = -1
404         IF (DRES .GE. (1.0E+00 - 5.0E+01 * EPMACH) * DEFABS) KSGN = 1
      C
      C    CICLO PRINCIPAL
      C
```

```
405         DO 140 LAST = 2,LIMIT
      C
      C   BISSECAO DO SUBINTERVALO COM NRMAX-ESIMA MAIOR ESTIMATIVA DE ERRO
      C
406           NRMOM = NNLOG(MAXERR) + 1
407           A1 = ALIST(MAXERR)
408           B1 = 5.0E-01 * (ALIST(MAXERR) + BLIST(MAXERR))
409           A2 = B1
410           B2 = BLIST(MAXERR)
411           ERLAST = ERRMAX
412           CALL DQC250 (F, A1, B1, DOMEGA, INTEGR, NRMOM, MAXP1, 0, AREA1,
          X                ERROR1, NEV, RESABS, DEFAB1, MOMCOM, CHEBMO)
413           NEVAL = NEVAL + NEV
414           CALL DQC250 (F, A2, B2, DOMEGA, INTEGR, NRMOM, MAXP1, 0, AREA2,
          X                ERROR2, NEV, RESABS, DEFAB2, MOMCOM, CHEBMO)
415           NEVAL = NEVAL + NEV
      C
      C   MELHORA AS APROXIMACOES DA INTEGRAL E DO ERRO E TESTA EXATIDAO
      C
416           AREA12 = AREA1 + AREA2
417           ERR012 = ERROR1 + ERROR2
418           ERRSUM = ERRSUM + ERR012 - ERRMAX
419           AREA = AREA + AREA12 - RLIST(MAXERR)
      C
420           IF (DEFAB1 .EQ. ERROR1 .OR. DEFAB2 .EQ. ERROR2) GOTO 25
      C
421           IF (DABS (RLIST(MAXERR) - AREA12) .GT. 1.0E-05 * DABS (AREA12)
          X       .OR. ERR012 .LT. 9.9E-01 * ERRMAX) GOTO 20
422           IF (EXTRAP) IROFF2 = IROFF2 + 1
423           IF (.NOT. EXTRAP) IROFF1 = IROFF1 + 1
424       20  CONTINUE
425           IF (LAST .GT. 10 .AND. ERR012 .GT. ERRMAX) IROFF3 = IROFF3 + 1
426       25  CONTINUE
427           RLIST(MAXERR) = AREA1
428           RLIST(LAST) = AREA2
429           NNLOG(MAXERR) = NRMOM
430           NNLOG(LAST) = NRMOM
431           ERRBND = DMAX1 (EPSABS, EPSREL * DABS (AREA))
      C
      C   TESTA ERRO DE ARREDONDAMENTO E EVENTUALMENTE SINALIZA ERRO
      C
432           IF (IROFF1 + IROFF2 .GE. 10 .OR. IROFF3 .GE. 20) IER = 2
433           IF (IROFF2 .GE. 5) IERRO = 3
      C
      C   SINALIZA ERRO EM CASO DE NUMERO DE SUBINTERVALOS SE TORNAR IGUAL
      C      A LIMIT
      C
434           IF (LAST .EQ. LIMIT) IER = 1
      C
      C   SINALIZA ERRO NO CASO DE MAU COMPORTAMENTO DO INTEGRANDO NUM
      C      PONTO INTERIOR AO INTERVALO DE INTEGRACAO
      C
435           IF (DMAX1 (DABS (A1), DABS (B2)) .LE. (1.0E+00 + 1.0E+03 *
          X      EPMACH) * (DABS (A2) + 1.0E+03 * UFLOW)) IER = 4
      C
      C   ADICIONA OS INTERVALOS RECEM CRIADOS A LISTA
      C
436           IF (ERROR2 .GT. ERROR1) GOTO 30
437           ALIST(LAST) = A2
438           BLIST(MAXERR) = B1
```

```
439           BLIST(LAST) = B2
440           ELIST(MAXERR) = ERROR1
441           ELIST(LAST) = ERROR2
442           GOTO 40
443       30  CONTINUE
444           ALIST(MAXERR) = A2
445           ALIST(LAST) = A1
446           BLIST(LAST) = B1
447           RLIST(MAXERR) = AREA2
448           RLIST(LAST) = AREA1
449           ELIST(MAXERR) = ERROR2
450           ELIST(LAST) = ERROR1
451       40  CONTINUE
      C
      C    CHAMA DQSORT PARA MANTER A ORDEM DECRESCENTE NA LISTA DE ESTIMATIVA
      C       DE ERRO E SELECIONA O SUBINTERVALO COM NRMAX-ESIMA MAIOR
      C       ESTIMA DE ERRO. (PARA A PROXIMA BISSECAO).
      C
452           CALL DQSORT (LIMIT, LAST, MAXERR, ERRMAX, ELIST, IORD, NRMAX)
      C
      C    DESVIO PARA FORA DO CICLO
      C
453           IF (ERRSUM .LE. ERRBND) GOTO 170
454           IF (IER .NE. 0) GOTO 150
455           IF (LAST .EQ. 2 .AND. EXTALL) GOTO 120
456           IF (NOEXT) GOTO 140
457           IF (.NOT. EXTALL) GOTO 50
458           ERLARG = ERLARG - ERLAST
459           IF (DABS (B1 - A1) .GT. SMALL) ERLARG = ERLARG + ERR012
460           IF (EXTRAP) GOTO 70
      C
      C    VERIFICA SE O INTERVALO A SER BISSECCIONADO A SEGUIR E O MENOR.
      C
461       50  CONTINUE
462           WIDTH = DABS (BLIST(MAXERR) - ALIST(MAXERR))
463           IF (WIDTH .GT. SMALL) GOTO 140
464           IF (EXTALL) GOTO 60
      C
      C    VERIFICA SE PODEMOS INICIAR COM EXTRAPOLACAO (PODEMOS FAZE-LO SE
      C       INTEGRAMOS SOBRE O PROXIMO INTERVALO USANDO A REGRA DE
      C       GAUSS-KONROD)
      C
465           SMALL = SMALL * 5.0E-01
466           IF (2.5E-01 * WIDTH * DOMEGA .GT. 2.0E+00) GOTO 140
467           EXTALL = .TRUE.
468           GOTO 130
469       60  CONTINUE
470           EXTRAP = .TRUE.
471           NRMAX = 2
472       70  CONTINUE
473           IF (IERRO .EQ. 3 .OR. ERLARG .LE. ERTEST) GOTO 90
      C
      C    O MENOR INTERVALO TEM O MAIOR ERRO. ANTES DA BISSECCAO DIMINUI
      C       A SOMA DOS ERROS, SOBRE INTERVALOS MAIORES (ERLARG) E EXECUTA
      C       EXTRAPOLACAO
      C
474           JUPBND = LAST
475           IF (LAST .GT. (LIMIT / 2 + 2)) JUPBND = LIMIT + 3 - LAST
476           ID = NRMAX
477           DO 80 K = ID,JUPBND
```

```
478            MAXERR = IORD(NRMAX)
479            ERRMAX = ELIST(MAXERR)
480            IF (DABS (BLIST(MAXERR) - ALIST(MAXERR)) .GT. SMALL) GOTO 140
481            NRMAX = NRMAX + 1
482      80    CONTINUE
       C
       C   EXECUTA EXTRAPOLACAO
       C
483      90    CONTINUE
484            NUMRL2 = NUMRL2 + 1
485            RLIST2(NUMRL2) = AREA
486            IF (NUMRL2 .LT. 3) GOTO 110
487            CALL DQEXT (NUMRL2, RLIST2, RESEPS, ABSEPS, RES3LA, NRES)
488            KTMIN = KTMIN + 1
489            IF (KTMIN .GT. 5 .AND. ABSERR .LT. 1.0E-03 * ERRSUM) IER = 5
490            IF (ABSEPS .GE. ABSERR) GOTO 100
491            KTMIN = 0
492            ABSERR = ABSEPS
493            RESULT = RESEPS
494            CORREC = ERLARG
495            ERTEST = DMAX1 (EPSABS, EPSREL * DABS (RESEPS))
       C
       C   DESVIO PARA FORA DO CICLO
       C
496            IF (ABSERR .LE. ERTEST) GOTO 150
       C
       C   PREPARA BISSECCAO DO MENOR INTERVALO
       C
497     100    CONTINUE
498            IF (NUMRL2 .EQ. 1) NOEXT = .TRUE.
499            IF (IER .EQ. 5) GOTO 150
500     110    CONTINUE
501            MAXERR = IORD(1)
502            ERRMAX = ELIST(MAXERR)
503            NRMAX = 1
504            EXTRAP = .FALSE.
505            SMALL = SMALL * 5.0E-01
506            ERLARG = ERRSUM
507            GOTO 140
508     120    CONTINUE
509            SMALL = SMALL * 5.0E-01
510            NUMRL2 = NUMRL2 + 1
511            RLIST(NUMRL2) = AREA
512     130    CONTINUE
513            ERTEST = ERRBND
514            ERLARG = ERRSUM
515     140 CONTINUE
       C
       C   DEFINE O RESULTADO FINAL
       C
516     150 CONTINUE
517         IF (ABSERR .EQ. OFLOW .OR. NRES .EQ. 0) GOTO 170
518         IF (IER + IERRO .EQ. 0) GOTO 165
519         IF (IERRO .EQ. 3) ABSERR = ABSERR + CORREC
520         IF (IER .EQ. 0) IER = 3
521         IF (RESULT .NE. 0.0E+00 .AND. AREA .NE. 0.0E+00) GOTO 160
522         IF (ABSERR .GT. ERRSUM) GOTO 170
523         IF (AREA .EQ. 0.0E+00) GOTO 190
524         GOTO 165
525     160 CONTINUE
```

```
526         IF (ABSERR / DABS (RESULT) .GT. ERRSUM / DABS (AREA)) GOTO 170
    C
    C   TESTA DIVERGENCIA
    C
527     165 CONTINUE
528         IF (KSGN .EQ. (-1) .AND. DMAX1 (DABS (RESULT), DABS (AREA)) .LE.
           X     DEFABS * 1.0E-02) GOTO 190
529         IF (1.0E-02 .GT. (RESULT / AREA) .OR. (RESULT / AREA) .GT. 1.0E+02
           X     .OR. ERRSUM .GE. DABS (AREA)) IER = 6
530         GOTO 190
    C
    C   COMPUTA SOMA GLOBAL
    C
531     170 CONTINUE
532         RESULT = 0.0E+00
533         DO 180 K = 1,LAST
534           RESULT = RESULT + RLIST(K)
535     180 CONTINUE
536         ABSERR = ERRSUM
537     190 CONTINUE
538         IF (IER .GT. 2) IER = IER - 1
539     200 CONTINUE
540         IF (INTEGR .EQ. 2 .AND. OMEGA .LT. 0.0E+00) RESULT = - RESULT
541     999 CONTINUE
542         RETURN
543         END
    C

544         SUBROUTINE DQEXT (N, EPSTAB, RESULT, ABSERR, RES3LA, NRES)
    C
545         INTEGER N, NRES
546         DOUBLE PRECISION EPSTAB(52), RESULT, ABSERR, RES3LA(3)
    C
    C       DETERMINA O LIMITE DE UMA DADA SEQUENCIA DE APROXIMACOES USANDO O
    C       ALGORITMO DE EPSILON DEVIDO A P. WYNN.
    C       UMA ESTIMATIVA DO ERRO ABSOLUTO TAMBEM E FORNECIDO.
    C       A TABELA CONDENSADA DE EPSILON E COMPUTADA. SOMENTE OS ELEMENTOS
    C       NECESSARIOS PARA A COMPUTACAO DA PROXIMA DIAGONAL SAO PRESERVADOS.
    C
    C       PARAMETROS
    C
    C       NA CHAMADA
    C          N        INTEGER
    C                   EPSTAB(N) CONTEM O NOVO ELEMENTO NA PRIMEIRA COLUNA DA
    C                   TABELA DE EPSILON.
    C          EPSTAB   DOUBLE PRECISION(52)
    C                   APENAS DECLARADO. USADO INTERNAMENTE.
    C          RESULT   DOUBLE PRECISION
    C                   APENAS DECLARADO
    C          ABSERR   DOUBLE PRECISION
    C                   APENAS DECLARADO
    C          RES3LA   DOUBLE PRECISION (3)
    C                   APENAS DECLARADO
    C          NRES     INTEGER
    C                   NUMERO DE CHAMADAS DA SUBROTINA. DEVE SER ZERO NA
    C                   PRIMEIRA CHAMADA.
    C
    C       NO RETORNO
    C          RESULT   APROXIMACAO DA INTEGRAL
    C          ABSERR   ESTIMATIVA DO ERRO ABSOLUTO CALCULADO A PARTIR DE
```

```
C                RESULT E 3 RESULTADOS ANTERIORES
C          RES3LA  CONTEM OS 3 ULTIMOS RESULTADOS.
C
C       SUBPROGRAMAS UTILIZADOS
C          DO FORTRAN - DABS, DMAX1
C          DA BIBLIOTECA - DMAQ
C
C       REFERENCIAS
C        1.PIESSENS, DE DONCKER-KAPENGA, UBERHUBER, KAHANER, QUADPACK - A
C            SUBROUTINE PACKAGE FOR AUTOMATIC INTEGRATION, SPRINGER-
C            VERLAG, BERLIN HEIDELBERG N. YORK TOKYO, 1983
C        2.WYNN, ON A DEVICE FOR COMPUTING THE E (S ) TRANSFORMATION,
C                                               M  M
C            MTAC 10, PP. 91-96.
C
C                UNIVERSIDADE FEDERAL DA PARAIBA
C                DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C                PROJETO BITAN
C                MARIO T. HATTORI  ABR/90
C
      INTEGER I, IB, IB2, IE, INDX, K1, K2, K3, LIMEXP, NEWELM, NUM
      DOUBLE PRECISION DELTA1, DELTA2, DELTA3, EPMACH, EPSINF,
     X                 ERROR, ERR1, ERR2, ERR3, E0, E1, E1ABS,
     X                 E2, E3, OFLOW, ONE, P5, RES, SS, TENM4,
     X                 TOL1, TOL2, TOL3, UFLOW
      DOUBLE PRECISION DABS, DMAQ, DMAX1
      DATA TENM4 /1.D-4/, P5 /.5D0/, ONE /1.D0/
C
      UFLOW = DMAQ (1)
      OFLOW = DMAQ (2)
      EPMACH = DMAQ (4)
C
      NRES = NRES + 1
      ABSERR = OFLOW
      RESULT = EPSTAB(N)
      IF (N .LT. 3) THEN
         ABSERR = DMAX1 (ABSERR , P5 * EPMACH * DABS (RESULT))
         RETURN
      ENDIF
      LIMEXP = 50
      EPSTAB(N+2) = EPSTAB(N)
      NEWELM = (N - 1) / 2
      EPSTAB(N) = OFLOW
      NUM = N
      K1 = N
      DO 40 I = 1,NEWELM
         K2 = K1 - 1
         K3 = K1 - 2
         RES = EPSTAB(K1+2)
         E0 = EPSTAB(K3)
         E1 = EPSTAB(K2)
         E2 = RES
         E1ABS = DABS (E1)
         DELTA2 = E2 - E1
         ERR2 = DABS (DELTA2)
         TOL2 = DMAX1 (DABS (E2), E1ABS) * EPMACH
         DELTA3 = E1 - E0
         ERR3 = DABS (DELTA3)
         TOL3 = DMAX1 (E1ABS, DABS (E0)) * EPMACH
         IF (ERR2 .LE. TOL2 .AND. ERR3 .LE. TOL3) THEN
```

```
C
C           SE E0, E1 E E2 SAO IGUAIS DENTRO DA PRECISAO DA
C           DA MAQUINA, ASSUME-SE QUE HOUVE CONVERGENCIA.
C           RESULT = E2
C           ABSERR = ABS (E1 - E0) + ABS (E2 - E1)
C
582            RESULT = RES
583            ABSERR = ERR2 + ERR3
584            ABSERR = DMAX1 (ABSERR, P5*EPMACH*DABS (RESULT))
585            RETURN
586         ENDIF
587         E3 = EPSTAB(K1)
588         EPSTAB(K1) = E1
589         DELTA1 = E1 - E3
590         ERR1 = DABS (DELTA1)
591         TOL1 = DMAX1 (E1ABS, DABS (E3)) * EPMACH
C
592         IF (ERR1 .GT. TOL1 .AND. ERR2 .GT. TOL2 .AND.
      X          ERR3 .GT. TOL3) THEN
593            SS = ONE / DELTA1 + ONE / DELTA2 - ONE / DELTA3
594            EPSINF = DABS (SS * E1)
C
C           SE DOIS ELEMENTOS TEM VALORES MUITO PROXIMOS, OMITE
C           A PARTE DA TABELA AJUSTANDO O VALOR DE N
C
595            IF (EPSINF .GT. TENM4) GOTO 30
596         ENDIF
597            N = I + I - 1
598            GOTO 50
599   30    CONTINUE
C
C           VERIFICA SE HA COMPORTAMENTO IRREGULAR NA TABLE E
C           EVENTUALMENTE OMITE PARTE DA TABELA AJUSTANDO O
C           VALOR DE N
C
600         IF (EPSINF .LE. TENM4 ) THEN
601            N = I + I -1
602            GOTO 50
603         ENDIF
C
C           COMPUTA UM NOVO ELEMENTO E EVENTUALMENTE AJUSTA O VALOR
C           DE RESULT.
C
604         RES = E1 + ONE / SS
605         EPSTAB(K1) = RES
606         K1 = K1 - 2
607         ERROR = ERR2 + DABS (RES -E2) + ERR3
608         IF (ERROR .LE. ABSERR ) THEN
609            ABSERR = ERROR
610            RESULT = RES
611         ENDIF
612 40    CONTINUE
C
C     DESLOCA TABELA
C
613 50    CONTINUE
614       IF (N .EQ. LIMEXP) N = 2 * (LIMEXP / 2) - 1
615       IB = 1
616       IF ((NUM / 2 ) * 2 .EQ. NUM) IB = 2
617       IE = NEWELM + 1
```

```
      DO 60 I = 1,IE
         IB2 = IB + 2
         EPSTAB(IB) = EPSTAB(IB2)
         IB = IB2
60    CONTINUE
      IF (NUM .NE. N) THEN
         INDX = NUM - N + 1
         DO 70 I = 1,N
            EPSTAB(I) = EPSTAB(INDX)
            INDX = INDX + 1
70       CONTINUE
      ENDIF
      IF (NRES .LT. 4) THEN
         RES3LA(NRES) = RESULT
         ABSERR = OFLOW
         ABSERR = DMAX1 (ABSERR, P5 * EPMACH * DABS (RESULT))
         RETURN
      ENDIF
C
C     COMPUTA ESTIMATIVA DE ERRO
C
      ABSERR = DABS (RESULT - RES3LA(3)) + DABS (RESULT - RES3LA(2))
     X         + DABS (RESULT - RES3LA(1))
      RES3LA(1) = RES3LA(2)
      RES3LA(2) = RES3LA(3)
      RES3LA(3) = RESULT
      ABSERR = DMAX1 (ABSERR , P5 * EPMACH * DABS (RESULT))
      RETURN
      END

      SUBROUTINE DQK15I(F, BOUN, INF, A, B, RESULT, ABSERR,
     X                  RESABS, RESASC)
C
      INTEGER INF
      DOUBLE PRECISION F, BOUN, A, B, RESULT, ABSERR, RESABS, RESASC
C
C     O INTERVALO ORIGINAL DE INTEGRACAO (INFINITO) E MAPEADO NO
C     INTERVALO (0,1) E (A,B) E UMA PARTE DE (0,1)
C     COMPUTA I = INTEGRAL DO INTEGRANDO TRANSFORMADO SOBRE (A,B)
C              J = INTEGRAL DE ABS(DO INTEGRANDO TRANSFORMADO) SOBRE
C                  (A,B)
C
C     PARAMETROS
C     NA CHAMADA
C        F       DOUBLE PRECISION FUNCTION
C                SUBPROGRAMA FUNCAO QUE DEFINE A FUNCAO INTEGRANDO F(X).
C                O NOME REAL DE F PRECISA SER DECLARADO NUM EXTERNAL NO
C                PROGRAMA ATIVADOR
C        BOUN    DOUBLE PRECISION
C                LIMITE FINITO DO INTERVALO ORIGINAL DE INTEGRACAO
C                (SERA ZERO SE INF = +2)
C        INF     INTEGER
C                INF = -1, O INTERVALO ORIGINAL DE INTEGRACAO E
C                          (-INFINITO,BOUN)
C                INF = +1, O INTERVALO ORIGINAL DE INTEGRACAO E
C                          (BOUN,INFINITO)
C                INF = +2, O INTERVALO ORIGINAL DE INTEGRACAO E
C                          (-INFINITO,+INFINITO)
C        A       DOUBLE PRECISION
C                O LIMITE INFERIOR DO INTERVALO DE INTEGRACAO
```

```
C                SOBRE O SUBINTERVALO (0,1)
C          B     DOUBLE PRECISION
C                O LIMITE SUPERIOR DO INTERVALO DE INTEGRACAO
C                SOBRE O SUBINTERVALO (0,1)
C          RESULT DOUBLE PRECISION
C                APENAS DECLARADO
C          ABSERR DOUBLE PRECISION
C                APENAS DECLARADO
C          RESABS DOUBLE PRECISION
C                APENAS DECLARADO
C          RESASC DOUBLE PRECISION
C                APENAS DECLARADO
C       NO RETORNO
C          RESULT APROXIMACAO DA INTEGRAL I. RESULT E COMPUTADO USANDO A
C                REGRA DE 15 PONTOS DE KONROD, OBTIDA PELA ADICAO OTIMA
C                DE ABCISSAS A REGRA DE GAUSS DE 7 PONTOS.
C          ABSERR ESTIMATIVA DO ERRO ABSOLUTO, QUE NAO DEVE EXCEDER
C                ABS (I - RESULT)
C          RESABS APROXIMACAO DA INTEGRAL J
C          RESASC APROXIMACAO DA INTEGRAL
C                ABS (INTEGRANDO TRANSFORMADO - I/(B-A))
C                SOBRE (A,B)
C
C       SUBPROGRAMAS UTILIZADOS
C          DO FORTRAN - DABS, DBLE, DMAX1, DMIN1, FLOAT, MIN0
C          DA BIBLIOTECA - DMAQ
C          DO USUARIO - F
C
C       REFERENCIA
C          PIESSENS, DE DONCKER-KAPENGA, UBERHUBER, KAHANER, QUADPACK - A
C            SUBROUTINE PACKAGE FOR AUTOMATIC INTEGRATION, SPRINGER-VERLAG
C            BERLIN HEIDELBERG N. YORK TOKYO, 1983
C
C                UNIVERSIDADE FEDERAL DA PARAIBA
C                DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C                PROJETO BITAN
C                MARIO T. HATTORI  ABR/90
C
646        INTEGER J, MIN0
647        REAL FLOAT
648        DOUBLE PRECISION ABSC, ABSC1, ABSC2, CENTR, DINF, EPMACH,
          X                 FC, FSUM, FVAL1, FVAL2,
          X                 FV1(7), FV2(7), HLGTH, OFLOW, ONE, ONEP5, RESG,
          X                 RESK, RESKH, TABSC1, TABSC2,
          X                 TWO, UFLOW, WG(8), WGK(8), XGK(8),
          X                 ZERO
649        DOUBLE PRECISION DABS, DBLE, DMAQ, DMAX1, DMIN1
C
C       AS ABCISSAS E PESOS SAO DADOS PARA O INTERVALO (-1,1). POR CAUSA
C       DA SIMETRIA SOMENTE AS ABCISSAS POSITIVAS E OS CORRESPONDENTES
C       PESOS SAO DADOS.
C
C       XGK    ABCISSAS DA REGRA DE 15 PONTOS DE KONROD
C              XGK(2), XGK(4), ... ABCISSAS DA REGRA DE 7 PONTOS DE GAUSS
C
C              XGK(1), XGK(3), ... ABCISSAS QUE FORAM ADICIONADAS
C              OTIMAMENTE A REGRA DE 7 PONTOS DE GAUSS.
C       WGK    PESOS DA REGRA DE 15 PONTOS DE KONROD
C       WG     PESOS DA REGRA DE 7 PONTOS DE GAUSS
C
```

```
C                  ESTIMATIVAS DE ERRO TAL QUE ELIST(IORD(1)), ... ,
C                  ELIST(IORD(K)) FORMA UMA SEQUENCIA DECRESCENTE COM
C                  K = LAST SE LAST .LE. (LIMIT / 2 + 2) E
C                  K = LIMIT + 1 - LAST, CASO CONTRARIO.
C
C     SUBPROGRAMAS UTILIZADOS
C        DO FORTRAN - DABS, DMAX1, DMIN1
C        DA BIBLIOTECA - DMAQ
C        DO USUARIO - F
C
C     REFERENCIA
C        PIESSENS, DE DONCKER-KAPENGA, UBERHUBER, KAHANER, QUADPACK - A
C          SUBROUTINE PACKAGE FOR AUTOMATIC INTEGRATION, SPRINGER-
C          VERLAG, BERLIN HEIDELBERG N. YORK TOKYO, 1983
C
C                  UNIVERSIDADE FEDERAL DA PARAIBA
C                  DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C                  PROJETO BITAN
C                  MARIO T. HATTORI  ABR/90
C
      INTEGER I, IBEG, IDO, ISUCC, J, JBND, JUPBN, K
      DOUBLE PRECISION ERRMAX, ERRMIN
C
C     VERIFICA SE A LISTA CONTEM MAIS DE DUAS ESTIMATIVAS DE ERRO
C
      IF (LAST .LE. 2) THEN
         IORD(1) = 1
         IORD(2) = 2
         MAXERR = IORD(NRMAX)
         ERMAX = ELIST(MAXERR)
         RETURN
      ENDIF
C
C     ESTA PARTE DA ROTINA SO SERA EXECUTADA SE, DEVIDO A UM INTEGRANDO
C     DIFICIL, A SUBDIVISAO AUMENTOU A ESTIMATIVA DE ERRO. NO CASO
C     NORMAL O PROCEDIMENTO DE INSERCAO DEVE COMECAR DEPOIS DA
C     NRMAX-ESIMA MAIOR ESTIMATIVA DE ERRO.
C
      ERRMAX = ELIST(MAXERR)
      IF (NRMAX .NE. 1) THEN
         IDO = NRMAX - 1
         DO 20 I = 1,IDO
            ISUCC = IORD(NRMAX - 1)
            IF (ERRMAX .LE. ELIST(ISUCC)) GOTO 30
            IORD(NRMAX) = ISUCC
            NRMAX = NRMAX - 1
   20    CONTINUE
      ENDIF
   30 CONTINUE
C
C     COMPUTA O NUMERO DE ELEMENTOS NA LISTA PARA SER MANTIDO EM
C     ORDEM DECRESCENTE. ESSE NUMERO DEPENDE DO NUMERO DE SUBDIVISOES
C     AINDA PERMITIDO.
C
      JUPBN = LAST
      IF (LAST .GT. (LIMIT / 2 + 2)) JUPBN = LIMIT + 3 - LAST
      ERRMIN = ELIST(LAST)
C
C     INSERE ERRMAX PERCORRENDO A LISTA DO FIM PARA O INICIO,
C     COMECANDO AS COMPARACOES A PARTIR DO ELEMENTO
```

```
C     ELIST(IORD(NRMAX+1))
C
      JBND = JUPBN - 1
      IBEG = NRMAX + 1
      IF (IBEG .LE. JBND) THEN
         DO 40 I = IBEG,JBND
            ISUCC = IORD(I)
            IF (ERRMAX .GE. ELIST(ISUCC)) GOTO 60
            IORD(I-1) = ISUCC
   40    CONTINUE
      ENDIF
      IORD(JBND) = MAXERR
      IORD(JUPBN) = LAST
      MAXERR = IORD(NRMAX)
      ERMAX = ELIST(MAXERR)
      RETURN
C
   60 CONTINUE
C
C     INSERE ERRMIN PERCORRENDO A LISTA DO INICIO
C
      IORD(I-1) = MAXERR
      K = JBND
      DO 70 J = I,JBND
         ISUCC = IORD(K)
         IF (ERRMIN .LT. ELIST(ISUCC)) GOTO 80
         IORD(K+1) = ISUCC
         K = K - 1
   70 CONTINUE
      IORD(I) = LAST
      MAXERR = IORD(NRMAX)
      ERMAX = ELIST(MAXERR)
      RETURN
C
   80 CONTINUE
      IORD(K+1) = LAST
      MAXERR = IORD(NRMAX)
      ERMAX = ELIST(MAXERR)
      RETURN
      END

      SUBROUTINE DQC250 (F, A, B, OMEGA, INTEGR, NRMOM, MAXP1, KSAVE,
     X                   RESULT, ABSERR, NEVAL, RESABS, RESASC,
     X                   MOMCOM, CHEBMO)
C
      INTEGER INTEGR, NRMOM, MAXP1, KSAVE, NEVAL, MOMCOM
      DOUBLE PRECISION F, A, B, OMEGA, RESULT, ABSERR, RESABS, RESASC,
     X                 CHEBMO(MAXP1,25)
C
C     CALCULA A INTEGRAL
C     I = INTEGRAL DE F(X)*W(X) SOBRE (A,B)
C     EM QUE W(X) = COS(OMEGA*X) OU W(X) = SIN(OMEGA*X), E
C     COMPUTA J = INTEGRAL DE ABS(F) SOBRE (A,B).
C     PARA VALORES PEQUENOS DE OMEGA OU INTERVALO (A,B) PEQUENO A
C     REGRA DE GAUSS-KONROD DE 15 PONTOS E USADA. EM TODOS OS OUTROS
C     CASOS UM METODO DE CLENSHAW-CURTIS GENERALIZADO E USADO, I. E.
C     UMA EXPANSAO TRUNCADA DE CHEBYSHEV DA FUNCAO F E COMPUTADA EM
C     (A,B), DE MODO QUE O INTEGRANDO PODE SER ESCRITO COMO UMA SOMA
C     DE TERMOS DA FORMA W(X)T(K,X), EM QUE T(K,X) E O POLINOMIO DE
C     CHEBYSHEV DE GRAU K. OS MOMENTOS DE CHEBYSHEV SAO COMPUTADOS COM
```

```
C     A UTILIZACAO DE UMA RELACAO DE RECORRENCIA LINEAR.
C
C     PARAMETROS
C     NA CHAMADA
C        F        DOUBLE PRECISION FUNCTION
C                 SUBPROGRAMA FUNCAO QUE DEFINE A FUNCAO INTEGRANDO. O
C                 NOME REAL DESSA FUNCAO DEVE SER DECLARADO NUM EXTERNAL
C                 NO PROGRAMA ATIVADOR.
C        A        DOUBLE PRECISION
C                 LIMITE INFERIOR DE INTEGRACAO
C        B        DOUBLE PRECISION
C                 LIMITE SUPERIOR DE INTEGRACAO
C        OMEGA    DOUBLE PRECISION
C                 PARAMETRO NA FUNCAO PESO
C        INTEGR   INTEGER
C                 INDICA QUAL A FUNCAO PESO DESEJADA.
C                   INTEGR = 1  W(X) = COS(OMEGA*X)
C                   INTEGR = 2  W(X) = SIN(OMEGA*X)
C        NRMOM    INTEGER
C                 DEVE TER VALOR ZERO NA PRIMEIRA CHAMADA.
C        MAPXP1   INTEGER
C                 UM LIMITE SUPERIOR NO NUMERO DE MOMENTOS DE CHEBYSHEV
C                 QUE PODE SER ARMAZENADO, I. E., PARA INTERVALOS DE
C                 COMPRIMENTOS ABS (BB - AA)*2**(-L), L = 0, 1, 2, ... ,
C                 MAXP1 - 2.
C        KSAVE    INTEGER
C                 FUNCIONA COMO CHAVE. TERA VALOR INICIAL 0.
C        RESULT   DOUBLE PRECISION
C                 APENAS DECLARADO
C        ABSERR   DOUBLE PRECISION
C                 APENAS DECLARADO
C        NEVAL    INTEGER
C                 APENAS DECLARADO
C        RESABS   DOUBLE PRECISION
C                 APENAS DECLARADO
C        RESASC   DOUBLE PRECISION
C                 APENAS DECLARADO
C        MOMCOM   INTEGER
C                 DEVE TER VALOR INICIAL ZERO
C        CHEBMO   DOUBLE PRECISION (MAXP1,25)
C                 APENAS DECLARADO
C
C     NO RETORNO
C        RESULT   APROXIMACAO DA INTEGRAL I
C        ABSERR   ESTIMATIVA DO MODULO DO ERRO ABSOLUTO E NAO DEVE
C                 EXCEDER ABS(I - RESULT)
C        NEVAL    NUMERO DE AVALIACOES DO INTEGRANDO
C        RESABS   APROXIMACAO DA INTEGRAL J
C        RESASC   APROXIMACAO DA INTEGRAL DE ABS(F - I/(B-A))
C        MOMCOM   CONTAGEM DO NUMERO DE INTERVALOS PARA OS QUAIS OS
C                 MOMENTOS DE CHEBYSHEV FORAM CALCULADOS.
C        CHEBMO   OS MOMENTOS DE CHEBYSHEV MODIFICADOS PARA OS
C                 MOMCOM INTERVALOS.
C
C     SUBPROGRAMAS UTILIZADOS
C        DO FORTRAN - DABS, DBLE, DCOS, DLOG, DMAX1, DMIN1, DSIN,
C                     FLOAT, IDINT, MIN0
C        DA BIBLIOTECA - DMAQ, DQCHEB, DQK15W, DQWGTO
C        DO USUARIO - F
C
```

```
C      SUBPROGRAMAS UTILIZADOS
C         DO FORTRAN - DABS, DMAX1, DMIN1
C         DA BIBLIOTECA - DMAQ
C         DO USUARIO - F
C
C      REFERENCIA
C         PIESSENS, DE DONCKER-KAPENGA, UBERHUBER, KAHANER, QUADPACK - A
C           SUBROUTINE PACKAGE FOR AUTOMATIC INTEGRATION, SPRINGER-
C           VERLAG, BERLIN HEIDELBERG N. YORK TOKYO, 1983
C
C                  UNIVERSIDADE FEDERAL DA PARAIBA
C                  DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C                  PROJETO BITAN
C                  MARIO T. HATTORI  ABR/90
C
      INTEGER I, ISYM, J, K, M, NOEQU, NOEQ1
      INTEGER IDINT, MIN0
      REAL FLOAT
      DOUBLE PRECISION AC, AN, AN2, AS, ASAP, ASS, CENTR, CHEB12(13),
     X                 CHEB24(25), CONC, CONS, COSPAR, D(43),
     X                 D1(43), D2(43), D3(43), EPMACH, ESTC,
     X                 ESTS, FVAL(25), HLGTH, ONE, OFLOW,
     X                 PARINT, PAR2, PAR22, P2, P3, P4, RESC12,
     X                 RESC24, RESS12, RESS24, SINPAR, TWO,
     X                 UFLOW, V(43), X(11), ZERO
      DOUBLE PRECISION DABS, DBLE, DCOS, DLOG, DMAQ, DMAX1, DMIN1,
     X                 DSIN, DQWGTO
C
      EXTERNAL F, DQWGTO
C
C     O VETOR X CONTEM OS VALORES DE COS(K*PI/24), K = 1, ... , 11
C     PARA SEREM USADOS NA EXPANSAO DE CHEBYSHEV DE F
C
      DATA X /
     X           9.914448613738104D-01,  9.659258262890683D-01,
     X           9.238795325112868D-01,  8.660254037844386D-01,
     X           7.933533402912352D-01,  7.071067811865475D-01,
     X           6.087614290087206D-01,  5.000000000000000D-01,
     X           3.826834323650898D-01,  2.588190451025208D-01,
     X           1.305261922200516D-01 /
      DATA ZERO /0.D0/, ONE /1.D0/, TWO /2.D0/
C
      UFLOW = DMAQ (1)
      OFLOW = DMAQ (2)
      EPMACH = DMAQ (4)
C
      CENTR = (B + A) / TWO
      HLGTH = (B - A) / TWO
      PARINT = OMEGA * HLGTH
C
C     COMPUTA A INTEGRAL USANDO A REGRA DE GAUSS-KONROD DE 15 PONTOS SE
C     O VALOR DO PARAMETRO NO INTEGRANDO FOR PEQUENO OU SE O INTERVALO
C     DE INTEGRACAO FOR MENOR QUE (BB - AA)/2**(MAXP1-2), ONDE (AA,BB)
C     E O INTERVALO ORIGINAL DE INTEGRACAO.
C
      IF (DABS (PARINT) .LE. TWO) THEN
         CALL DQK15W (F, DQWGTO, OMEGA, P2, P3, P4, INTEGR, A, B,
     X                RESULT, ABSERR, RESABS, RESASC)
         NEVAL = 15
         RETURN
```

```
779         ENDIF
            C
            C     COMPUTA A INTEGRAL USANDO O METODO DE CLENSHAW-CURTIS
            C     GENERALIZADO.
            C
780         CONC = HLGTH * DCOS (CENTR * OMEGA)
781         CONS = HLGTH * DSIN (CENTR * OMEGA)
782         RESASC = OFLOW
783         NEVAL = 25
            C
            C     VERIFICA SE OS MOMENTOS DE CHEBYSHEV PARA ESTE INTERVALO JA
            C     FORAM CALCULADOS
            C
784         IF (NRMOM .GE. MOMCOM .AND. KSAVE .NE. 1) THEN
            C
            C        COMPUTA UMA NOVA SERIE DE MOMENTOS DE CHEBYSHEV
            C
785            M = MOMCOM + 1
786            PAR2 = PARINT * PARINT
787            PAR22 = PAR2 + TWO
788            SINPAR = DSIN (PARINT)
789            COSPAR = DCOS (PARINT)
            C
            C        COMPUTA OS MOMENTOS DE CHEBYSHEV COM RELACAO AO COSSENO
            C
790            V(1) = TWO * SINPAR / PARINT
791            V(2) = (8.D0 * COSPAR + (PAR2 + PAR2 -8.D0) * SINPAR /
           X             PARINT) / PAR2
792            V(3) = (32.D0 * (PAR2 -12.D0) * COSPAR + (TWO *
           X              ((PAR2 - 80.D0) * PAR2 + 192.D0) * SINPAR) /
           X               PARINT) /(PAR2 * PAR2)
793            AC = 8.D0 * COSPAR
794            AS = 24.D0 * PARINT * SINPAR
795            IF (DABS (PARINT) .LE. 24.D0) THEN
            C
            C           CALCULA OS MOMENTOS DE CHEBYSHEV COMO SOLUCOES DE UM
            C           PROBLEMA DE CONTORNO COM UM VALOR DE CONTORNO (V(3)) E
            C           O OUTRO COMPUTADO USANDO UMA FORMULA ASSINTOTICA.
            C
796               NOEQU = MIN0 (40, 13+IDINT (-DLOG (EPMACH))/3)
797               NOEQ1 = NOEQU - 1
798               AN = 6.D0
799               DO 20 K = 1,NOEQ1
800                  AN2 = AN * AN
801                  D(K) = -TWO * (AN2 - 4.D0) * (PAR22 - AN2 - AN2)
802                  D2(K) = (AN - ONE) * (AN - TWO) * PAR2
803                  D1(K) = (AN + 3.D0) * (AN + 4.D0) * PAR2
804                  V(K+3) = AS - (AN2 - 4.D0) * AC
805                  AN = AN + TWO
806         20       CONTINUE
807               AN2 = AN * AN
808               D(NOEQU) = -TWO * (AN2 - 4.D0) *(PAR22 - AN2 - AN2)
809               V(NOEQU+3) = AS - (AN2 - 4.D0) * AC
810               V(4) = V(4) - 56.D0 * PAR2 * V(3)
811               ASS = PARINT * SINPAR
812               ASAP = (((((210.D0 * PAR2 - ONE) * COSPAR - (105.D0 * PAR2
           X                    -63.D0) * ASS) / AN2 - (ONE - 15.D0 * PAR2) *
           X                   COSPAR + 15.D0 * ASS) / AN2 - COSPAR + 3.D0 *
           X                   ASS) / AN2 - COSPAR) / AN2
813               V(NOEQU+3) = V(NOEQU+3) - TWO * ASAP * PAR2 * (AN - ONE) *
```

```
     X                    (AN - TWO)
C
C          RESOLVE O SISTEMA TRIDIAGONAL PELA ELIMINACAO DE GAUSS
C          COM PIVOTAMENTO PARCIAL
C
           DO 30 I = 1,NOEQU
              D3(I) = ZERO
30         CONTINUE
           D2(NOEQU) = ZERO
           DO 50 I = 1,NOEQ1
              IF (DABS (D1(I)) .GT. DABS (D(I))) THEN
                 AN = D1(I)
                 D1(I) = D(I)
                 D(I) = AN
                 AN = D2(I)
                 D2(I) = D(I+1)
                 D(I+1) = AN
                 D3(I) = D2(I+1)
                 D2(I+1) = ZERO
                 AN = V(I+4)
                 V(I+4) = V(I+3)
                 V(I+3) = AN
              ENDIF
              D(I+1) = D(I+1) -D2(I) * D1(I) / D(I)
              D2(I+1) = D2(I+1) - D3(I) * D1(I) / D(I)
              V(I+4) = V(I+4) - V(I+3) * D1(I) / D(I)
50         CONTINUE
           V(NOEQU+3) = V(NOEQU+3) / D(NOEQU)
           V(NOEQU+2) = (V(NOEQU+2) -D2(NOEQ1) * V(NOEQU+3)) / D(NOEQ1)
           DO 60 I = 2,NOEQ1
              K = NOEQU - I
              V(K+3) = (V(K+3) -D3(K) * V(K+5) -D2(K) * V(K+4)) / D(K)
60         CONTINUE
C
        ELSE
           AN = 4.D0
           DO 80 I = 4,13
              AN2 = AN * AN
              V(I) = ((AN2 - 4.D0) * (TWO * (PAR22 - AN2 - AN2 ) *
     X                  V(I-1) - AC) + AS - PAR2 * (AN + ONE) *
     X               (AN + TWO) * V(I-2)) / (PAR2 * (AN -ONE) *
     X               (AN - TWO))
              AN = AN + TWO
80         CONTINUE
        ENDIF
        DO 100 J = 1,13
           CHEBMO(M,2*J-1) = V(J)
100     CONTINUE
C
C          COMPUTA MOMENTOS DE CHEBYSHEV COM RESPEITO AOS SENOS
C
        V(1) = TWO * (SINPAR - PARINT * COSPAR) / PAR2
        V(2) = (18.D0 - 48.D0 / PAR2) * SINPAR / PAR2+
     X            (-TWO + 48.D0 / PAR2) * COSPAR / PARINT
        AC = -24.D0 * PARINT * COSPAR
        AS = -8.D0 * SINPAR
        CHEBMO(M,2) = V(1)
        CHEBMO(M,4) = V(2)
        IF (DABS (PARINT) .LE. 24.D0) THEN
           DO 110 K = 3,12
```

```
861                  AN = DBLE (FLOAT (K))
862                  CHEBMO(M,2*K) = -SINPAR / (AN * (TWO * AN -TWO))
          X                            -.25D0 * PARINT * (V(K+1)/AN
          X                            - V(K) / (AN - ONE))
863      110         CONTINUE
864              ELSE
      C
      C              COMPUTA MOMENTOS DE CHEBYSHEV PELA RECURSAO PROGRESSIVA
      C
865                  AN = 3.D0
866                  DO 130 I = 3,12
867                     AN2 = AN * AN
868                     V(I) = ((AN2 - 4.D0) * (TWO *(PAR22 - AN2 - AN2) *
          X                       V(I-1) + AS) + AC - PAR2 * (AN + ONE) *
          X                       (AN + TWO) * V(I - 2)) /
          X                       (PAR2 * (AN - ONE) * (AN - TWO))
869                     AN = AN + TWO
870                     CHEBMO(M,2*I) = V(I)
871      130         CONTINUE
872              ENDIF
873          ENDIF
874          IF (NRMOM .LT. MOMCOM) M = NRMOM + 1
875          IF (MOMCOM .LT. (MAXP1 - 1) .AND. NRMOM .GE. MOMCOM)
          X         MOMCOM = MOMCOM + 1
      C
      C      CALCULA OS COEFICIENTES DAS EXPANSOES DE CHEBYSHEV DE GRAUS 12 E
      C      24 DA FUNCAO F
      C
876          FVAL(1) = F (CENTR + HLGTH) / TWO
877          FVAL(13) = F (CENTR)
878          FVAL(25) = F (CENTR - HLGTH) / TWO
879          DO 150 I = 2,12
880             ISYM = 26 - I
881             FVAL(I) = F (CENTR + HLGTH * X(I-1))
882             FVAL(ISYM) = F (CENTR - HLGTH * X(I-1))
883      150 CONTINUE
      C
884          CALL DQCHEB (X, FVAL, CHEB12, CHEB24)
      C
      C      COMPUTA A INTEGRAL E ESTIMATIVA DE ERRO
      C
885          RESC12 = CHEB12(13) * CHEBMO(M,13)
886          RESS12 = ZERO
887          ESTC = DABS (CHEB24(25) * CHEBMO(M,25)) +
          X         DABS (CHEB12(13) - CHEB24(13) * CHEBMO(M,13))
888          ESTS = ZERO
889          K = 11
      C
890          DO 160 J = 1,6
891             RESC12 = RESC12 + CHEB12(K) * CHEBMO(M,K)
892             RESS12 = RESS12 + CHEB12(K+1) * CHEBMO(M,K+1)
893             ESTC = ESTC + DABS ((CHEB12(K) - CHEB24(K)) * CHEBMO(M,K))
894             ESTS = ESTS + DABS ((CHEB12(K+1) - CHEB24(K+1)) *
          X                        CHEBMO(M,K+1))
895             K = K - 2
896      160 CONTINUE
      C
897          RESC24 = CHEB24(25) * CHEBMO(M,25)
898          RESS24 = ZERO
899          RESABS = DABS (CHEB24(25))
```

```
      K = 23
C
      DO 170 J = 1,12
         RESC24 = RESC24 + CHEB24(K) * CHEBMO(M,K)
         RESS24 = RESS24 + CHEB24(K+1) * CHEBMO(M,K+1)
         RESABS = RESABS + DABS (CHEB24(K)) + DABS (CHEB24(K+1))
         IF (J .LE. 5) THEN
            ESTC = ESTC + DABS (CHEB24(K) * CHEBMO(M,K))
            ESTS = ESTS + DABS (CHEB24(K+1) * CHEBMO(M,K+1))
         ENDIF
         K = K - 2
  170 CONTINUE
C
      RESABS = RESABS * DABS (HLGTH)
      IF (INTEGR .NE. 2) THEN
         RESULT = CONC * RESC24 - CONS * RESS24
         ABSERR = DABS (CONC * ESTC)+ DABS (CONS * ESTS)
      ELSE
         RESULT = CONC * RESS24 + CONS * RESC24
         ABSERR = DABS (CONC * ESTS) + DABS (CONS * ESTC)
      ENDIF
C
      RETURN
      END

      DOUBLE PRECISION FUNCTION DQWGTO (X, OMEGA, P2, P3, P4, INTEGR)
C
      INTEGER INTEGR
      DOUBLE PRECISION X, OMEGA, P2, P3, P4
C
      DOUBLE PRECISION OMX
      DOUBLE PRECISION DCOS, DSIN
C
      OMX = OMEGA * X
      IF (INTEGR .EQ. 1) THEN
         DQWGTO = DCOS (OMX)
      ELSE
         DQWGTO = DSIN (OMX)
      ENDIF
C
      RETURN
      END
C


      SUBROUTINE DQK15W (F, W, P1, P2, P3, P4, KP,
     X                   A, B, RESULT, ABSERR, RESABS, RESASC)
C
      INTEGER KP
      DOUBLE PRECISION F, W, P1, P2, P3, P4,
     X                 A, B, RESULT, ABSERR, RESABS, RESASC
C
C     COMPUTA I = INTEGRAL DE F*W SOBRE (A,B) COM ESTIMATIVA DE ERRO
C               J = INTEGRAL DE ABS(F*W) SOBRE (A,B)
C
C     PARAMETROS
C     NA CHAMADA
C        F       DOUBLE PRECISION FUNCTION
C                SUBPROGRAMA FUNCAO QUE DEFINE A FUNCAO INTEGRANDO F(X).
C                O NOME REAL DE F PRECISA SER DECLARADO NUM EXTERNAL NO
C                PROGRAMA ATIVADOR
```

```
C          W       DOUBLE PRECISION FUNCTION
C                  SUBPROGRAMA FUNCAO QUE DEFINE A FUNCAO PESO W(X) DO
C                  INTEGRANDO. O NOME DESTA FUNCAO PRECISA SER DECLARADO
C                  NUM EXTERNAL NO PROGRAMA ATIVADOR.
C          P1, P2
C           P3, P4 DOUBLE PRECISION
C                  PARAMETROS DA FUNCAO PESO.
C          A       DOUBLE PRECISION
C                  O LIMITE INFERIOR DO INTERVALO DE INTEGRACAO
C          B       DOUBLE PRECISION
C                  O LIMITE SUPERIOR DO INTERVALO DE INTEGRACAO
C          RESULT  DOUBLE PRECISION
C                  APENAS DECLARADO
C          ABSERR  DOUBLE PRECISION
C                  APENAS DECLARADO
C          RESABS  DOUBLE PRECISION
C                  APENAS DECLARADO
C          RESASC  DOUBLE PRECISION
C                  APENAS DECLARADO
C       NO RETORNO
C          RESULT  APROXIMACAO DA INTEGRAL I. RESULT E COMPUTADO USANDO A
C                  REGRA DE 15 PONTOS DE KONROD, OBTIDA PELA ADICAO OTIMA
C                  DE ABCISSAS A REGRA DE GAUSS DE 7 PONTOS.
C          ABSERR  ESTIMATIVA DO ERRO ABSOLUTO, QUE NAO DEVE EXCEDER
C                  ABS (I - RESULT)
C          RESABS  APROXIMACAO DA INTEGRAL J
C          RESASC  APROXIMACAO DA INTEGRAL DE ABS (F - I/(B -A)) SOBRE
C                  (A,B)
C
C       SUBPROGRAMAS UTILIZADOS
C          DO FORTRAN - DABS, DMAX1, DMIN1
C          DA BIBLIOTECA - DMAQ, DQWGTY, Y = C, F, S DEPENDENDO DO
C                          PROGRAMA ATIVADOR.
C          DO USUARIO - F
C
C       REFERENCIA
C          PIESSENS, DE DONCKER-KAPENGA, UBERHUBER, KAHANER, QUADPACK - A
C            SUBROUTINE PACKAGE FOR AUTOMATIC INTEGRATION, SPRINGER-VERLAG
C            BERLIN HEIDELBERG N. YORK TOKYO, 1983
C
C                  UNIVERSIDADE FEDERAL DA PARAIBA
C                  DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C                  PROJETO BITAN
C                  MARIO T. HATTORI  ABR/90
C
937         INTEGER J, JTW, JTWM1
938         DOUBLE PRECISION ABSC, ABSC1, ABSC2, CENTR, DHLGTH, EPMACH,
           X                 FC, FSUM, FVAL1, FVAL2,
           X                 FV1(7), FV2(7), HLGTH, OFLOW, ONE, ONEP5, RESG,
           X                 RESK, RESKH, TWO, UFLOW, WG(4), WGK(8), XGK(8),
           X                 ZERO
939         DOUBLE PRECISION DABS, DMAQ, DMAX1, DMIN1
C
C       AS ABCISSAS E PESOS SAO DADOS PARA O INTERVALO (-1,1). POR CAUSA
C       DA SIMETRIA SOMENTE AS ABCISSAS POSITIVAS E OS CORRESPONDENTES
C       PESOS SAO DADOS.
C
C       XGK   ABCISSAS DA REGRA DE 15 PONTOS DE KONROD
C             XGK(2), XGK(4), ... ABCISSAS DA REGRA DE 7 PONTOS DE GAUSS
C
```

```
C            XGK(1), XGK(3), ... ABCISSAS QUE FORAM ADICIONADAS
C            OTIMAMENTE A REGRA DE 7 PONTOS DE GAUSS.
C      WGK   PESOS DA REGRA DE 15 PONTOS DE KONROD
C      WG    PESOS DA REGRA DE 7 PONTOS DE GAUSS
C
940         DATA XGK /
           X           9.914553711208126D-01,  9.491079123427585D-01,
           X           8.648644233597691D-01,  7.415311855993944D-01,
           X           5.860872354676911D-01,  4.058451513773972D-01,
           X           2.077849550789850D-01,  0.000000000000000D+00 /
941         DATA WGK /
           X           2.293532201052922D-02,  6.309209262997855D-02,
           X           1.047900103222502D-01,  1.406532597155259D-01,
           X           1.690047266392679D-01,  1.903505780647854D-01,
           X           2.044329400752989D-01,  2.094821410847278D-01 /
942         DATA WG /
           X           1.294849661688697D-01,  2.797053914892767D-01,
           X           3.818300505051889D-01,  4.179591836734694D-01 /
C
943         DATA ZERO /0.D0/, ONE /1.D0/, ONEP5 /1.5D0/, TWO /2.D0/
C
944         EPMACH = DMAQ (4)
945         UFLOW = DMAQ (1)
946         OFLOW = DMAQ (2)
C
947         CENTR = (A + B) / TWO
948         HLGTH = (B - A) / TWO
949         DHLGTH = DABS (HLGTH)
C
C      COMPUTA A APROXIMACAO DA INTEGRAL PELA REGRA DE 15 PONTOS DE
C      KONROD E FAZ UMA ESTIMATIVA DO ERRO ABSOLUTO
C
950         FC = F (CENTR) * W (CENTR, P1, P2, P3, P4, KP)
951         RESG = FC * WG(4)
952         RESK = FC * WGK(8)
953         RESABS = DABS (RESK)
C
954         DO 20 J = 1,3
955            JTW = 2 * J
956            ABSC = HLGTH * XGK(JTW)
957            ABSC1 = CENTR - ABSC
958            ABSC2 = CENTR + ABSC
959            FVAL1 = F (ABSC1) * W (ABSC1, P1, P2, P3, P4, KP)
960            FVAL2 = F (ABSC2) * W (ABSC2, P1, P2, P3, P4, KP)
961            FV1(JTW) = FVAL1
962            FV2(JTW) = FVAL2
963            FSUM = FVAL1 + FVAL2
964            RESG = RESG + WG(J) * FSUM
965            RESK = RESK + WGK(JTW) * FSUM
966            RESABS = RESABS + WGK(JTW) * (DABS (FVAL1) + DABS (FVAL2))
967     20 CONTINUE
C
968         DO 40 J = 1,4
969            JTWM1 = 2 * J - 1
970            ABSC = HLGTH * XGK(JTWM1)
971            ABSC1 = CENTR - ABSC
972            ABSC2 = CENTR + ABSC
973            FVAL1 = F (ABSC1) * W (ABSC1, P1, P2, P3, P4, KP)
974            FVAL2 = F (ABSC2) * W (ABSC2, P1, P2, P3, P4, KP)
975            FV1(JTWM1) = FVAL1
```

```
976            FV2(JTWM1) = FVAL2
977            FSUM = FVAL1 + FVAL2
978            RESK = RESK + WGK(JTWM1) * FSUM
979            RESABS = RESABS + WGK(JTWM1) * (DABS (FVAL1) + DABS (FVAL2))
980       40 CONTINUE
      C
981         RESKH = RESK / TWO
982         RESASC = WGK(8) * DABS (FC - RESKH)
      C
983         DO 60 J = 1,7
984            RESASC = RESASC + WGK(J) * (DABS (FV1(J) - RESKH) +
          X                               DABS (FV2(J) - RESKH))
985       60 CONTINUE
      C
986         RESULT = RESK * HLGTH
987         RESABS = RESABS * DHLGTH
988         RESASC = RESASC * DHLGTH
989         ABSERR = DABS ((RESK - RESG) * HLGTH)
990         IF (RESASC .NE. ZERO .AND. ABSERR .NE. ZERO)  ABSERR =
          X     RESASC * DMIN1 (ONE, (200.D0 * ABSERR / RESASC)**ONEP5)
991         IF (RESABS .GT. UFLOW / (50.D0 * EPMACH)) ABSERR =
          X     DMAX1 ((EPMACH * 50.D0) * RESABS, ABSERR)
      C
992         RETURN
993         END

994         SUBROUTINE DQCHEB (X, FVAL, CHEB12, CHEB24)
      C
995         DOUBLE PRECISION X(11), FVAL(25), CHEB12(13), CHEB24(25)
      C
      C     COMPUTA A EXPANSAO EM SERIE DE CHEBYSHEV DE GRAUS 12 E 24 DE
      C     UMA FUNCAO USANDO UM METODO DE TRANSFORMACAO RAPIDA DE FOURIER
      C     F(X) = SOMA (CHEB12(K) * T(K-1,X)), K = 1, ..., 13,
      C     F(X) = SOMA (CHEB24(K) * T(K-1,X)), K = 1, ..., 25,
      C     EM T(K,X) E O POLINOMIO DE CHEBYSHEV DE GRAU K.
      C
      C     PARAMETROS
      C
      C     NA CHAMADA
      C        X        DOUBLE PRECISION (11)
      C                 CONTEM OS VALORES DE COS(K*PI/24), K = 1, ..., 11
      C        FVAL     DOUBLE PRECISION (25)
      C                 CONTEM OS VALORES DA FUNCAO NOS PONTOS
      C                     (B+A+(B-A)*COS(K*PI/24))/2
      C                 K = 0, ... , 24, EM QUE (A,B) E O INTERVALO DE
      C                 APROXIMACAO. FVAL(1) E FVAL(25) SAO DIVIDIDOS POR
      C                 DOIS (SAO DESTRUIDOS NO RETORNO)
      C        CHEB12   DOUBLE PRECISION (13)
      C                 APENAS DECLARADO
      C        CHEB24   DOUBLE PRECISION (25)
      C                 APENAS DECLARADO
      C
      C     NO RETORNO
      C        CHEB12   CONTEM OS COEFICIENTES DO POLINOMIO DE CHEBYSHEV DE
      C                 GRAU 12.
      C        CHEB24   CONTEM OS COEFICIENTES DO POLINOMIO DE CHEBYSHEV DE
      C                 GRAU 24.
      C
      C     REFERENCIA
      C        PIESSENS, DE DONCKER-KAPENGA, UBERHUBER, KAHANER, QUADPACK - A
```

```
      C           SUBROUTINE PACKAGE FOR AUTOMATIC INTEGRATION, SPRINGER-
      C           VERLAG, BERLIN HEIDELBERG N. YORK TOKYO, 1983
      C
      C                    UNIVERSIDADE FEDERAL DA PARAIBA
      C                    DEPARTAMENTO DE SISTEMAS E COMPUTACAO
      C                    PROJETO BITAN
      C                    MARIO T. HATTORI  ABR/90
      C
  996         INTEGER I, J
  997         DOUBLE PRECISION ALAM, ALAM1, ALAM2, ONE, PART1, PART2,
     X                         PART3, SIX, TWO, V(12)
  998         DATA ONE /1.D0/, TWO /2.D0/, SIX /6.D0/
      C
  999         DO 10 I = 1,12
 1000            J = 26 - I
 1001            V(I) = FVAL(I) - FVAL(J)
 1002            FVAL(I) = FVAL(I) + FVAL(J)
 1003       10 CONTINUE
      C
 1004         ALAM1 = V(1) - V(9)
 1005         ALAM2 = X(6) * (V(3) - V(7) - V(11))
 1006         CHEB12(4) = ALAM1 + ALAM2
 1007         CHEB12(10) = ALAM1 - ALAM2
 1008         ALAM1 = V(2) - V(8) - V(10)
 1009         ALAM2 = V(4) - V(6) - V(12)
 1010         ALAM = X(3) * ALAM1 + X(9) * ALAM2
 1011         CHEB24(4) = CHEB12(4) + ALAM
 1012         CHEB24(22) = CHEB12(4) - ALAM
 1013         ALAM = X(9) * ALAM1 - X(3) * ALAM2
 1014         CHEB24(10) = CHEB12(10) + ALAM
 1015         CHEB24(16) = CHEB12(10) - ALAM
 1016         PART1 = X(4) * V(5)
 1017         PART2 = X(8) * V(9)
 1018         PART3 = X(6) * V(7)
 1019         ALAM1 = V(1) + PART1 + PART2
 1020         ALAM2 = X(2) * V(3) + PART3 + X(10) * V(11)
 1021         CHEB12(2) = ALAM1 + ALAM2
 1022         CHEB12(12) = ALAM1 - ALAM2
 1023         ALAM = X(1) * V(2) + X(3) * V(4) + X(5) * V(6) + X(7) * V(8) +
     X           X(9) * V(10) + X(11) * V(12)
 1024         CHEB24(2) = CHEB12(2) + ALAM
 1025         CHEB24(24) = CHEB12(2) - ALAM
 1026         ALAM = X(11) * V(2) - X(9) * V(4) + X(7) * V(6) - X(5) * V(8) +
     X           X(3) * V(10) - X(1) * V(12)
 1027         CHEB24(12) = CHEB12(12) + ALAM
 1028         CHEB24(14) = CHEB12(12) - ALAM
 1029         ALAM1 = V(1) - PART1 + PART2
 1030         ALAM2 = X(10) * V(3) - PART3 + X(2) * V(11)
 1031         CHEB12(6) = ALAM1 + ALAM2
 1032         CHEB12(8) = ALAM1 - ALAM2
 1033         ALAM = X(5) * V(2) - X(9) * V(4) -X(1) * V(6) -
     X           X(11) * V(8) + X(3) * V(10) + X(7) * V(12)
 1034         CHEB24(6) = CHEB12(6) + ALAM
 1035         CHEB24(20) = CHEB12(6) - ALAM
 1036         ALAM = X(7) * V(2) - X(3) * V(4) - X(11) * V(6) +
     X           X(1) * V(8) - X(9) * V(10) - X(5) * V(12)
 1037         CHEB24(8) = CHEB12(8) + ALAM
 1038         CHEB24(18) = CHEB12(8) - ALAM
      C
 1039         DO 20 I = 1,6
```

```
1040          J = 14 - I
1041          V(I) = FVAL(I) - FVAL(J)
1042          FVAL(I) = FVAL(I) + FVAL(J)
1043       20 CONTINUE
      C
1044          ALAM1 = V(1) + X(8) * V(5)
1045          ALAM2 = X(4) * V(3)
1046          CHEB12(3) = ALAM1 + ALAM2
1047          CHEB12(11) = ALAM1 - ALAM2
1048          CHEB12(7) = V(1) - V(5)
1049          ALAM = X(2) * V(2) + X(6) * V(4) + X(10) * V(6)
1050          CHEB24(3) = CHEB12(3) + ALAM
1051          CHEB24(23) = CHEB12(3) - ALAM
1052          ALAM = X(6) * (V(2) - V(4) - V(6))
1053          CHEB24(7) = CHEB12(7) + ALAM
1054          CHEB24(19) = CHEB12(7) - ALAM
1055          ALAM = X(10) * V(2) - X(6) * V(4) + X(2) * V(6)
1056          CHEB24(11) = CHEB12(11) + ALAM
1057          CHEB24(15) = CHEB12(11) - ALAM
      C
1058          DO 30 I = 1,3
1059             J = 8 - I
1060             V(I) = FVAL(I) - FVAL(J)
1061             FVAL(I) = FVAL(I) + FVAL(J)
1062       30 CONTINUE
      C
1063          CHEB12(5) = V(1) + X(8) * V(3)
1064          CHEB12(9) = FVAL(1) - X(8) * FVAL(3)
1065          ALAM = X(4) * V(2)
1066          CHEB24(5) = CHEB12(5) + ALAM
1067          CHEB24(21) = CHEB12(5) - ALAM
1068          ALAM = X(8) * FVAL(2) - FVAL(4)
1069          CHEB24(9) = CHEB12(9) + ALAM
1070          CHEB24(17) = CHEB12(9) - ALAM
1071          CHEB12(1) = FVAL(1) + FVAL(3)
1072          ALAM = FVAL(2) + FVAL(4)
1073          CHEB24(1) = CHEB12(1) + ALAM
1074          CHEB24(25) = CHEB12(1) - ALAM
1075          CHEB12(13) = V(1) - V(3)
1076          CHEB24(13) = CHEB12(13)
1077          ALAM = ONE / SIX
      C
1078          DO 40 I = 2,12
1079             CHEB12(I) = CHEB12(I) * ALAM
1080       40 CONTINUE
      C
1081          ALAM = ALAM / TWO
1082          CHEB12(1) = CHEB12(1) * ALAM
1083          CHEB12(13) = CHEB12(13) * ALAM
      C
1084          DO 50 I = 2,24
1085             CHEB24(I) = CHEB24(I) * ALAM
1086       50 CONTINUE
      C
1087          CHEB24(1) = ALAM * CHEB24(1) / TWO
1088          CHEB24(25) = ALAM * CHEB24(25) / TWO
      C
1089          RETURN
1090          END
```

```
1091        INTEGER FUNCTION IMAQ(I)
      C
1092        INTEGER I
      C*****************************************************************
      C     ESTA FUNCAO FORNECE CONSTANTES DE UM AMBIENTE DE COMPUTACAO PARA *
      C     SEREM UTILIZADOS EM (SUB)PROGRAMAS TRANSPORTAVEIS EM FORTRAN.    *
      C                                                                      *
      C     ------ PARAMETRO------                                          *
      C                                                                      *
      C     NA CHAMADA                                                       *
      C        I          O VALOR DE I ENDERECA O PARAMETRO DESEJADO CONFORME *
      C                   DESCRICAO ABAIXO.DEVE SATISFAZER AS SEGUINTES      *
      C                   CONDICOES..  1 .LE. I .AND. I .LE. 16              *
      C                                                                      *
      C  NUMEROS LOGICOS DE ENTRADA/SAIDA                                    *
      C     IMAQ( 1) = NUMERO LOGICO DA UNIDADE PADRAO DE ENTRADA            *
      C     IMAQ( 2) = NUMERO LOGICO DA UNIDADE PADRAO DE SAIDA              *
      C     IMAQ( 3) = NUMERO LOGICO DA UNIDADE PADRAO DE PERFURACAO         *
      C     IMAQ( 4) = NUMERO LOGICO DA UNIDADE PADRAO DE MENSAGENS DE ERRO  *
      C                                                                      *
      C  PALAVRAS                                                            *
      C                                                                      *
      C     IMAQ( 5) = O NUMERO DE BITS POR UNIDADE DE ARMAZENAMENTO DE      *
      C                INTEIROS                                              *
      C     IMAQ( 6) = O NUMERO DE CARACTERES POR UNIDADE DE ARMAZENAMENTO   *
      C                DE INTEIRO                                            *
      C                                                                      *
      C  INTEIROS                                                            *
      C                                                                      *
      C     SUPONHA QUE OS INTEIROS SAO REPRESENTADOS EM S DIGITOS NA BASE A *
      C        SINAL(X(S-1)*A**(S-1) + ... + X(1)*A + X(0) )                 *
      C     EM QUE  0 .LE. X(I) .LT. A  PARA I = 0, ..., S-1.                *
      C                                                                      *
      C     IMAQ( 7) = A, A BASE                                             *
      C     IMAQ( 8) = S, O NUMERO DE DIGITOS NA BASE A                      *
      C     IMAQ( 9) = A**S-1, O MAIOR VALOR ABSOLUTO                        *
      C                                                                      *
      C  NUMEROS EM PONTO FLUTUANTE                                          *
      C                                                                      *
      C     SUPONHA QUE OS NUMEROS SAO REPRESENTADOS  EM T DIGITOS NA BASE B *
      C        SINAL (B**E)*( (X(1)/B + ...+ (X(T)/**T) )                    *
      C     EM QUE     0 .LE. X(I) .LT. B PARA I = 1, ... ,T                 *
      C                0 .LT. X(1) E   EMIN  .LE. EMAX                       *
      C                                                                      *
      C     IMAQ(10) = B, A BASE                                             *
      C                                                                      *
      C  PRECISAO SIMPLES                                                    *
      C                                                                      *
      C     IMAQ(11) = T, O NUMERO DE DIGITOS NA BASE B                      *
      C     IMAQ(12) = EMIN, O MENOR EXPOENTE                                *
      C     IMAQ(13) = EMAX, O MAIOR EXPOENTE                                *
      C                                                                      *
      C  DUPLA PRECISAO                                                      *
      C                                                                      *
      C     IMAQ(14) = T, O NUMERO DE DIGITOS NA BASE B                      *
      C     IMAQ(15) = EMIN, O MENOR EXPOENTE                                *
      C     IMAQ(16) = EMAX, O MAIOR EXPOENTE                                *
      C                                                                      *
      C     PARA ALTERAR ESTA FUNCAO PARA UM AMBIENTE PARTICULAR, O CONJUNTO *
      C     DE DECLARACOES  DATA DEVE SER ATIVADO REMOVENDO  O C  DA COLUNA 1*
```

```
C     A CONSISTENCIA DOS VALORES IMAQ(1) A IMAQ(4) COM O SISTEMA OPERA-*
C     CIONAL LOCAL DEVE SER VERIFCADA.                                 *
C-----------------------------------------------------------------------*
C     REFERENCIAS                                                      *
C     1. P. A. FOX, A. D. HALL, AND N. L. SCHRYER, THE PORT            *
C        MATHEMATICAL SUBROUTINE LIBRARY, ACM TRANS. MATH.             *
C        SOFTWARE 4(1978), PP 104-126.                                 *
C     2. P. A. FOX, A. D. HALL, AND N. L. SCHRYER, ALGORITHM 528 -     *
C        FRAMEWORK FOR A PORTABLE LIBRARY, ACM TRANS. MATH.            *
C        SOFTWARE 4(1978), PP 177-188.                                 *
C-----------------------------------------------------------------------*
C           UNIVERSIDADE FEDERAL DA PARAIBA                           *
C           DEPARTAMENTO DE SISTEMAS E COMPUTACAO                     *
C           MARIO T. HATTORI                                          *
C           VERSAO OUT/84                                             *
C***********************************************************************
C
1093        INTEGER IMACH(16), NOUT
C
1094        EQUIVALENCE (IMACH(4),NOUT)
C
C  -------- CONSTANTES PARA BURROUGHS 1700 ---------
C
C     DATA IMACH( 1) /     7 /
C     DATA IMACH( 2) /     2 /
C     DATA IMACH( 3) /     2 /
C     DATA IMACH( 4) /     2 /
C     DATA IMACH( 5) /    36 /
C     DATA IMACH( 6) /     4 /
C     DATA IMACH( 7) /     2 /
C     DATA IMACH( 8) /    33 /
C     DATA IMACH( 9) / Z1FFFFFFFF /
C     DATA IMACH(10) /     2 /
C     DATA IMACH(11) /    24 /
C     DATA IMACH(12) / -256 /
C     DATA IMACH(13) /  255 /
C     DATA IMACH(14) /    60 /
C     DATA IMACH(15) / -256 /
C     DATA IMACH(16) /  255 /
C
C  -------- CONSTANTES PARA BURROUGHS 5700 ---------
C
C     DATA IMACH( 1) /     5 /
C     DATA IMACH( 2) /     6 /
C     DATA IMACH( 3) /     7 /
C     DATA IMACH( 4) /     6 /
C     DATA IMACH( 5) /    48 /
C     DATA IMACH( 6) /     6 /
C     DATA IMACH( 7) /     2 /
C     DATA IMACH( 8) /    39 /
C     DATA IMACH( 9) / O0007777777777777 /
C     DATA IMACH(10) /     8 /
C     DATA IMACH(11) /    13 /
C     DATA IMACH(12) /   -50 /
C     DATA IMACH(13) /    76 /
C     DATA IMACH(14) /    26 /
C     DATA IMACH(15) /   -50 /
C     DATA IMACH(16) /    76 /
C
C  -------- CONSTANTES PARA BURROUGHS 6700/7700 ---------
```

```
C
C      DATA IMACH( 1) /    5 /
C      DATA IMACH( 2) /    6 /
C      DATA IMACH( 3) /    7 /
C      DATA IMACH( 4) /    6 /
C      DATA IMACH( 5) /   48 /
C      DATA IMACH( 6) /    6 /
C      DATA IMACH( 7) /    2 /
C      DATA IMACH( 8) /   39 /
C      DATA IMACH( 9) / O0007777777777777 /
C      DATA IMACH(10) /    8 /
C      DATA IMACH(11) /   13 /
C      DATA IMACH(12) /  -50 /
C      DATA IMACH(13) /   76 /
C      DATA IMACH(14) /   26 /
C      DATA IMACH(15) / -32754 /
C      DATA IMACH(16) /  32780 /
C
C      CONSTANTES PARA CDC SERIES 6000/7000
C
C      DATA IMACH( 1) /    5 /
C      DATA IMACH( 2) /    6 /
C      DATA IMACH( 3) /    7 /
C      DATA IMACH( 4) /    6 /
C      DATA IMACH( 5) /   60 /
C      DATA IMACH( 6) /   10 /
C      DATA IMACH( 7) /    2 /
C      DATA IMACH( 8) /   48 /
C      DATA IMACH( 9) / 00007777777777777777B /
C      DATA IMACH(10) /    2 /
C      DATA IMACH(11) /   48 /
C      DATA IMACH(12) / -974 /
C      DATA IMACH(13) / 1070 /
C      DATA IMACH(14) /   96 /
C      DATA IMACH(15) / -927 /
C      DATA IMACH(16) / 1070 /
C
C     ------- CONSTANTES PARA SERIES HONEYWELL 600/6000
C
C      DATA IMACH( 1) /    5 /
C      DATA IMACH( 2) /    6 /
C      DATA IMACH( 3) /   43 /
C      DATA IMACH( 4) /    6 /
C      DATA IMACH( 5) /   36 /
C      DATA IMACH( 6) /    6 /
C      DATA IMACH( 7) /    2 /
C      DATA IMACH( 8) /   35 /
C      DATA IMACH( 9) / O377777777777 /
C      DATA IMACH(10) /    2 /
C      DATA IMACH(11) /   27 /
C      DATA IMACH(12) / -127 /
C      DATA IMACH(13) /  127 /
C      DATA IMACH(14) /   63 /
C      DATA IMACH(15) / -127 /
C      DATA IMACH(16) /  127 /
C
C     ------- CONSTANTES PARA AS SERIES IBM 360/370/3000/4000
C
1095        DATA IMACH( 1) /    5 /
1096        DATA IMACH( 2) /    6 /
```

```
      DATA IMACH( 3) /    7 /
      DATA IMACH( 4) /    6 /
      DATA IMACH( 5) /   32 /
      DATA IMACH( 6) /    4 /
      DATA IMACH( 7) /    2 /
      DATA IMACH( 8) /   31 /
      DATA IMACH( 9) / Z7FFFFFFF /
                     $
EXT* DA-06 data initialization with hexadecimal constant is not FORTRAN 77 stan
      DATA IMACH(10) /   16 /
      DATA IMACH(11) /    6 /
      DATA IMACH(12) /  -64 /
      DATA IMACH(13) /   63 /
      DATA IMACH(14) /   14 /
      DATA IMACH(15) /  -64 /
      DATA IMACH(16) /   63 /
C
C     CONSTANTES PARA PDP-10 PROCESSADOR KI
C
C     DATA IMACH( 1) /         5/
C     DATA IMACH( 2) /         6/
C     DATA IMACH( 3) /         5/
C     DATA IMACH( 4) /         3/
C     DATA IMACH( 5) /        36/
C     DATA IMACH( 6) /         5/
C     DATA IMACH( 7) /         2/
C     DATA IMACH( 8) /        35/
C     DATA IMACH( 9) /"377777777777/
C     DATA IMACH(10) /         2/
C     DATA IMACH(11) /        27/
C     DATA IMACH(12) /    -128 /
C     DATA IMACH(13) /      127/
C     DATA IMACH(14) /       62/
C     DATA IMACH(15) /     -128/
C     DATA IMACH(16) /      127/
C
C     CONSTANTES PARA O FORTRAN DO PDP-11 QUE SUPORTA ARITMETICA INTEIRA
C     DE 32 BITS.
C
C     DATA IMACH( 1) /          5 /
C     DATA IMACH( 2) /          6 /
C     DATA IMACH( 3) /          5 /
C     DATA IMACH( 4) /          6 /
C     DATA IMACH( 5) /         32 /
C     DATA IMACH( 6) /          4 /
C     DATA IMACH( 7) /          2 /
C     DATA IMACH( 8) /         31 /
C     DATA IMACH( 9) / 2147483647 /
C     DATA IMACH(10) /          2 /
C     DATA IMACH(11) /         24 /
C     DATA IMACH(12) /       -128 /
C     DATA IMACH(13) /        127 /
C     DATA IMACH(14) /         56 /
C     DATA IMACH(15) /       -127 /
C     DATA IMACH(16) /        127 /
C
C     CONSTANTES PARA O FORTRAN DO PDP-11 QUE SUPORTA ARITMETICA INTEIRA
C     DE 16 BITS.
C
C     DATA IMACH( 1) /          5 /
```

```
C      DATA IMACH( 2) /           6 /
C      DATA IMACH( 3) /           5 /
C      DATA IMACH( 4) /           6 /
C      DATA IMACH( 5) /          16 /
C      DATA IMACH( 6) /           2 /
C      DATA IMACH( 7) /           2 /
C      DATA IMACH( 8) /          15 /
C      DATA IMACH( 9) /       32767 /
C      DATA IMACH(10) /           2 /
C      DATA IMACH(11) /          24 /
C      DATA IMACH(12) /        -127 /
C      DATA IMACH(13) /         127 /
C      DATA IMACH(14) /          56 /
C      DATA IMACH(15) /        -127 /
C      DATA IMACH(16) /         127 /
C
C      CONSTANTES PARA A SERIE UNIVAC 1100
C
C      OBSERVE QUE A UNIDADE PERFURADORA, IMACH(3), RECEBEU O NUMERO 7, O
C      ADEQUADO PARA O SISTEMA UNIVAC-FOR. SE VOCE TEM OS SISTEMA
C      UNIVAC-FTN, ATRIBUA O NUMERO 1.
C
C      DATA IMACH( 1) /           5/
C      DATA IMACH( 1) /           5/
C      DATA IMACH( 2) /           6/
C      DATA IMACH( 3) /           7/
C      DATA IMACH( 4) /           6/
C      DATA IMACH( 5) /          36/
C      DATA IMACH( 6) /           6/
C      DATA IMACH( 7) /           2/
C      DATA IMACH( 8) /          35/
C      DATA IMACH( 9) / 0377777777777 /
C      DATA IMACH(10) /           2/
C      DATA IMACH(11) /          27/
C      DATA IMACH(12) /        -128/
C      DATA IMACH(13) /         127/
C      DATA IMACH(14) /          60/
C      DATA IMACH(15) /       -1024/
C      DATA IMACH(16) /        1023 /
C
1111        IF (I .LT. 1 .OR. I.GT. 16) GO TO 20
1112           IMAQ = IMACH(I)
1113           RETURN
1114     20 CONTINUE
1115        WRITE(NOUT,9000)
1116   9000 FORMAT(33HERRO EM IMAQ - I FORA DOS LIMITES)
1117        RETURN
1118        END
C

1119        DOUBLE PRECISION FUNCTION DMAQ(I)
C
1120        INTEGER I
C
C      ESTA FUNCAO FORNECE OS PARAMETROS DO PONTO FLUTUANTE EM PRECISAO
C
C      ------ PARAMETRO ------
C
C      NA CHAMADA
C         I          INTEGER
```

```
C                   ENDERECA O PARAMETRO DESEJADO CONFORME DESCRICAO
C                   ABAIXO. O VALOR DE I DEVE SATISFAZER AS SEGUINTES
C                   CONDICOES..     1 .LE. I .AND. I .LE. 5
C
C     CONSTANTES DE MAQUINA PARA PRECISAO SIMPLES.
C
C     DMACH(1) = B**(EMIN - 1), O MENOR POSITIVO
C
C     DMACH(2) = B**EMAX(1 - B**(-T)), O MAIOR POSITIVO
C
C     DMACH(3) = B**(-T), O MENOR ESPA$AMENTO RELATIVO
C
C     DMACH(4) = B**(1 - T), O MAIOR ESPA$AMENTO RELATIVO
C
C     DMACH(5) = LOG10(B)
C
C     CONSTANTES PARA OUTROS EQUIPAMENTOS PODEM SER ENCONTRADAS NA
C     REFERENCIA 2 ABAIXO CITADA.
C
C     REFERENCIAS
C     1. P. A. FOX, A. D. HALL, AND N. L. SCHRYER, THE PORT
C        MATHEMATICAL SUBROUTINE LIBRARY, ACM TRANS. MATH.
C        SOFTWARE 4(1978), PP 104-126.
C     2. P. A. FOX, A. D. HALL, AND N. L. SCHRYER, ALGORITHM 528 -
C        FRAMEWORK FOR A PORTABLE LIBRARY, ACM TRANS. MATH.
C        SOFTWARE 4(1978), PP 177-188.
C
C           UNIVERSIDADE FEDERAL DA PARAIBA
C           DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C           PROJETO MICRO-BITAN
C           MARIO T. HATTORI  OUT/84
C           ESTA VERSAO MAI/89
C
 1121       CHARACTER*1 STAR
 1122       DOUBLE PRECISION DMACH(5)
 1123       INTEGER DIVER(2), LARGE(2), RIGHT(2), SMALL(2)
C
 1124       EQUIVALENCE (DMACH(1), SMALL(1))
 1125       EQUIVALENCE (DMACH(2), LARGE(1))
 1126       EQUIVALENCE (DMACH(3), RIGHT(1))
 1127       EQUIVALENCE (DMACH(4), DIVER(1))
 1128       DATA STAR / '*' /
 1129       DATA SMALL(1), SMALL(2) / 0,     1048576 /
 1130       DATA LARGE(1), LARGE(2) /-1, 2146435071 /
 1131       DATA RIGHT(1), RIGHT(2) / 0, 1016070144 /
 1132       DATA DIVER(1), DIVER(2) / 0, 1017118720 /
 1133       DATA DMACH(5) /  3.010299956639812D+00 /
C
 1134       IF(I .LT. 1 .OR. I .GT. 5) WRITE(STAR, 9000)
 1135  9000 FORMAT(32HERRO EM DMAQ. I FORA DOS LIMITES)
 1136       DMAQ = DMACH(I)
C
 1137       RETURN
 1138       END
```

```
ompile time:                    28.73  Execution time:                02.36
ize of object code:             30056  Number of extensions:             1
ize of local data area(s):       9759  Number of warnings:               0
```

```
ize of global data area:          56609  Number of errors:              0
bject/Dynamic bytes free:   92575/42534  Statements Executed:        9982
```

LISTAGEM DO PACOTE DAQWFT UTILIZANDO DQFOUR EM FORTRAN-77

```
318         IF (IER + IERRO .EQ. 0) GOTO 110
319         IF (IERRO .EQ. 3) ABSERR = ABSERR + CORREC
320         IF (IER .EQ. 0) IER = 3
321         IF (RESULT .NE. ZERO .AND. AREA .NE. ZERO) GOTO 105
322         IF (ABSERR .GT. ERRSUM) GOTO 115
323         IF (AREA .EQ. ZERO) GOTO 130
324         GOTO 110
325    105  CONTINUE
326         IF (ABSERR / DABS (RESULT) .GT. ERRSUM / DABS (AREA)) GOTO 115
      C
      C     TESTA DIVERGENCIA
      C
327    110  CONTINUE
328         IF (KSGN .EQ. (-1) .AND. DMAX1 (DABS (RESULT), DABS (AREA)) .LE.
           X    DEFABS * TENM2) GOTO 130
329         IF (TENM2 .GT. (RESULT / AREA) .OR. (RESULT / AREA) .GT. HUNDRD
           X     .OR. ERRSUM .GT. DABS (AREA)) IER = 6
330         GOTO 130
331    115  CONTINUE
      C
      C     COMPUTA A INTEGRAL
      C
332         RESULT = ZERO
333         DO 120 K = 1,LAST
334            RESULT = RESULT + RLIST(K)
335    120  CONTINUE
336         ABSERR = ERRSUM
337    130  CONTINUE
338         NEVAL = 30 * LAST - 15
339         IF (IER .GT. 2) IER = IER - 1
340         RETURN
341         END
      C

342         SUBROUTINE DQFOUR (F, A, B, OMEGA, INTEGR, EPSABS, EPSREL, LIMIT,
           X                   ICALL, MAXP1, RESULT, ABSERR, NEVAL, IER,
           X                   ALIST, BLIST, RLIST, ELIST, IORD, NNLOG,
           X                   MOMCOM, CHEBMO)
      C
343         INTEGER INTEGR, LIMIT, ICALL, MAXP1, NEVAL, IER, IORD(LIMIT),
           X        NNLOG(LIMIT), MOMCOM
344         DOUBLE PRECISION F, A, B, OMEGA, EPSABS, EPSREL, RESULT, ABSERR,
           X                 ALIST(LIMIT), BLIST(LIMIT), RLIST(LIMIT),
           X                 ELIST(LIMIT), CHEBMO(MAXP1,25)
      C
      C     CALCULA UMA APROXIMACAO DE
      C          I = INTEGRAL DE F(X)*W(X) SOBRE (A,B),
      C     EM QUE
      C          W(X) = COS(OMEGA*X) OU
      C          W(X) = SIN(OMEGA*X),
      C     ESPERANDO QUE O ERRO COMETIDO SATISFACA
      C          ABS (I - RESULT) .LE. MAX (EPSABS, EPSREL*ABS(I)).
      C     A ROTINA E CHAMADA POR DQAWO E DQAWF. CONTUDO, PODE SER
      C     CHAMADO DIRETAMENTE PELO USUARIO.
      C
      C     PARAMETROS
      C
      C     NA CHAMADA
      C        F       DOUBLE PRECISION FUNCTION
      C                E A FUNCAO INTEGRANDO DEFINIDA PELO USUARIO EM UM
```

```
C               SUBPROGRAMA FUNCAO. O NOME REAL DE F PRECISA SER
C               DECLARADO NUM EXTERNAL NO PROGRAMA ATIVADOR.
C         A     DOUBLE PRECISION
C               LIMITE INFERIOR DO INTERVALO DE INTEGRACAO
C         B     DOUBLE PRECISION
C               LIMITE SUPERIOR DO INTERVALO DE INTEGRACAO
C         OMEGA DOUBLE PRECISION
C               PARAMETRO DA FUNCAO PESO
C         INTEGR INTEGER
C               INDICA QUAL A FUNCAO PESO A SER UTILIZADA. SE INTEGR
C               = 1  W(X) = COS(OMEGA*X)
C               = 2  W(X) = SIN(OMEGA*X)
C               QUALQUER OUTRO VALOR DE INTEGR SINALIZARA ERRO.
C         EPSABS DOUBLE PRECISION
C               TOLERANCIA DE ERRO ABSOLUTO SOLICITADA
C         EPSREL DOUBLE PRECISION
C               TOLERANCIA DE ERRO RELATIVO SOLICITADA. SE EPSABS E
C               EPSREL FOREM NEGATIVOS DQFOUR SINALIZARA A OCORRENCIA
C               DE ERRO.
C         LIMIT  INTEGER
C               E O LIMITE SUPERIOR NO NUMERO DE SUBDIVISOES DO
C               INTERVALO (A,B). DEVE SER MAIOR OU IGUAL A 1.
C         ICALL  INTEGER
C               SE DQFOUR FOR SER USADO UMA UNICA VEZ, ICALL DEVE TER
C               VALOR 1. SUPONHA QUE DURANTE ESSA CHAMADA UNICA OS
C               MOMENTOS DE CHEBYSHEV (PARA O METODO DE INTEGRACAO
C               DE CLENSHAW-CURTIS DE GRAU 24) FORAM CALCULADOS PARA
C               INTERVALOS DE COMPRIMENTOS (ABS(B-A))*2**(-L),
C               L = 0, 1, 2, ... , MOMCOM-1. OS MOMENTOS DE CHEBYSHEV
C               JA CALCULADOS PODEM SER SER RE-UTILIZADOS EM CHAMADAS
C               SUBSEQUENTES, SE DQFOUR DEVE SER CHAMADO DUAS OU MAIS
C               VEZES NO INTERVALO DE MESMO TAMANHO ABS(B-A). A PARTIR
C               DA SEGUNDA CHAMADA DEVE-SE USAR ICALL MAIOR QUE 1. SE
C               ICALL FOR MENOR QUE 1 A ROTINA SINALIZARA UMA
C               OCORRENCIA DE ERRO.
C         MAXP1  INTEGER
C               FORNECE UM LIMITE SUPERIOR NO NUMERO DE MOMENTOS DE
C               CHEBYSHEV QUE PODE SER ARMAZENADO, ISTO E, PARA
C               INTERVALOS DE TAMANHOS ABS(B-A)*2**(-L),
C               L = 0, 1, 2, ... , MAXP1-2 COM MAXP1 MAIOR OU
C               IGUAL A 1. SE MAXP1 FOR MENOR QUE 1 DQFOUR TERMINARA
C               SINALIZANDO OCORRENCIA DE ERRO.
C         RESULT DOUBLE PRECISION
C               APENAS DECLARADO
C         ABSERR DOUBLE PRECISION
C               APENAS DECLARADO
C         NEVAL  INTEGER
C               APENAS DECLARADO
C         IER    INTEGER
C               APENAS DECLARADO
C         ALIST  DOUBLE PRECISION(LIMIT)
C               APENAS DECLARADO
C         BLIST  DOUBLE PRECISION(LIMIT)
C               APENAS DECLARADO
C         RLIST  DOUBLE PRECISION(LIMIT)
C               APENAS DECLARADO
C         ELIST  DOUBLE PRECISION(LIMIT)
C               APENAS DECLARADO
C         IORD   INTEGER(LIMIT)
C               APENAS DECLARADO
```

```
C          NNLOG   INTEGER(LIMIT)
C                  APENAS DECLARADO
C          MOMCOM  INTEGER
C                  NA PRIMEIRA CHAMADA DEVE TER VALOR ZERO
C          CHEBMO  DOUBLE PRECISION(MAXP1,25)
C                  APENAS DECLARADO NA PRIMEIRA CHAMADA
C
C      NO RETORNO
C          RESULT  APROXIMACAO DA INTEGRAL
C          ABSERR  ESTIMATIVA DO MODULO DO ERRO ABSOLUTO QUE DEVE
C                  SER MAIOR OU IGUAL A ABS(I - RESULT).
C          NEVAL   O NUMERO DE AVALIACOES DO INTEGRANDO.
C          IER     SE O SEU VALOR FOR
C                  = 0 TERMINO NORMAL DA ROTINA. SUPOE-SE QUE A
C                      TOLERANCIA DE ERRO SOLICITADA FOI SATISFEITA
C                  = 1 O MAXIMO NUMERO SOLICITADO DE SUBDIVISOES DO
C                      INTERVALO FOI ATINGIDO. PODE-SE PERMITIR MAIS
C                      SUBDIVISOES AUMENTANDO-SE O VALOR DE LIMIT
C                      (PRECISA AJUSTAR DIMENSOES). SE ESSE AUMENTO NAO
C                      ACARRETAR MELHORIA NO RESULTADO ACONSELHA-SE
C                      ANALISAR O INTEGRANDO A FIM DE AVERIGUAR AS
C                      DIFICULDADES DE INTEGRACAO. SE A DIFICULDADE PUDER
C                      SER LOCALIZADA (POR EXEMPLO, SINGULARIDADE, DES-
C                      CONTINUIDADE DENTRO DO INTERVALO) VALERA A PENA
C                      SUBDIVIDIR O INTERVALO NESSE PONTO E CHAMAR O
C                      DQFOUR EM CADA SUBINTERVALO. SE POSSIVEL, ESCOLHER
C                      UMA ROTINA ESPECIALIZADA QUE SEJA CAPAZ DE CONTOR-
C                      NAR A DIFICULDADE DETETADA.
C                  = 2 FOI DETETADA A OCORRENCIA DE ERRO DE ARREDONDAMENTO
C                      QUE NAO PERMITE ATINGIR A TOLERANCIA DE ERRO
C                      SOLICITADA.
C                  = 3 UM MAU COMPORTAMENTO EXTREMO DO INTEGRANDO OCORRE
C                      EM ALGUNS PONTOS NO INTERVALO DE INTEGRACAO.
C                  = 4 O ALGORITMO NAO CONVERGE. ERRO DE ARREDONDAMENTO
C                      E DETETADO NA TABELA DE EXTRAPOLACAO. PRESUME-SE
C                      QUE A TOLERANCIA SOLICITADA NAO PODE SER ALCANCADA
C                      DEVIDO AO ERRO DE ARREDONDAMENTO NA TABELA E QUE
C                      O RESULTADO FORNECIDO E O MELHOR QUE SE PODE OBTER.
C                  = 5 A INTEGRAL E PROVAVELMENTE DIVERGENTE OU LENTAMEN-
C                      TE CONVERGENTE. DEVE-SE OBSERVAR QUE A DIVERGENCIA
C                      PODE OCORRER COM QUALQUER OUTRO VALOR DE IER DIFE-
C                      RENTE DE ZERO.
C                  = 6 HOUVE ERRO NOS DADOS DE ENTRADA (EPSABS  E EPSREL
C                      NEGATIVOS, OU INTEGR NAO 1 OU 2, OU ICALL MENOR QUE
C                      1 OU AINDA MAXP1 MENOR QUE 1). OS VALORES DE
C                      RESULT, ABSERR, NEVAL, LAST, RLIST(1), ELIST(1),
C                      IORD(1) E NNLOG(1) SAO NULOS. ALIST(1) E BLIST(1)
C                      CONTERAO OS VALORES DE A E DE B, RESPECTIVAMENTE.
C          ALIST   OS PRIMEIROS LAST ELEMENTOS SAO OS LIMITES INFERIORES
C                  DOS SUBINTERVALOS NA PARTICAO DO INTERVALO (A,B) DE
C                  INTEGRACAO.
C          BLIST   OS PRIMEIROS LAST ELEMENTOS SAO OS LIMITES SUPERIORES
C                  DOS SUBINTERVALOS NA PARTICAO DO INTERVALO (A,B) DE
C                  INTEGRACAO.
C          RLIST   OS PRIMEIROS LAST ELEMENTOS CONTEM APROXIMACOES
C                  DAS INTEGRAIS NOS SUBINTERVALOS.
C          ELIST   OS PRIMEIROS LAST ELEMENTOS CONTEM OS MODULOS DAS
C                  ESTIMATIVAS DOS ERROS ABSOLUTOS NOS SUBINTERVALOS.
C          IORD    OS PRIMEIROS K ELEMENTOS SAO APONTADORES PARA AS
C                  ESTIMATIVAS DE ERROS NOS SUBINTERVALOS, DE MODO QUE
```

```
C                ELIST(IORD(1)), ... , ELIST(IORD(K)) FORMAM UMA
C                SEQUENCIA DECRESCENTE COM K = LAST SE LAST FOR
C                MENOR OU IGUAL LIMIT/2 + 2 E K = LIMIT + 1 - LAST
C                EM OUTROS CASOS.
C          NNLOG   INDICA OS NIVEIS DE SUBDIVISAO, OU SEJA, SE
C                NNLOG(I) = L SIGNIFICA QUE O SUBINTERVALO DE NUMERO I
C                TEM TAMANHO ABS(B-A)*2**(-L).
C          MOMCOM  INDICA QUE OS MOMENTOS DE CHEBYSHEV FORAM CALCULADOS
C                PARA INTERVALOS DE TAMANHO (ABS(B-A)*2**(-L),
C                L = 0, 1, ... , MOMCOM-1.
C          CHEBMO  CONTEM OS ELEMENTOS DE CHEBYSHEV.
C
C       SUBPROGRAMAS UTILIZADOS
C          DO FORTRAN - DABS, DMAX1
C          DA BIBLIOTECA - DQEXT, DQC250, DQSORT
C          DO USUARIO - F
C
C       REFERENCIA
C          PIESSENS, DE DONCKER-KAPENGA, UBERHUBER, KAHANER, QUADPACK - A
C            SUBROUTINE PACKAGE FOR AUTOMATIC INTEGRATION, SPRINGER-
C            VERLAG, BERLIN HEIDELBERG N. YORK TOKYO, 1983
C
C                    UNIVERSIDADE FEDERAL DA PARAIBA
C                    DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C                    PROJETO BITAN
C                    MARIO T. HATTORI  ABR/90
C
345         INTEGER ID, IERRO, IROFF1, IROFF2, IROFF3, JUPBND, K,
           X        KSGN, KTMIN, LAST, MAXERR, NEV,
           X        NRES, NRMAX, NRMOM, NUMRL2
346         LOGICAL EXTRAP, NOEXT, EXTALL
347         DOUBLE PRECISION ABSEPS, AREA, AREA1, AREA12, AREA2,
           X                 A1, A2, B1, B2, CORREC, DEFAB1, DEFAB2,
           X                 DEFABS, DOMEGA, DRES, EPMACH, ERLARG,
           X                 ERLAST, ERRBND, ERRMAX, ERROR1, ERRO12,
           X                 ERROR2, ERRSUM, ERTEST, OFLOW, RESABS, RESEPS,
           X                 RES3LA(3), RLIST2(52), SMALL, UFLOW, WIDTH
348         DOUBLE PRECISION DABS, DMAX1, DMAQ, DQWGTO
349         EXTERNAL F
350         EPMACH = DMAQ (4)
351         UFLOW = DMAQ (1)
352         OFLOW = DMAQ (2)
C
C     VERIFICA VALIDADE DOS PARAMETROS
C
353         IER = 0
354         NEVAL = 0
355         LAST = 0
356         RESULT = 0.0E+00
357         ABSERR = 0.0E+00
358         ALIST(1) = A
359         BLIST(1) = B
360         RLIST(1) = 0.0E+00
361         ELIST(1) = 0.0E+00
362         IORD(1) = 0
363         NNLOG(1) = 0
C
364         IF ((INTEGR .NE. 1 .AND. INTEGR .NE. 2) .OR. (EPSABS .LT.
           X   0.0E+00 .AND. EPSREL .LT. 0.0E+00) .OR. ICALL .LT. 1 .OR.
           X   MAXP1 .LT. 1)  THEN
```

```
365           IER = 6
366         ENDIF
      C
      C   ERRO NO(S) PARAMETRO(S), RETORNA.
      C
367         IF (IER .NE. 6) THEN
      C
      C   PRIMEIRA APROXIMACAO DA INTEGRAL
      C
368           DOMEGA = DABS (OMEGA)
369           NRMOM = 0
370           IF (ICALL .LE. 1) THEN
371              MOMCOM = 0
372           ENDIF
373           CALL DQC250 (F, A, B, DOMEGA, INTEGR, NRMOM, MAXP1, 0, RESULT,
     X          ABSERR, NEVAL, DEFABS, RESABS, MOMCOM, CHEBMO)
      C
      C   COMPARA ERRO COM TOLERANCIA DE ERRO
      C
374           DRES = DABS (RESULT)
375           ERRBND = DMAX1 (EPSABS, EPSREL * DRES)
376           RLIST(1) = RESULT
377           ELIST(1) = ABSERR
378           IORD(1) = 1
379           IF (ABSERR .LE. 1.0E+02 * EPMACH * DEFABS .AND. ABSERR .GT.
     X        ERRBND)  THEN
380              IER = 2
381           ENDIF
382           IF (LIMIT .EQ. 1)  THEN
383              IER = 1
384           ENDIF
      C
      C   RETORNA SE TOLERANCIA DE ERRO SATISFEITA
      C
385           IF (IER .EQ. 0 .AND. ABSERR .GT. ERRBND) THEN
      C
      C   ESTABELECE CONDICOES INICIAIS
      C
386              ERRMAX = ABSERR
387              MAXERR = 1
388              AREA = RESULT
389              ERRSUM = ABSERR
390              ABSERR = OFLOW
391              NRMAX = 1
392              EXTRAP = .FALSE.
393              NOEXT = .FALSE.
394              IERRO = 0
395              IROFF1 = 0
396              IROFF2 = 0
397              IROFF3 = 0
398              KTMIN = 0
399              SMALL = DABS (B - A) * 7.5E-01
400              NRES = 0
401              NUMRL2 = 0
402              EXTALL = .FALSE.
403              IF (5.0E-01 * DABS (B - A) * DOMEGA .LE. 2.0E+00) THEN
404                 NUMRL2 = 1
405                 EXTALL = .TRUE.
406                 RLIST2(1) = RESULT
407              ENDIF
```

```
408               IF (2.5E-01 * DABS (B - A) * DOMEGA .LE. 2.0E+00)  THEN
409                  EXTALL = .TRUE.
410               ENDIF
411               KSGN = -1
412               IF (DRES .GE. (1.0E+00 - 5.0E+01 * EPMACH) * DEFABS)  THEN
413                  KSGN = 1
414               ENDIF
      C
      C    CICLO PRINCIPAL
      C
415               DO 140 LAST = 2,LIMIT
      C
      C    BISSECAO DO SUBINTERVALO COM NRMAX-ESIMA MAIOR ESTIMATIVA DE ERRO
      C
416                  NRMOM = NNLOG(MAXERR) + 1
417                  A1 = ALIST(MAXERR)
418                  B1 = 5.0E-01 * (ALIST(MAXERR) + BLIST(MAXERR))
419                  A2 = B1
420                  B2 = BLIST(MAXERR)
421                  ERLAST = ERRMAX
422                  CALL DQC250 (F, A1, B1, DOMEGA, INTEGR, NRMOM, MAXP1, 0,
       X                  AREA1, ERROR1, NEV, RESABS, DEFAB1, MOMCOM, CHEBMO)
423                  NEVAL = NEVAL + NEV
424                  CALL DQC250 (F, A2, B2, DOMEGA, INTEGR, NRMOM, MAXP1, 0,
       X                  AREA2, ERROR2, NEV, RESABS, DEFAB2, MOMCOM, CHEBMO)
425                  NEVAL = NEVAL + NEV
      C
      C    MELHORA AS APROXIMACOES DA INTEGRAL E DO ERRO E TESTA EXATIDAO
      C
426                  AREA12 = AREA1 + AREA2
427                  ERRO12 = ERROR1 + ERROR2
428                  ERRSUM = ERRSUM + ERRO12 - ERRMAX
429                  AREA = AREA + AREA12 - RLIST(MAXERR)
      C
430                  IF (DEFAB1 .NE. ERROR1 .AND. DEFAB2 .NE. ERROR2) THEN
      C
431                     IF (DABS (RLIST(MAXERR) - AREA12) .LE. 1.0E-05 *
       X                     DABS (AREA12) .AND. ERRO12 .GE. 9.9E-01 *
       X                     ERRMAX) THEN
432                        IF (EXTRAP)  THEN
433                           IROFF2 = IROFF2 + 1
434                        ELSE
435                           IROFF1 = IROFF1 + 1
436                        ENDIF
437                     ENDIF
438                     IF (LAST .GT. 10 .AND. ERRO12 .GT. ERRMAX)  THEN
439                        IROFF3 = IROFF3 + 1
440                     ENDIF
441                  ENDIF
442                  RLIST(MAXERR) = AREA1
443                  RLIST(LAST) = AREA2
444                  NNLOG(MAXERR) = NRMOM
445                  NNLOG(LAST) = NRMOM
446                  ERRBND = DMAX1 (EPSABS, EPSREL * DABS (AREA))
      C
      C    TESTA ERRO DE ARREDONDAMENTO E EVENTUALMENTE SINALIZA ERRO
      C
447                  IF (IROFF1 + IROFF2 .GE. 10 .OR. IROFF3 .GE. 20)  THEN
448                     IER = 2
449                  ENDIF
```

```
450                IF (IROFF2 .GE. 5)  THEN
451                   IERRO = 3
452                ENDIF
     C
     C    SINALIZA ERRO EM CASO DE NUMERO DE SUBINTERVALOS SE TORNAR IGUAL
     C       A LIMIT
     C
453                IF (LAST .EQ. LIMIT)  THEN
454                   IER = 1
455                ENDIF
     C
     C    SINALIZA ERRO NO CASO DE MAU COMPORTAMENTO DO INTEGRANDO NUM
     C       PONTO INTERIOR AO INTERVALO DE INTEGRACAO
     C
456                IF (DMAX1 (DABS (A1), DABS (B2)) .LE. (1.0E+00 +
      X                1.0E+03 * EPMACH) * (DABS (A2) + 1.0E+03 *
      X                UFLOW))  THEN
457                   IER = 4
458                ENDIF
     C
     C    ADICIONA OS INTERVALOS RECEM CRIADOS A LISTA
     C
459                IF (ERROR2 .LE. ERROR1) THEN
460                   ALIST(LAST) = A2
461                   BLIST(MAXERR) = B1
462                   BLIST(LAST) = B2
463                   ELIST(MAXERR) = ERROR1
464                   ELIST(LAST) = ERROR2
465                ELSE
466                   ALIST(MAXERR) = A2
467                   ALIST(LAST) = A1
468                   BLIST(LAST) = B1
469                   RLIST(MAXERR) = AREA2
470                   RLIST(LAST) = AREA1
471                   ELIST(MAXERR) = ERROR2
472                   ELIST(LAST) = ERROR1
473                ENDIF
     C
     C    CHAMA DQSORT PARA MANTER A ORDEM DECRESCENTE NA LISTA DE ESTIMATIVA
     C       DE ERRO E SELECIONA O SUBINTERVALO COM NRMAX-ESIMA MAIOR
     C       ESTIMA DE ERRO. (PARA A PROXIMA BISSECAO).
     C
474                CALL DQSORT (LIMIT, LAST, MAXERR, ERRMAX, ELIST, IORD,
       X                NRMAX)
     C
     C    DESVIO PARA FORA DO CICLO
     C
475                IF (ERRSUM .LE. ERRBND) THEN
476                   GOTO 170
477                ENDIF
478                IF (IER .NE. 0) THEN
479                   GOTO 150
480                ENDIF
481                IF (LAST .NE. 2 .OR. .NOT. EXTALL) THEN
482                   IF (NOEXT) THEN
483                      GOTO 140
484                   ENDIF
485                   IF (EXTALL) THEN
486                      ERLARG = ERLARG - ERLAST
487                      IF (DABS (B1 - A1) .GT. SMALL)  THEN
```

```
488                        ERLARG = ERLARG + ERRO12
489                     ENDIF
490                     IF (EXTRAP) THEN
491                        GOTO 70
492                     ENDIF
      C
      C   VERIFICA SE O INTERVALO A SER BISSECCIONADO A SEGUIR E O MENOR.
      C
493                  ENDIF
494                  WIDTH = DABS (BLIST(MAXERR) - ALIST(MAXERR))
495                  IF (WIDTH .GT. SMALL) THEN
496                     GOTO 140
497                  ENDIF
498                  IF (.NOT. EXTALL) THEN
      C
      C   VERIFICA SE PODEMOS INICIAR COM EXTRAPOLACAO (PODEMOS FAZE-LO SE
      C      INTEGRAMOS SOBRE O PROXIMO INTERVALO USANDO A REGRA DE
      C      GAUSS-KONROD)
      C
499                     SMALL = SMALL * 5.0E-01
500                     IF (2.5E-01 * WIDTH * DOMEGA .GT. 2.0E+00) THEN
501                        GOTO 140
502                     ENDIF
503                     EXTALL = .TRUE.
504                     GOTO 130
505                  ENDIF
506                  EXTRAP = .TRUE.
507                  NRMAX = 2
508               70 CONTINUE
509                  IF (IERRO .NE. 3 .AND. ERLARG .GT. ERTEST) THEN
      C
      C   O MENOR INTERVALO TEM O MAIOR ERRO. ANTES DA BISSECCAO DIMINUI
      C      A SOMA DOS ERROS, SOBRE INTERVALOS MAIORES (ERLARG) E EXECUTA
      C      EXTRAPOLACAO
      C
510                     JUPBND = LAST
511                     IF (LAST .GT. (LIMIT / 2 + 2))  THEN
512                        JUPBND = LIMIT + 3 - LAST
513                     ENDIF
514                     ID = NRMAX
515                     DO 80 K = ID,JUPBND
516                        MAXERR = IORD(NRMAX)
517                        ERRMAX = ELIST(MAXERR)
518                        IF (DABS (BLIST(MAXERR) - ALIST(MAXERR)) .GT.
     X                     SMALL) THEN
519                           GOTO 140
520                        ENDIF
521                        NRMAX = NRMAX + 1
522               80    CONTINUE
      C
      C   EXECUTA EXTRAPOLACAO
      C
523                  ENDIF
524                  NUMRL2 = NUMRL2 + 1
525                  RLIST2(NUMRL2) = AREA
526                  IF (NUMRL2 .GE. 3) THEN
527                     CALL DQEXT (NUMRL2, RLIST2, RESEPS, ABSEPS, RES3LA,
     X                    NRES)
528                     KTMIN = KTMIN + 1
529                     IF (KTMIN .GT. 5 .AND. ABSERR .LT. 1.0E-03 *
```

```
      X                    ERRSUM)  THEN
530                        IER = 5
531                     ENDIF
532                     IF (ABSEPS .LT. ABSERR) THEN
533                        KTMIN = 0
534                        ABSERR = ABSEPS
535                        RESULT = RESEPS
536                        CORREC = ERLARG
537                        ERTEST = DMAX1 (EPSABS, EPSREL * DABS (RESEPS))
      C
      C   DESVIO PARA FORA DO CICLO
      C
538                        IF (ABSERR .LE. ERTEST) THEN
539                           GOTO 150
540                        ENDIF
      C
      C   PREPARA BISSECCAO DO MENOR INTERVALO
      C
541                     ENDIF
542                     IF (NUMRL2 .EQ. 1)  THEN
543                        NOEXT = .TRUE.
544                     ENDIF
545                     IF (IER .EQ. 5) THEN
546                        GOTO 150
547                     ENDIF
548                  ENDIF
549                  MAXERR = IORD(1)
550                  ERRMAX = ELIST(MAXERR)
551                  NRMAX = 1
552                  EXTRAP = .FALSE.
553                  SMALL = SMALL * 5.0E-01
554                  ERLARG = ERRSUM
555                  GOTO 140
556               ENDIF
557               SMALL = SMALL * 5.0E-01
558               NUMRL2 = NUMRL2 + 1
559               RLIST(NUMRL2) = AREA
560   130         CONTINUE
561               ERTEST = ERRBND
562               ERLARG = ERRSUM
563   140      CONTINUE
      C
      C   DEFINE O RESULTADO FINAL
      C
564   150      CONTINUE
565            IF (ABSERR .NE. OFLOW .AND. NRES .NE. 0) THEN
566               IF (IER + IERRO .NE. 0) THEN
567                  IF (IERRO .EQ. 3)  THEN
568                     ABSERR = ABSERR + CORREC
569                  ENDIF
570                  IF (IER .EQ. 0)  THEN
571                     IER = 3
572                  ENDIF
573                  IF (RESULT .EQ. 0.0E+00 .OR. AREA .EQ. 0.0E+00) THEN
574                     IF (ABSERR .GT. ERRSUM) THEN
575                        GOTO 170
576                     ENDIF
577                     IF (AREA .EQ. 0.0E+00) THEN
578                        GOTO 190
579                     ENDIF
```

```
580                     ELSE
581                        IF (ABSERR / DABS (RESULT) .GT. ERRSUM /
     X                        DABS (AREA)) THEN
582                        GOTO 170
583                        ENDIF
     C
     C   TESTA DIVERGENCIA
     C
584                     ENDIF
585                  ENDIF
586                  IF (KSGN .EQ. (-1) .AND.
     X                  DMAX1 (DABS (RESULT), DABS (AREA)) .LE. DEFABS *
     X                  1.0E-02) THEN
587                     GOTO 190
588                  ENDIF
589                  IF (1.0E-02 .GT. (RESULT / AREA) .OR. (RESULT /
     X                  AREA) .GT. 1.0E+02 .OR. ERRSUM .GE. DABS (AREA))  THEN
590                     IER = 6
591                  ENDIF
592                  GOTO 190
     C
     C   COMPUTA SOMA GLOBAL
     C
593               ENDIF
594   170         CONTINUE
595               RESULT = 0.0E+00
596               DO 180 K = 1,LAST
597                  RESULT = RESULT + RLIST(K)
598   180         CONTINUE
599               ABSERR = ERRSUM
600   190         CONTINUE
601               IF (IER .GT. 2)  THEN
602                  IER = IER - 1
603               ENDIF
604            ENDIF
605            IF (INTEGR .EQ. 2 .AND. OMEGA .LT. 0.0E+00)  THEN
606               RESULT = - RESULT
607            ENDIF
608         ENDIF
609         RETURN
610         END
     C
```

```
 1199       DATA DIVER(1), DIVER(2) / 0, 1017118720 /
 1200       DATA DMACH(5) /  3.010299956639812D+00 /
     C
 1201       IF(I .LT. 1 .OR. I .GT. 5) WRITE(STAR, 9000)
 1202  9000 FORMAT(32HERRO EM DMAQ. I FORA DOS LIMITES)
 1203       DMAQ = DMACH(I)
     C
 1204       RETURN
 1205       END
```

```
ompile time:                    30.54  Execution time:                02.36
ze of object code:              30320  Number of extensions:             1
ze of local data area(s):        9759  Number of warnings:                0
ze of global data area:         56609  Number of errors:                  0
ject/Dynamic bytes free:  92319/42534  Statements Executed:           10147
```

APÊNDICE IV

RESULTADOS OBTIDOS COM O PACOTE DQAGST DA BITAN

- NOME DO PACOTE: DQAGST

- NOME DA ROTINA EM FORTRAN-66 UTILIZADA PARA CONVERSÃO EM FORTRAN-77: DQAGS

- NÚMERO DE LINHAS DA ROTINA EM FORTRAN-66: 349

- TEMPO MÉDIO DE TRANSFORMAÇÃO DA ROTINA PARA O FORTRAN-77 GASTO PELO SISCO: 17.89 segundos

- NÚMERO DE LINHAS DA ROTINA EM FORTRAN-77: 418

	TEMPO DE COMPILACAO		TEMPO DE EXECUCAO	
ORDEM	PACOTE COM ROTINA EM FORTRAN-66	PACOTE COM ROTINA EM FORTRAN-77	PACOTE COM ROTINA EM FORTRAN-66	PACOTE COM ROTINA EM FORTRAN-77
1	10.38	11.20	1.60	1.59
2	10.49	12.13	1.59	1.59
3	11.09	11.69	1.58	1.59
4	11.20	12.68	1.59	1.59
5	10.22	16.30	1.59	1.59
6	10.54	11.74	1.65	1.59
7	10.65	11.74	1.61	1.59
8	11.20	11.91	1.59	1.59
9	11.15	12.46	1.59	1.59
10	10.65	12.30	1.59	1.59
MEDIA	10.76	12.42	1.60	1.59

Podemos observar que o tempo médio de compilação é menor para o pacote que utiliza a rotina escrita em FORTRAN-66, enquanto que o tempo médio de execução é praticamente o mesmo para as duas versões do pacote.

APÊNDICE V

LISTAGENS DO PACOTE DQAGST

LISTAGEM DO PACOTE DQAGST UTILIZANDO DQAGS EM FORTRAN-66

```
TFOR-77 V3.0 Copyright WATCOM Systems Inc. 1984,1988  92/06/12 21:46:16

tions: list,disk,xtype,terminal,extensions,warnings,check,arraycheck

     1        INTEGER IER, KEY, NEVAL
     2        DOUBLE PRECISION A, ABSERR, B, EPSABS, EPSREL, F, RESULT
     3        DOUBLE PRECISION DABS, ERRABS
     4        EXTERNAL F
       C
     5        A = 0.D0
     6        B = 1.D0
       C
     7        EPSABS = 0.D0
     8        EPSREL = 1.D-3
       C
     9        CALL DQAGS (F, A, B, EPSABS, EPSREL, RESULT,
             X             ABSERR, NEVAL, IER)
       C
    10        ERRABS = DABS (-4.D0 - RESULT)
    11        WRITE (6,900) RESULT, ABSERR, ERRABS, NEVAL, IER
    12    900 FORMAT( ' APROXIMACAO DA INTEGRAL =', D24.16//
             X          ' ESTIMATIVA DO ERRO ABSOLUTO =', D9.2//
             X          ' ERRO ABSOLUTO REAL =', D9.2//
             X          ' NUMERO DA AVALIACOES DO INTEGRANDO =', I5//
             X          ' CODIGO DE RETORNO =', I2)
    13        STOP
    14        END

    15        DOUBLE PRECISION FUNCTION F(X)
    16        DOUBLE PRECISION X
       C
    17        DOUBLE PRECISION DLOG, DSQRT
       C
    18        F = DLOG (X) / DSQRT (X)
    19        RETURN
    20        END


    21        SUBROUTINE DQAGS (F, A, B, EPSABS, EPSREL, RESULT, ABSERR,
             X                  NEVAL, IER)
       C
    22        INTEGER NEVAL, IER
    23        DOUBLE PRECISION F, A, B, EPSABS, EPSREL, RESULT, ABSERR
       C
       C      CALCULA UMA APROXIMACAO DE
       C           I = INTEGRAL DE F(X) SOBRE (A,B),
       C      ESPERANDO QUE O ERRO COMETIDO SATISFACA
       C           ABS (I - RESULT) .LE. MAX (EPSABS, EPSREL*ABS(I)).
       C      A ROTINA E CHAMADA POR DQAG. CONTUDO, PODE SER
       C      CHAMADO DIRETAMENTE PELO USUARIO.
       C
       C      PARAMETROS
       C
       C      NA CHAMADA
       C         F        DOUBLE PRECISION FUNCTION
       C                  E A FUNCAO INTEGRANDO DEFINIDA PELO USUARIO EM UM
       C                  SUBPROGRAMA FUNCAO. O NOME REAL DE F PRECISA SER
       C                  DECLARADO NUM EXTERNAL NO PROGRAMA ATIVADOR.
       C         A        DOUBLE PRECISION
```

```
C                LIMITE INFERIOR DO INTERVALO DE INTEGRACAO
C          B     DOUBLE PRECISION
C                LIMITE SUPERIOR DO INTERVALO DE INTEGRACAO
C          EPSABS DOUBLE PRECISION
C                TOLERANCIA DE ERRO ABSOLUTO SOLICITADA
C          EPSREL DOUBLE PRECISION
C                TOLERANCIA DE ERRO RELATIVO SOLICITADA. SE EPSABS E
C                EPSREL FOREM NEGATIVOS DQAGS SINALIZARA A OCORRENCIA
C                DE ERRO.
C          RESULT DOUBLE PRECISION
C                APENAS DECLARADO
C          ABSERR DOUBLE PRECISION
C                APENAS DECLARADO
C          NEVAL  INTEGER
C                APENAS DECLARADO
C          IER    INTEGER
C                APENAS DECLARADO
C
C      NO RETORNO
C          RESULT APROXIMACAO DA INTEGRAL
C          ABSERR ESTIMATIVA DO MODULO DO ERRO ABSOLUTO QUE DEVE
C                SER MAIOR OU IGUAL A ABS(I - RESULT).
C          NEVAL  O NUMERO DE AVALIACOES DO INTEGRANDO.
C          IER    SE O SEU VALOR FOR
C                 = 0 TERMINO NORMAL DA ROTINA. SUPOE-SE QUE A
C                     TOLERANCIA DE ERRO SOLICITADA FOI SATISFEITA
C                 = 1 O MAXIMO NUMERO SOLICITADO DE SUBDIVISOES DO
C                     INTERVALO FOI ATINGIDO. PODE-SE PERMITIR MAIS
C                     SUBDIVISOES AUMENTANDO-SE O VALOR DE LIMIT
C                     (PRECISA AJUSTAR DIMENSOES). SE ESSE AUMENTO NAO
C                     ACARRETAR MELHORIA NO RESULTADO ACONSELHA-SE
C                     ANALISAR O INTEGRANDO A FIM DE AVERIGUAR AS
C                     DIFICULDADES DE INTEGRACAO. SE A DIFICULDADE PUDER
C                     SER LOCALIZADA (POR EXEMPLO, SINGULARIDADE, DES-
C                     CONTINUIDADE DENTRO DO INTERVALO) VALERA A PENA
C                     SUBDIVIDIR O INTERVALO NESSE PONTO E CHAMAR O
C                     DQAGS EM CADA SUBINTERVALO. SE POSSIVEL, ESCOLHER
C                     UMA ROTINA ESPECIALIZADA QUE SEJA CAPAZ DE CONTOR-
C                     NAR A DIFICULDADE DETETADA.
C                 = 2 FOI DETETADA A OCORRENCIA DE ERRO DE ARREDONDAMENTO
C                     QUE NAO PERMITE ATINGIR A TOLERANCIA DE ERRO
C                     SOLICITADA.
C                 = 3 UM MAU COMPORTAMENTO EXTREMO DO INTEGRANDO OCORRE
C                     EM ALGUNS PONTOS NO INTERVALO DE INTEGRACAO.
C                 = 4 O ALGORITMO NAO CONVERGE. ERRO DE ARREDONDAMENTO
C                     E DETETADO NA TABELA DE EXTRAPOLACAO. PRESUME-SE
C                     QUE A TOLERANCIA SOLICITADA NAO PODE SER ALCANCADA
C                     DEVIDO AO ERRO DE ARREDONDAMENTO NA TABELA E QUE
C                     O RESULTADO FORNECIDO E O MELHOR QUE SE PODE OBTER.
C                 = 5 A INTEGRAL E PROVAVELMENTE DIVERGENTE OU LENTAMEN-
C                     TE CONVERGENTE. DEVE-SE OBSERVAR QUE A DIVERGENCIA
C                     PODE OCORRER COM QUALQUER OUTRO VALOR DE IER DIFE-
C                     RENTE DE ZERO.
C                 = 6 HOUVE ERRO NOS DADOS DE ENTRADA (EPSABS  E EPSREL
C                     NEGATIVOS). OS VALORES DE RESULT, ABSERR E NEVAL
C                     SERAO NULOS.
C
C      SUBPROGRAMAS UTILIZADOS
C          DO FORTRAN - DABS, DMAX1
C          DA BIBLIOTECA - DQEXT, DQK21, DQSORT
```

```
C         DO USUARIO - F
C
C       REFERENCIA
C          PIESSENS, DE DONCKER-KAPENGA, UBERHUBER, KAHANER, QUADPACK - A
C            SUBROUTINE PACKAGE FOR AUTOMATIC INTEGRATION, SPRINGER-
C            VERLAG, BERLIN HEIDELBERG N. YORK TOKYO, 1983
C
C                    UNIVERSIDADE FEDERAL DA PARAIBA
C                    DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C                    PROJETO BITAN
C                    MARIO T. HATTORI  ABR/90
C
24        INTEGER ID, IERRO, IORD(500), IROFF1, IROFF2, IROFF3,
         X        JUPBND, K, KSGN, KTMIN, LAST, LIMIT, MAXERR,
         X        NRES, NRMAX, NUMRL2
25        LOGICAL EXTRAP, NOEXT
26        DOUBLE PRECISION ABSEPS, ALIST(500), AREA, AREA1, AREA12, AREA2,
         X                 A1, A2, BLIST(500), B1, B2, CORREC, DEFAB1,
         X                 DEFAB2, DEFABS, DRES, ELIST(500), EPMACH, ERLARG,
         X                 ERLAST, ERRBND, ERRMAX, ERROR1, ERR012,
         X                 ERROR2, ERRSUM, ERTEST, OFLOW, RESABS, RESEPS,
         X                 RES3LA(3), RLIST(500), RLIST2(52), SMALL, UFLOW
27        DOUBLE PRECISION DABS, DMAX1, DMAQ
28        EXTERNAL F
29        DATA LIMIT /500/
C
30        EPMACH = DMAQ (4)
31        OFLOW = DMAQ (2)
32        UFLOW = DMAQ (1)
C
C     TESTA VALIDADE DOS PARAMETROS
C
33        IER = 0
34        NEVAL = 0
35        LAST = 0
36        RESULT = 0.0E+00
37        ABSERR = 0.0E+00
38        ALIST(1) = A
39        BLIST(1) = B
40        RLIST(1) = 0.0E+00
41        ELIST(1) = 0.0E+00
42        IF (EPSABS .LT. 0.0E+00 .AND. EPSREL .LT. 0.0E+00) IER = 6
43        IF (IER .EQ. 6) GOTO 999
C
C     PRIMEIRA APROXIMACAO DA INTEGRAL
C
44        IERRO = 0
45        CALL DQK21 (F, A, B , RESULT, ABSERR, DEFABS, RESABS)
C
C     TESTA EXATIDAO
C
46        DRES = DABS (RESULT)
47        ERRBND = DMAX1 (EPSABS, EPSREL * DRES)
48        LAST = 1
49        RLIST(1) = RESULT
50        ELIST(1) = ABSERR
51        IORD(1) = 1
52        IF (ABSERR .LE. 1.0E+02 * EPMACH * DEFABS .AND.
         X    ABSERR .GT. ERRBND) IER = 2
53        IF (LIMIT .EQ. 1) IER = 1
```

```
54          IF (IER .NE. 0 .OR. (ABSERR .LE. ERRBND .AND. ABSERR .NE. RESABS)
           X    .OR. ABSERR .EQ. 0.0E+00) GOTO 140
      C
      C   DEFINE CONDICOES INICIAIS
      C
55          RLIST2(1) = RESULT
56          ERRMAX = ABSERR
57          MAXERR = 1
58          AREA = RESULT
59          ERRSUM = ABSERR
60          ABSERR = OFLOW
61          NRMAX = 1
62          NRES = 0
63          NUMRL2 = 2
64          KTMIN = 0
65          EXTRAP = .FALSE.
66          NOEXT = .FALSE.
67          IROFF1 = 0
68          IROFF2 = 0
69          IROFF3 = 0
70          KSGN = -1
71          IF (DRES .GE. (1.0E+00 - 5.0E+01 * EPMACH) * DEFABS) KSGN = 1
      C
      C   CICLO PRINCIPAL
      C
72          DO 90 LAST = 2,LIMIT
      C
      C   BISSECCIONA O SUBINTERVALO COM A ESTIMATIVA DE ERRO MAIOR
      C
73            A1 = ALIST(MAXERR)
74            B1 = 5.0E-01 * (ALIST(MAXERR) + BLIST(MAXERR))
75            A2 = B1
76            B2 = BLIST(MAXERR)
77            ERLAST = ERRMAX
78            CALL DQK21 (F, A1, B1, AREA1, ERROR1, RESABS, DEFAB1)
79            CALL DQK21 (F, A2, B2, AREA2, ERROR2, RESABS, DEFAB2)
      C
      C   MELHORA AS APROXIMACOES ANTERIORES DA INTEGRAL E DO ERRO E
      C      FAZ TESTE DE EXATIDAO
      C
80            AREA12 = AREA1 + AREA2
81            ERRO12 = ERROR1 + ERROR2
82            ERRSUM = ERRSUM + ERRO12 - ERRMAX
83            AREA = AREA + AREA12 - RLIST(MAXERR)
      C
84            IF (DEFAB1 .EQ. ERROR1 .OR. DEFAB2 .EQ. ERROR2) GOTO 15
85            IF (DABS (RLIST(MAXERR) - AREA12) .GT. 1.0E-05 * DABS (AREA12)
           X       .OR. ERRO12 .LT. 9.9E-01 * ERRMAX) GOTO 10
86            IF (EXTRAP) IROFF2 = IROFF2 + 1
87            IF (.NOT. EXTRAP) IROFF1 = IROFF1 + 1
88        10  CONTINUE
89            IF (LAST .GT. 10 .AND. ERRO12 .GT. ERRMAX) IROFF3 = IROFF3 + 1
90        15  CONTINUE
91            RLIST(MAXERR) = AREA1
92            RLIST(LAST) = AREA2
93            ERRBND = DMAX1 (EPSABS, EPSREL * DABS (AREA))
      C
      C   TESTA ERRO DE ARREDONDAMENTO E EVENTUALMENTE SINALIZA ERRO
      C
94            IF (IROFF1 + IROFF2 .GE. 10 .OR. IROFF3 .GE. 20) IER = 2
```

```
 95          IF (IROFF2 .GE. 5) IERRO = 3
     C
     C    NUMERO DE SUBINTERVALOS ATINGIU LIMIT
     C
 96          IF (LAST .EQ. LIMIT) IER = 1
     C
     C    INTEGRANDO TEM MAU COMPORTAMENTO EM UM PONTO DO INTERVALO
     C       DE INTEGRACAO
     C
 97          IF (DMAX1 (DABS (A1), DABS (B2)) .LE. (1.0E+00 + 1.0E+03 *
        X       EPMACH) * (DABS (A2) + 1.0E+03 * UFLOW)) IER = 4
     C
     C    ANEXA OS INTERVALOS RECEM-CRIADOS A LISTA
     C
 98          IF (ERROR2 .GT. ERROR1) GOTO 20
 99          ALIST(LAST) = A2
100          BLIST(MAXERR) = B1
101          BLIST(LAST) = B2
102          ELIST(MAXERR) = ERROR1
103          ELIST(LAST) = ERROR2
104          GOTO 30
105     20   CONTINUE
106          ALIST(MAXERR) = A2
107          ALIST(LAST) = A1
108          BLIST(LAST) = B1
109          RLIST(MAXERR) = AREA2
110          RLIST(LAST) = AREA1
111          ELIST(MAXERR) = ERROR2
112          ELIST(LAST) = ERROR1
113     30   CONTINUE
     C
     C    MANTEM A ORDENACAO DESCENDENTE NA LISTA DE ESTIMATIVAS DE ERRO
     C       E SELECIONA O SUBINTERVALO COM ESTIMATIVA DE ERRO MAIOR
     C       (PARA SER BISSECCIONADA EM SEGUIDA)
     C
114          CALL DQSORT (LIMIT, LAST, MAXERR, ERRMAX, ELIST, IORD, NRMAX)
     C
     C    DESVIO PARA FORA DO CICLO
     C
115          IF (ERRSUM .LE. ERRBND) GOTO 115
116          IF (IER .NE. 0) GOTO 100
117          IF (LAST .EQ. 2) GOTO 80
118          IF (NOEXT) GOTO 90
119          ERLARG = ERLARG - ERLAST
120          IF (DABS (B1 - A1) .GT. SMALL) ERLARG = ERLARG + ERRO12
121          IF (EXTRAP) GOTO 40
     C
     C    TESTA SE O PROXIMO INTERVALO A SER BISSECCIONADO E O MENOR
     C
122          IF (DABS (BLIST(MAXERR) - ALIST(MAXERR)) .GT. SMALL) GOTO 90
123          EXTRAP = .TRUE.
124          NRMAX = 2
125     40   CONTINUE
126          IF (IERRO .EQ. 3 .OR. ERLARG .LT. ERTEST) GOTO 60
     C
     C    O MENOR INTERVALO TEM O MAIOR ERRO. ANTES DE BISSECCIONAR DIMINUI
     C       A SOMA DOS ERROS SOBRE OS INTERVALOS MAIORES (ERLARG)
     C       E EXECUTA EXTRAPOLACAO.
     C
127          ID = NRMAX
```

```
128           JUPBND = LAST
129           IF (LAST .GT. (2 + LIMIT / 2)) JUPBND = LIMIT + 3 - LAST
130           DO 50 K = ID,JUPBND
131             MAXERR = IORD(NRMAX)
132             ERRMAX = ELIST(MAXERR)
      C
      C    DESVIO PARA FORA DO CICLO
      C
133             IF (DABS (BLIST(MAXERR) - ALIST(MAXERR)) .GT. SMALL) GOTO 90
134             NRMAX = NRMAX + 1
135        50   CONTINUE
      C
      C    EXECUTA EXTRAPOLACAO
      C
136        60   CONTINUE
137             NUMRL2 = NUMRL2 + 1
138             RLIST2(NUMRL2) = AREA
139             CALL DQEXT (NUMRL2, RLIST2, RESEPS, ABSEPS, RES3LA, NRES)
140             KTMIN = KTMIN + 1
141             IF (KTMIN .GT. 5 .AND. ABSERR .LT. 1.0E-03 * ERRSUM) IER = 5
142             IF (ABSEPS .GE. ABSERR) GOTO 70
143             KTMIN = 0
144             ABSERR = ABSEPS
145             RESULT = RESEPS
146             CORREC = ERLARG
147             ERTEST = DMAX1 (EPSABS, EPSREL * DABS (RESEPS))
      C
      C    DESVIO PARA FORA DO CICLO
      C
148             IF (ABSERR .LE. ERTEST) GOTO 100
      C
      C    PREPARA BISSECCAO DO MENOR INTERVALO.
      C
149        70   CONTINUE
150             IF (NUMRL2 .EQ. 1) NOEXT = .TRUE.
151             IF (IER .EQ. 5) GOTO 100
152             MAXERR = IORD(1)
153             ERRMAX = ELIST(MAXERR)
154             NRMAX = 1
155             EXTRAP = .FALSE.
156             SMALL = SMALL * 5.0E-01
157             ERLARG = ERRSUM
158             GOTO 90
159        80   CONTINUE
160             SMALL = DABS (B - A) * 3.75E-01
161             ERLARG = ERRSUM
162             ERTEST = ERRBND
163             RLIST2(2) = AREA
164        90 CONTINUE
      C
      C    COMPUTA O RESULTADO FINAL
      C
165       100 CONTINUE
166           IF (ABSERR .EQ. OFLOW) GOTO 115
167           IF (IER + IERRO .EQ. 0) GOTO 110
168           IF (IERRO .EQ. 3) ABSERR = ABSERR + CORREC
169           IF (IER .EQ. 0) IER = 3
170           IF (RESULT .NE. 0.0E+00 .AND. AREA .NE. 0.0E+00) GOTO 105
171           IF (ABSERR .GT. ERRSUM) GOTO 115
172           IF (AREA .EQ. 0.0E+00) GOTO 130
```

```
      GOTO 110
  105 CONTINUE
      IF (ABSERR / DABS (RESULT) .GT. ERRSUM / DABS (AREA)) GOTO 115
C
C   TESTA DIVERGENCIA
C
  110 CONTINUE
      IF (KSGN .EQ. (-1) .AND. DMAX1 (DABS (RESULT), DABS (AREA)) .LE.
     X    DEFABS * 1.0E-02) GOTO 130
      IF (1.0E-02 .GT. (RESULT / AREA) .OR. (RESULT / AREA) .GT. 1.0E+02
     X    .OR. ERRSUM .GT. DABS (AREA)) IER = 6
      GOTO 130
  115 CONTINUE
C
C   COMPUTA INTEGRAL
C
      RESULT = 0.0E+00
      DO 120 K = 1,LAST
        RESULT = RESULT + RLIST(K)
  120 CONTINUE
      ABSERR = ERRSUM
  130 CONTINUE
      IF (IER .GT. 2) IER = IER - 1
  140 CONTINUE
      NEVAL = 42 * LAST - 21
  999 CONTINUE
      RETURN
      END



      SUBROUTINE DQK21 (F, A, B, RESULT, ABSERR, RESABS, RESASC)
C
      DOUBLE PRECISION F, A, B, RESULT, ABSERR, RESABS, RESASC
C
C     COMPUTA I = INTEGRAL DE F SOBRE (A,B) COM ESTIMATIVA DE ERRO
C               J = INTEGRAL DE ABS(F) SOBRE (A,B)
C
C     PARAMETROS
C     NA CHAMADA
C        F        DOUBLE PRECISION FUNCTION
C                 SUBPROGRAMA FUNCAO QUE DEFINE A FUNCAO INTEGRANDO F(X).
C                 O NOME REAL DE F PRECISA SER DECLARADO NUM EXTERNAL NO
C                 PROGRAMA ATIVADOR
C        A        DOUBLE PRECISION
C                 O LIMITE INFERIOR DO INTERVALO DE INTEGRACAO
C        B        DOUBLE PRECISION
C                 O LIMITE SUPERIOR DO INTERVALO DE INTEGRACAO
C        RESULT   DOUBLE PRECISION
C                 APENAS DECLARADO
C        ABSERR   DOUBLE PRECISION
C                 APENAS DECLARADO
C        RESABS   DOUBLE PRECISION
C                 APENAS DECLARADO
C        RESASC   DOUBLE PRECISION
C                 APENAS DECLARADO
C     NO RETORNO
C        RESULT   APROXIMACAO DA INTEGRAL I. RESULT E COMPUTADO USANDO A
C                 REGRA DE 21 PONTOS DE KONROD, OBTIDA PELA ADICAO OTIMA
C                 DE ABCISSAS A REGRA DE GAUSS DE 10 PONTOS.
```

```
C          ABSERR  ESTIMATIVA DO ERRO ABSOLUTO, QUE NAO DEVE EXCEDER
C                  ABS (I - RESULT)
C          RESABS  APROXIMACAO DA INTEGRAL J
C          RESASC  APROXIMACAO DA INTEGRAL DE ABS (F - I/(B -A)) SOBRE
C                  (A,B)
C
C      SUBPROGRAMAS UTILIZADOS
C          DO FORTRAN - DABS, DMAX1, DMIN1
C          DA BIBLIOTECA - DMAQ
C          DO USUARIO - F
C
C      REFERENCIA
C          PIESSENS, DE DONCKER-KAPENGA, UBERHUBER, KAHANER, QUADPACK - A
C            SUBROUTINE PACKAGE FOR AUTOMATIC INTEGRATION, SPRINGER-VERLAG
C            BERLIN HEIDELBERG N. YORK TOKYO, 1983
C
C                  UNIVERSIDADE FEDERAL DA PARAIBA
C                  DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C                  PROJETO BITAN
C                  MARIO T. HATTORI  ABR/90
C
195        INTEGER J, JTW, JTWM1
196        DOUBLE PRECISION ABSC, CENTR, DHLGTH, EPMACH, FC, FSUM, FVAL1,
          X                 FVAL2, FV1(10), FV2(10), HLGTH, OFLOW, ONE,
          X                 ONEP5, RESG, RESK, RESKH, TWO, UFLOW, WG(5),
          X                 WGK(11), XGK(11), ZERO
197        DOUBLE PRECISION DABS, DMAQ, DMAX1, DMIN1
C
C      AS ABCISSAS E PESOS SAO DADOS PARA O INTERVALO (-1,1). POR CAUSA
C      DA SIMETRIA SOMENTE AS ABCISSAS POSITIVAS E OS CORRESPONDENTES
C      PESOS SAO DADOS.
C
C      XGK    ABCISSAS DA REGRA DE 21 PONTOS DE KONROD
C             XGK(2), XGK(4), ... ABCISSAS DA REGRA DE 10 PONTOS DE GAUSS
C
C             XGK(1), XGK(3), ... ABCISSAS QUE FORAM ADICIONADAS
C             OTIMAMENTE A REGRA DE 10 PONTOS DE GAUSS.
C      WGK    PESOS DA REGRA DE 21 PONTOS DE KONROD
C      WG     PESOS DA REGRA DE 10 PONTOS DE GAUSS
C
198        DATA XGK /
          X           9.956571630258081D-01,  9.739065285171717D-01,
          X           9.301574913557082D-01,  8.650633666889845D-01,
          X           7.808177265864169D-01,  6.794095682990244D-01,
          X           5.627571346686047D-01,  4.333953941292472D-01,
          X           2.943928627014602D-01,  1.488743389816312D-01,
          X           0.000000000000000D+00 /
199        DATA WGK /
          X           1.169463886737187D-02,  3.255816230796473D-02,
          X           5.475589657435200D-02,  7.503967481091995D-02,
          X           9.312545458369761D-02,  1.093871588022976D-01,
          X           1.234919762620659D-01,  1.347092173114733D-01,
          X           1.427759385770601D-01,  1.477391049013385D-01,
          X           1.494455540029169D-01 /
200        DATA WG /
          X           6.667134430868814D-02,  1.494513491505806D-01,
          X           2.190863625159820D-01,  2.692667193099964D-01,
          X           2.955242247147529D-01 /
C
201        DATA ZERO /0.D0/, ONE /1.D0/, ONEP5 /1.5D0/, TWO /2.D0/
```

```
      C
  202       EPMACH = DMAQ (4)
  203       UFLOW = DMAQ (1)
  204       OFLOW = DMAQ (2)
      C
  205       CENTR = (A + B) / TWO
  206       HLGTH = (B - A) / TWO
  207       DHLGTH = DABS (HLGTH)
      C
      C     COMPUTA A APROXIMACAO DA INTEGRAL PELA REGRA DE 21 PONTOS DE
      C     KONROD E FAZ UMA ESTIMATIVA DO ERRO ABSOLUTO
      C
  208       RESG = ZERO
  209       FC = F (CENTR)
  210       RESK = FC * WGK(11)
  211       RESABS = DABS (RESK)
      C
  212       DO 20 J = 1,5
  213          JTW = 2 * J
  214          ABSC = HLGTH * XGK(JTW)
  215          FVAL1 = F (CENTR - ABSC)
  216          FVAL2 = F (CENTR + ABSC)
  217          FV1(JTW) = FVAL1
  218          FV2(JTW) = FVAL2
  219          FSUM = FVAL1 + FVAL2
  220          RESG = RESG + WG(J) * FSUM
  221          RESK = RESK + WGK(JTW) * FSUM
  222          RESABS = RESABS + WGK(JTW) * (DABS (FVAL1) + DABS (FVAL2))
  223    20 CONTINUE
      C
  224       DO 40 J = 1,5
  225          JTWM1 = 2 * J - 1
  226          ABSC = HLGTH * XGK(JTWM1)
  227          FVAL1 = F (CENTR - ABSC)
  228          FVAL2 = F (CENTR + ABSC)
  229          FV1(JTWM1) = FVAL1
  230          FV2(JTWM1) = FVAL2
  231          FSUM = FVAL1 + FVAL2
  232          RESK = RESK + WGK(JTWM1) * FSUM
  233          RESABS = RESABS + WGK(JTWM1) * (DABS (FVAL1) + DABS (FVAL2))
  234    40 CONTINUE
      C
  235       RESKH = RESK / TWO
  236       RESASC = WGK(11) * DABS (FC - RESKH)
      C
  237       DO 60 J = 1,10
  238          RESASC = RESASC + WGK(J) * (DABS (FV1(J) - RESKH) +
     X                                    DABS (FV2(J) - RESKH))
  239    60 CONTINUE
      C
  240       RESULT = RESK * HLGTH
  241       RESABS = RESABS * DHLGTH
  242       RESASC = RESASC * DHLGTH
  243       ABSERR = DABS ((RESK - RESG) * HLGTH)
  244       IF (RESASC .NE. ZERO .AND. ABSERR .NE. ZERO)  ABSERR =
     X     RESASC * DMIN1 (ONE, (200.D0 * ABSERR / RESASC)**ONEP5)
  245       IF (RESABS .GT. UFLOW / (50.D0 * EPMACH)) ABSERR =
     X     DMAX1 ((EPMACH * 50.D0) * RESABS, ABSERR)
      C
  246       RETURN
```

```
247         END

248         SUBROUTINE DQSORT (LIMIT, LAST, MAXERR, ERMAX, ELIST, IORD,
           X                   NRMAX)
      C
249         INTEGER LIMIT, LAST, MAXERR, IORD(LIMIT), NRMAX
250         DOUBLE PRECISION ERMAX, ELIST(LIMIT)
      C
      C     MANTEM A ORDENACAO DESCENDENTE NA LISTA DE ESTIMATIVAS DE ERRO
      C     LOCAIS RESULTANTES DO PROCESSO DE SUBDIVISAO DO INTERVALO. A
      C     CADA CHAMADA DUAS ESTIMATIVAS DE ERRO SAO INSERIDAS USANDO
      C     PESQUISA LINEAR DO FIM DA LISTA PARA O INICIO PARA INSERIR  A
      C     MAIOR ESTIMATIVA DO ERRO E DO INICIO PARA O FIM DA LISTA PARA
      C     INSERIR A MENOR ESTIMATIVA DO ERRO.
      C
      C     PARAMETROS
      C
      C     NA CHAMADA
      C        LIMIT   INTEGER
      C                O MAXIMO NUMERO DE ESTIMATIVAS DE ERRO QUE
      C                PODE SER ARMAZENADO NA LISTA.
      C        LAST    INTEGER
      C                O NUMERO DE ESTIMATIVAS DE ERRO CORRENTEMENTE NA LISTA
      C                (DEVE SER GERENCIADA PELO PROGRAMA ATIVADOR,
      C                COMECANDO COM VALOR INICIAL 2)
      C        MAXERR  INTEGER
      C                APENAS DECLARADO
      C        ERMAX   DOUBLE PRECISION
      C                APENAS DECLARADO
      C        ELIST   DOUBLE PRECISION (LIMIT)
      C                A LISTA DEVE SER CONSTRUIDA PELO PROGRAMA ATIVADOR
      C                EM GERAL ADOTA-SE ELIST(1) = 0.D0
      C        IORD    INTEGER (LIMIT)
      C                APENAS DECLARADO
      C        NRMAX   INTEGER
      C                NA PRIMEIRA CHAMADA DEVE TER VALOR 1
      C
      C     NO RETORNO
      C        LAST    O NUMERO DE ESTIMATIVAS DE ERRO CORRENTEMENTE NA LISTA.
      C        MAXERR  APONTA PARA NRMAX-ESIMA MAIOR ESTIMATIVA DE ERRO
      C                CORRENTEMENTE NA LISTA.
      C        ERMAX   A NRMAX-ESIMA MAIOR ESTIMATIVA DE ERRO.
      C                ERMAX = ELIST(MAXERR)
      C        ELIST   CONTEM AS ESTIMATIVAS DE ERRO
      C        IORD    OS PRIMEIROS K ELEMENTOS CONTEM APONTADORES PARA AS
      C                ESTIMATIVAS DE ERRO TAL QUE ELIST(IORD(1)), ... ,
      C                ELIST(IORD(K)) FORMA UMA SEQUENCIA DECRESCENTE COM
      C                K = LAST SE LAST .LE. (LIMIT / 2 + 2) E
      C                K = LIMIT + 1 - LAST, CASO CONTRARIO.
      C
      C     SUBPROGRAMAS UTILIZADOS
      C        DO FORTRAN - DABS, DMAX1, DMIN1
      C        DA BIBLIOTECA - DMAQ
      C        DO USUARIO - F
      C
      C     REFERENCIA
      C        PIESSENS, DE DONCKER-KAPENGA, UBERHUBER, KAHANER, QUADPACK - A
      C          SUBROUTINE PACKAGE FOR AUTOMATIC INTEGRATION, SPRINGER-
      C          VERLAG, BERLIN HEIDELBERG N. YORK TOKYO, 1983
      C
```

```
C                UNIVERSIDADE FEDERAL DA PARAIBA
C                DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C                PROJETO BITAN
C                MARIO T. HATTORI  ABR/90
C
      INTEGER I, IBEG, IDO, ISUCC, J, JBND, JUPBN, K
      DOUBLE PRECISION ERRMAX, ERRMIN
C
C     VERIFICA SE A LISTA CONTEM MAIS DE DUAS ESTIMATIVAS DE ERRO
C
      IF (LAST .LE. 2) THEN
         IORD(1) = 1
         IORD(2) = 2
         MAXERR = IORD(NRMAX)
         ERMAX = ELIST(MAXERR)
         RETURN
      ENDIF
C
C     ESTA PARTE DA ROTINA SO SERA EXECUTADA SE, DEVIDO A UM INTEGRANDO
C     DIFICIL, A SUBDIVISAO AUMENTOU A ESTIMATIVA DE ERRO. NO CASO
C     NORMAL O PROCEDIMENTO DE INSERCAO DEVE COMECAR DEPOIS DA
C     NRMAX-ESIMA MAIOR ESTIMATIVA DE ERRO.
C
      ERRMAX = ELIST(MAXERR)
      IF (NRMAX .NE. 1) THEN
         IDO = NRMAX - 1
         DO 20 I = 1,IDO
            ISUCC = IORD(NRMAX - 1)
            IF (ERRMAX .LE. ELIST(ISUCC)) GOTO 30
            IORD(NRMAX) = ISUCC
            NRMAX = NRMAX - 1
   20    CONTINUE
      ENDIF
   30 CONTINUE
C
C     COMPUTA O NUMERO DE ELEMENTOS NA LISTA PARA SER MANTIDO EM
C     ORDEM DECRESCENTE. ESSE NUMERO DEPENDE DO NUMERO DE SUBDIVISOES
C     AINDA PERMITIDO.
C
      JUPBN = LAST
      IF (LAST .GT. (LIMIT / 2 + 2)) JUPBN = LIMIT + 3 - LAST
      ERRMIN = ELIST(LAST)
C
C     INSERE ERRMAX PERCORRENDO A LISTA DO FIM PARA O INICIO,
C     COMECANDO AS COMPARACOES A PARTIR DO ELEMENTO
C     ELIST(IORD(NRMAX+1))
C
      JBND = JUPBN - 1
      IBEG = NRMAX + 1
      IF (IBEG .LE. JBND) THEN
         DO 40 I = IBEG,JBND
            ISUCC = IORD(I)
            IF (ERRMAX .GE. ELIST(ISUCC)) GOTO 60
            IORD(I-1) = ISUCC
   40    CONTINUE
      ENDIF
      IORD(JBND) = MAXERR
      IORD(JUPBN) = LAST
      MAXERR = IORD(NRMAX)
      ERMAX = ELIST(MAXERR)
```

```
287         RETURN
      C
288      60 CONTINUE
      C
      C     INSERE ERRMIN PERCORRENDO A LISTA DO INICIO
      C
289         IORD(I-1) = MAXERR
290         K = JBND
291         DO 70 J = I,JBND
292            ISUCC = IORD(K)
293            IF (ERRMIN .LT. ELIST(ISUCC)) GOTO 80
294            IORD(K+1) = ISUCC
295            K = K - 1
296      70 CONTINUE
297         IORD(I) = LAST
298         MAXERR = IORD(NRMAX)
299         ERMAX = ELIST(MAXERR)
300         RETURN
      C
301      80 CONTINUE
302         IORD(K+1) = LAST
303         MAXERR = IORD(NRMAX)
304         ERMAX = ELIST(MAXERR)
305         RETURN
306         END

307         SUBROUTINE D@EXT (N, EPSTAB, RESULT, ABSERR, RES3LA, NRES)
      C
308         INTEGER N, NRES
309         DOUBLE PRECISION EPSTAB(52), RESULT, ABSERR, RES3LA(3)
      C
      C     DETERMINA O LIMITE DE UMA DADA SEQUENCIA DE APROXIMACOES USANDO O
      C     ALGORITMO DE EPSILON DEVIDO A P. WYNN.
      C     UMA ESTIMATIVA DO ERRO ABSOLUTO TAMBEM E FORNECIDO.
      C     A TABELA CONDENSADA DE EPSILON E COMPUTADA. SOMENTE OS ELEMENTOS
      C     NECESSARIOS PARA A COMPUTACAO DA PROXIMA DIAGONAL SAO PRESERVADOS.
      C
      C     PARAMETROS
      C
      C     NA CHAMADA
      C        N        INTEGER
      C                 EPSTAB(N) CONTEM O NOVO ELEMENTO NA PRIMEIRA COLUNA DA
      C                 TABELA DE EPSILON.
      C        EPSTAB   DOUBLE PRECISION(52)
      C                 APENAS DECLARADO. USADO INTERNAMENTE.
      C        RESULT   DOUBLE PRECISION
      C                 APENAS DECLARADO
      C        ABSERR   DOUBLE PRECISION
      C                 APENAS DECLARADO
      C        RES3LA   DOUBLE PRECISION (3)
      C                 APENAS DECLARADO
      C        NRES     INTEGER
      C                 NUMERO DE CHAMADAS DA SUBROTINA. DEVE SER ZERO NA
      C                 PRIMEIRA CHAMADA.
      C
      C     NO RETORNO
      C        RESULT   APROXIMACAO DA INTEGRAL
      C        ABSERR   ESTIMATIVA DO ERRO ABSOLUTO CALCULADO A PARTIR DE
      C                 RESULT E 3 RESULTADOS ANTERIORES
      C        RES3LA   CONTEM OS 3 ULTIMOS RESULTADOS.
```

```
C
C     SUBPROGRAMAS UTILIZADOS
C        DO FORTRAN - DABS, DMAX1
C        DA BIBLIOTECA - DMAQ
C
C     REFERENCIAS
C      1.PIESSENS, DE DONCKER-KAPENGA, UBERHUBER, KAHANER, QUADPACK - A
C          SUBROUTINE PACKAGE FOR AUTOMATIC INTEGRATION, SPRINGER-
C          VERLAG, BERLIN HEIDELBERG N. YORK TOKYO, 1983
C      2.WYNN, ON A DEVICE FOR COMPUTING THE E (S ) TRANSFORMATION,
C                                             M  M
C          MTAC 10, PP. 91-96.
C
C                UNIVERSIDADE FEDERAL DA PARAIBA
C                DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C                PROJETO BITAN
C                MARIO T. HATTORI  ABR/90
C
310       INTEGER I, IB, IB2, IE, INDX, K1, K2, K3, LIMEXP, NEWELM, NUM
311       DOUBLE PRECISION DELTA1, DELTA2, DELTA3, EPMACH, EPSINF,
         X                 ERROR, ERR1, ERR2, ERR3, E0, E1, E1ABS,
         X                 E2, E3, OFLOW, ONE, P5, RES, SS, TENM4,
         X                 TOL1, TOL2, TOL3, UFLOW
312       DOUBLE PRECISION DABS, DMAQ, DMAX1
313       DATA TENM4 /1.D-4/, P5 /.5D0/, ONE /1.D0/
C
314       UFLOW = DMAQ (1)
315       OFLOW = DMAQ (2)
316       EPMACH = DMAQ (4)
C
317       NRES = NRES + 1
318       ABSERR = OFLOW
319       RESULT = EPSTAB(N)
320       IF (N .LT. 3) THEN
321          ABSERR = DMAX1 (ABSERR , P5 * EPMACH * DABS (RESULT))
322          RETURN
323       ENDIF
324       LIMEXP = 0
325       EPSTAB(N+2) = EPSTAB(N)
326       NEWELM = (N - 1) / 2
327       EPSTAB(N) = OFLOW
328       NUM = N
329       K1 = N
330       DO 40 I = 1,NEWELM
331          K2 = K1 - 1
332          K3 = K1 - 2
333          RES = EPSTAB(K1+2)
334          E0 = EPSTAB(K3)
335          E1 = EPSTAB(K2)
336          E2 = RES
337          E1ABS = DABS (E1)
338          DELTA2 = E2 - E1
339          ERR2 = DABS (DELTA2)
340          TOL2 = DMAX1 (DABS (E2), E1ABS) * EPMACH
341          DELTA3 = E1 - E0
342          ERR3 = DABS (DELTA3)
343          TOL3 = DMAX1 (E1ABS, DABS (E0)) * EPMACH
344          IF (ERR2 .LE. TOL2 .AND. ERR3 .LE. TOL3) THEN
C
C             SE E0, E1 E E2 SAO IGUAIS DENTRO DA PRECISAO DA
```

```
C            DA MAQUINA, ASSUME-SE QUE HOUVE CONVERGENCIA.
C            RESULT = E2
C            ABSERR = ABS (E1 - E0) + ABS (E2 - E1)
C
345             RESULT = RES
346             ABSERR = ERR2 + ERR3
347             ABSERR = DMAX1 (ABSERR, P5*EPMACH*DABS (RESULT))
348             RETURN
349          ENDIF
350          E3 = EPSTAB(K1)
351          EPSTAB(K1) = E1
352          DELTA1 = E1 - E3
353          ERR1 = DABS (DELTA1)
354          TOL1 = DMAX1 (E1ABS, DABS (E3)) * EPMACH
C
355          IF (ERR1 .GT. TOL1 .AND. ERR2 .GT. TOL2 .AND.
     X           ERR3 .GT. TOL3) THEN
356             SS = ONE / DELTA1 + ONE / DELTA2 - ONE / DELTA3
357             EPSINF = DABS (SS * E1)
358          ELSE
C
C            SE DOIS ELEMENTOS TEM VALORES MUITO PROXIMOS, OMITE
C            A PARTE DA TABELA AJUSTANDO O VALOR DE N
C
359             N = I + I - 1
360             GOTO 50
361          ENDIF
C
C            VERIFICA SE HA COMPORTAMENTO IRREGULAR NA TABLE E
C            EVENTUALMENTE OMITE PARTE DA TABELA AJUSTANDO O
C            VALOR DE N
C
362          IF (EPSINF .LE. TENM4 ) THEN
363             N = I + I -1
364             GOTO 50
365          ENDIF
C
C            COMPUTA UM NOVO ELEMENTO E EVENTUALMENTE AJUSTA O VALOR
C            DE RESULT.
C
366          RES = E1 + ONE / SS
367          EPSTAB(K1) = RES
368          K1 = K1 - 2
369          ERROR = ERR2 + DABS (RES -E2) + ERR3
370          IF (ERROR .LE. ABSERR ) THEN
371             ABSERR = ERROR
372             RESULT = RES
373          ENDIF
374 40    CONTINUE
C
C     DESLOCA TABELA
C
375 50    CONTINUE
376       IF (N .EQ. LIMEXP) N = 2 * (LIMEXP / 2) - 1
377       IB = 1
378       IF ((NUM / 2 ) * 2 .EQ. NUM) IB = 2
379       IE = NEWELM + 1
380       DO 60 I = 1,IE
381          IB2 = IB + 2
382          EPSTAB(IB) = EPSTAB(IB2)
```

```
383            IB = IB2
384 60      CONTINUE
385         IF (NUM .NE. N) THEN
386            INDX = NUM - N + 1
387            DO 70 I = 1,N
388               EPSTAB(I) = EPSTAB(INDX)
389               INDX = INDX + 1
390 70         CONTINUE
391         ENDIF
392         IF (NRES .LT. 4) THEN
393            RES3LA(NRES) = RESULT
394            ABSERR = OFLOW
395            ABSERR = DMAX1 (ABSERR, P5 * EPMACH * DABS (RESULT))
396            RETURN
397         ENDIF
      C
      C     COMPUTA ESTIMATIVA DE ERRO
      C
398         ABSERR = DABS (RESULT - RES3LA(3)) + DABS (RESULT - RES3LA(2))
           X          + DABS (RESULT - RES3LA(1))
399         RES3LA(1) = RES3LA(2)
400         RES3LA(2) = RES3LA(3)
401         RES3LA(3) = RESULT
402         ABSERR = DMAX1 (ABSERR , P5 * EPMACH * DABS (RESULT))
403         RETURN
404         END


405         DOUBLE PRECISION FUNCTION DMAQ(I)
      C
406         INTEGER I
      C
      C     ESTA FUNCAO FORNECE OS PARAMETROS DO PONTO FLUTUANTE EM PRECISAO
      C
      C     ------ PARAMETRO ------
      C
      C     NA CHAMADA
      C        I          INTEGER
      C                   ENDERECA O PARAMETRO DESEJADO CONFORME DESCRICAO
      C                   ABAIXO. O VALOR DE I DEVE SATISFAZER AS SEGUINTES
      C                   CONDICOES..    1 .LE. I .AND. I .LE. 5
      C
      C     CONSTANTES DE MAQUINA PARA PRECISAO SIMPLES.
      C
      C     DMACH(1) = B**(EMIN - 1), O MENOR POSITIVO
      C
      C     DMACH(2) = B**EMAX(1 - B**(-T)), O MAIOR POSITIVO
      C
      C     DMACH(3) = B**(-T), O MENOR ESPA$AMENTO RELATIVO
      C
      C     DMACH(4) = B**(1 - T), O MAIOR ESPA$AMENTO RELATIVO
      C
      C     DMACH(5) = LOG10(B)
      C
      C     CONSTANTES PARA OUTROS EQUIPAMENTOS PODEM SER ENCONTRADAS NA
      C     REFERENCIA 2 ABAIXO CITADA.
      C
      C     REFERENCIAS
      C     1. P. A. FOX, A. D. HALL, AND N. L. SCHRYER, THE PORT
      C        MATHEMATICAL SUBROUTINE LIBRARY, ACM TRANS. MATH.
```

```
C        SOFTWARE 4(1978), PP 104-126.
C     2. P. A. FOX, A. D. HALL, AND N. L. SCHRYER, ALGORITHM 528 -
C        FRAMEWORK FOR A PORTABLE LIBRARY, ACM TRANS. MATH.
C        SOFTWARE 4(1978), PP 177-188.
C
C           UNIVERSIDADE FEDERAL DA PARAIBA
C           DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C           PROJETO MICRO-BITAN
C           MARIO T. HATTORI  OUT/84
C           ESTA VERSAO MAI/89
C
      CHARACTER*1 STAR
      DOUBLE PRECISION DMACH(5)
      INTEGER DIVER(2), LARGE(2), RIGHT(2), SMALL(2)
C
      EQUIVALENCE (DMACH(1), SMALL(1))
      EQUIVALENCE (DMACH(2), LARGE(1))
      EQUIVALENCE (DMACH(3), RIGHT(1))
      EQUIVALENCE (DMACH(4), DIVER(1))
      DATA STAR / '*' /
      DATA SMALL(1), SMALL(2) / 0,    1048576 /
      DATA LARGE(1), LARGE(2) /-1, 2146435071 /
      DATA RIGHT(1), RIGHT(2) / 0, 1016070144 /
      DATA DIVER(1), DIVER(2) / 0, 1017118720 /
      DATA DMACH(5) /  3.010299956639812D+00 /
C
      IF(I .LT. 1 .OR. I .GT. 5) WRITE(STAR, 9000)
 9000 FORMAT(32HERRO EM DMAQ. I FORA DOS LIMITES)
      DMAQ = DMACH(I)
C
      RETURN
      END



      INTEGER FUNCTION IMAQ(I)
C
      INTEGER I
C*************************************************************************
C     ESTA FUNCAO FORNECE CONSTANTES DE UM AMBIENTE DE COMPUTACAO PARA  *
C     SEREM UTILIZADOS EM (SUB)PROGRAMAS TRANSPORTAVEIS EM FORTRAN.     *
C                                                                       *
C     ------ PARAMETRO------                                           *
C                                                                       *
C     NA CHAMADA                                                        *
C        I          O VALOR DE I ENDERECA O PARAMETRO DESEJADO CONFORME *
C                   DESCRICAO ABAIXO.DEVE SATISFAZER AS SEGUINTES       *
C                   CONDICOES..  1 .LE. I .AND. I .LE. 16               *
C                                                                       *
C  NUMEROS LOGICOS DE ENTRADA/SAIDA                                     *
C     IMAQ( 1) = NUMERO LOGICO DA UNIDADE PADRAO DE ENTRADA             *
C     IMAQ( 2) = NUMERO LOGICO DA UNIDADE PADRAO DE SAIDA               *
C     IMAQ( 3) = NUMERO LOGICO DA UNIDADE PADRAO DE PERFURACAO          *
C     IMAQ( 4) = NUMERO LOGICO DA UNIDADE PADRAO DE MENSAGENS DE ERRO   *
C                                                                       *
C  PALAVRAS                                                             *
C                                                                       *
C     IMAQ( 5) = O NUMERO DE BITS POR UNIDADE DE ARMAZENAMENTO DE       *
C                INTEIROS                                               *
C     IMAQ( 6) = O NUMERO DE CARACTERES POR UNIDADE DE ARMAZENAMENTO    *
```

```
C                DE INTEIRO                                             *
C                                                                       *
C  INTEIROS                                                             *
C                                                                       *
C     SUPONHA QUE OS INTEIROS SAO REPRESENTADOS EM S DIGITOS NA BASE A  *
C        SINAL(X(S-1)*A**(S-1) + ... + X(1)*A + X(0) )                  *
C     EM QUE  0 .LE. X(I) .LT. A  PARA I = 0, ..., S-1.                 *
C                                                                       *
C     IMAQ( 7) = A, A BASE                                              *
C     IMAQ( 8) = S, O NUMERO DE DIGITOS NA BASE A                       *
C     IMAQ( 9) = A**S-1, O MAIOR VALOR ABSOLUTO                         *
C                                                                       *
C  NUMEROS EM PONTO FLUTUANTE                                           *
C                                                                       *
C     SUPONHA QUE OS NUMEROS SAO REPRESENTADOS  EM T DIGITOS NA BASE B  *
C        SINAL (B**E)*( (X(1)/B + ...+ (X(T)/**T) )                     *
C     EM QUE    0 .LE. X(I) .LT. B PARA I = 1, ... ,T                   *
C               0 .LT. X(1) E   EMIN  .LE. EMAX                         *
C                                                                       *
C     IMAQ(10) = B, A BASE                                              *
C                                                                       *
C  PRECISAO SIMPLES                                                     *
C                                                                       *
C     IMAQ(11) = T, O NUMERO DE DIGITOS NA BASE B                       *
C     IMAQ(12) = EMIN, O MENOR EXPOENTE                                 *
C     IMAQ(13) = EMAX, O MAIOR EXPOENTE                                 *
C                                                                       *
C  DUPLA PRECISAO                                                       *
C                                                                       *
C     IMAQ(14) = T, O NUMERO DE DIGITOS NA BASE B                       *
C     IMAQ(15) = EMIN, O MENOR EXPOENTE                                 *
C     IMAQ(16) = EMAX, O MAIOR EXPOENTE                                 *
C                                                                       *
C     PARA ALTERAR ESTA FUNCAO PARA UM AMBIENTE PARTICULAR, O CONJUNTO  *
C     DE DECLARACOES  DATA DEVE SER ATIVADO REMOVENDO  O C  DA COLUNA 1 *
C     A CONSISTENCIA DOS VALORES IMAQ(1) A IMAQ(4) COM O SISTEMA OPERA- *
C     CIONAL LOCAL DEVE SER VERIFCADA.                                  *
C-----------------------------------------------------------------------*
C     REFERENCIAS                                                       *
C     1. P. A. FOX, A. D. HALL, AND N. L. SCHRYER, THE PORT             *
C        MATHEMATICAL SUBROUTINE LIBRARY, ACM TRANS. MATH.              *
C        SOFTWARE 4(1978), PP 104-126.                                  *
C     2. P. A. FOX, A. D. HALL, AND N. L. SCHRYER, ALGORITHM 528 -      *
C        FRAMEWORK FOR A PORTABLE LIBRARY, ACM TRANS. MATH.             *
C        SOFTWARE 4(1978), PP 177-188.                                  *
C-----------------------------------------------------------------------*
C              UNIVERSIDADE FEDERAL DA PARAIBA                          *
C              DEPARTAMENTO DE SISTEMAS E COMPUTACAO                    *
C              MARIO T. HATTORI                                         *
C              VERSAO OUT/84                                            *
C***********************************************************************
C
      INTEGER IMACH(16), NOUT
C
      EQUIVALENCE (IMACH(4),NOUT)
C
C  -------- CONSTANTES PARA BURROUGHS 1700 ----------
C
C     DATA IMACH( 1) /    7 /
C     DATA IMACH( 2) /    2 /
```

```
C      DATA IMACH( 3) /    2 /
C      DATA IMACH( 4) /    2 /
C      DATA IMACH( 5) /   36 /
C      DATA IMACH( 6) /    4 /
C      DATA IMACH( 7) /    2 /
C      DATA IMACH( 8) /   33 /
C      DATA IMACH( 9) / Z1FFFFFFFF /
C      DATA IMACH(10) /    2 /
C      DATA IMACH(11) /   24 /
C      DATA IMACH(12) / -256 /
C      DATA IMACH(13) /  255 /
C      DATA IMACH(14) /   60 /
C      DATA IMACH(15) / -256 /
C      DATA IMACH(16) /  255 /
C
C  ------- CONSTANTES PARA BURROUGHS 5700 -------
C
C      DATA IMACH( 1) /    5 /
C      DATA IMACH( 2) /    6 /
C      DATA IMACH( 3) /    7 /
C      DATA IMACH( 4) /    6 /
C      DATA IMACH( 5) /   48 /
C      DATA IMACH( 6) /    6 /
C      DATA IMACH( 7) /    2 /
C      DATA IMACH( 8) /   39 /
C      DATA IMACH( 9) / 00007777777777777 /
C      DATA IMACH(10) /    8 /
C      DATA IMACH(11) /   13 /
C      DATA IMACH(12) /  -50 /
C      DATA IMACH(13) /   76 /
C      DATA IMACH(14) /   26 /
C      DATA IMACH(15) /  -50 /
C      DATA IMACH(16) /   76 /
C
C  ------- CONSTANTES PARA BURROUGHS 6700/7700 -------
C
C      DATA IMACH( 1) /    5 /
C      DATA IMACH( 2) /    6 /
C      DATA IMACH( 3) /    7 /
C      DATA IMACH( 4) /    6 /
C      DATA IMACH( 5) /   48 /
C      DATA IMACH( 6) /    6 /
C      DATA IMACH( 7) /    2 /
C      DATA IMACH( 8) /   39 /
C      DATA IMACH( 9) / 000077777777777777 /
C      DATA IMACH(10) /    8 /
C      DATA IMACH(11) /   13 /
C      DATA IMACH(12) /  -50 /
C      DATA IMACH(13) /   76 /
C      DATA IMACH(14) /   26 /
C      DATA IMACH(15) / -32754 /
C      DATA IMACH(16) /  32780 /
C
C      CONSTANTES PARA CDC SERIES 6000/7000
C
C      DATA IMACH( 1) /    5 /
C      DATA IMACH( 2) /    6 /
C      DATA IMACH( 3) /    7 /
C      DATA IMACH( 4) /    6 /
C      DATA IMACH( 5) /   60 /
```

```
C     DATA IMACH( 6) /   10 /
C     DATA IMACH( 7) /    2 /
C     DATA IMACH( 8) /   48 /
C     DATA IMACH( 9) / 00007777777777777777B /
C     DATA IMACH(10) /    2 /
C     DATA IMACH(11) /   48 /
C     DATA IMACH(12) / -974 /
C     DATA IMACH(13) / 1070 /
C     DATA IMACH(14) /   96 /
C     DATA IMACH(15) / -927 /
C     DATA IMACH(16) / 1070 /
C
C  ------- CONSTANTES PARA SERIES HONEYWELL 600/6000
C
C     DATA IMACH( 1) /    5 /
C     DATA IMACH( 2) /    6 /
C     DATA IMACH( 3) /   43 /
C     DATA IMACH( 4) /    6 /
C     DATA IMACH( 5) /   36 /
C     DATA IMACH( 6) /    6 /
C     DATA IMACH( 7) /    2 /
C     DATA IMACH( 8) /   35 /
C     DATA IMACH( 9) / 0377777777777 /
C     DATA IMACH(10) /    2 /
C     DATA IMACH(11) /   27 /
C     DATA IMACH(12) / -127 /
C     DATA IMACH(13) /  127 /
C     DATA IMACH(14) /   63 /
C     DATA IMACH(15) / -127 /
C     DATA IMACH(16) /  127 /
C
C  -------- CONSTANTES PARA AS SERIES IBM 360/370/3000/4000
C
      DATA IMACH( 1) /    5 /
      DATA IMACH( 2) /    6 /
      DATA IMACH( 3) /    7 /
      DATA IMACH( 4) /    6 /
      DATA IMACH( 5) /   32 /
      DATA IMACH( 6) /    4 /
      DATA IMACH( 7) /    2 /
      DATA IMACH( 8) /   31 /
      DATA IMACH( 9) / Z7FFFFFFF /
                       $
EXT* DA-06 data initialization with hexadecimal constant is not FORTRAN 77 stan
      DATA IMACH(10) /   16 /
      DATA IMACH(11) /    6 /
      DATA IMACH(12) /  -64 /
      DATA IMACH(13) /   63 /
      DATA IMACH(14) /   14 /
      DATA IMACH(15) /  -64 /
      DATA IMACH(16) /   63 /
C
C     CONSTANTES PARA PDP-10 PROCESSADOR KI
C
C     DATA IMACH( 1) /        5/
C     DATA IMACH( 2) /        6/
C     DATA IMACH( 3) /        5/
C     DATA IMACH( 4) /        3/
C     DATA IMACH( 5) /       36/
C     DATA IMACH( 6) /        5/
```

```
C      DATA IMACH( 7) /          2/
C      DATA IMACH( 8) /         35/
C      DATA IMACH( 9) /"377777777777/
C      DATA IMACH(10) /          2/
C      DATA IMACH(11) /         27/
C      DATA IMACH(12) /     -128 /
C      DATA IMACH(13) /        127/
C      DATA IMACH(14) /         62/
C      DATA IMACH(15) /       -128/
C      DATA IMACH(16) /        127/
C
C      CONSTANTES PARA O FORTRAN DO PDP-11 QUE SUPORTA ARITMETICA INTEIRA
C      DE 32 BITS.
C
C      DATA IMACH( 1) /           5 /
C      DATA IMACH( 2) /           6 /
C      DATA IMACH( 3) /           5 /
C      DATA IMACH( 4) /           6 /
C      DATA IMACH( 5) /          32 /
C      DATA IMACH( 6) /           4 /
C      DATA IMACH( 7) /           2 /
C      DATA IMACH( 8) /          31 /
C      DATA IMACH( 9) / 2147483647 /
C      DATA IMACH(10) /           2 /
C      DATA IMACH(11) /          24 /
C      DATA IMACH(12) /        -128 /
C      DATA IMACH(13) /         127 /
C      DATA IMACH(14) /          56 /
C      DATA IMACH(15) /        -127 /
C      DATA IMACH(16) /         127 /
C
C      CONSTANTES PARA O FORTRAN DO PDP-11 QUE SUPORTA ARITMETICA INTEIRA
C      DE 16 BITS.
C
C      DATA IMACH( 1) /           5 /
C      DATA IMACH( 2) /           6 /
C      DATA IMACH( 3) /           5 /
C      DATA IMACH( 4) /           6 /
C      DATA IMACH( 5) /          16 /
C      DATA IMACH( 6) /           2 /
C      DATA IMACH( 7) /           2 /
C      DATA IMACH( 8) /          15 /
C      DATA IMACH( 9) /       32767 /
C      DATA IMACH(10) /           2 /
C      DATA IMACH(11) /          24 /
C      DATA IMACH(12) /        -127 /
C      DATA IMACH(13) /         127 /
C      DATA IMACH(14) /          56 /
C      DATA IMACH(15) /        -127 /
C      DATA IMACH(16) /         127 /
C
C      CONSTANTES PARA A SERIE UNIVAC 1100
C
C      OBSERVE QUE A UNIDADE PERFURADORA, IMACH(3), RECEBEU O NUMERO 7, (
C      ADEQUADO PARA O SISTEMA UNIVAC-FOR. SE VOCE TEM OS SISTEMA
C      UNIVAC-FTN, ATRIBUA O NUMERO 1.
C
C      DATA IMACH( 1) /           5/
C      DATA IMACH( 1) /           5/
C      DATA IMACH( 2) /           6/
```

```
      C     DATA IMACH( 3) /           7/
      C     DATA IMACH( 4) /           6/
      C     DATA IMACH( 5) /          36/
      C     DATA IMACH( 6) /           6/
      C     DATA IMACH( 7) /           2/
      C     DATA IMACH( 8) /          35/
      C     DATA IMACH( 9) / 0377777777777 /
      C     DATA IMACH(10) /           2/
      C     DATA IMACH(11) /          27/
      C     DATA IMACH(12) /        -128/
      C     DATA IMACH(13) /         127/
      C     DATA IMACH(14) /          60/
      C     DATA IMACH(15) /       -1024/
      C     DATA IMACH(16) /        1023 /
      C
  445       IF (I .LT. 1 .OR. I.GT. 16) GO TO 20
  446          IMAQ = IMACH(I)
  447          RETURN
  448    20 CONTINUE
  449       WRITE(NOUT,9000)
  450  9000 FORMAT(33HERRO EM IMAQ - I FORA DOS LIMITES)
  451       RETURN
  452       END

ompile time:                    10.38  Execution time:              01.60
ize of object code:             10378  Number of extensions:           1
ize of local data area(s):       3615  Number of warnings:              0
ize of global data area:        18921  Number of errors:                0
bject/Dynamic bytes free:  156087/44648  Statements Executed:          4372
```

LISTAGEM DO PACOTE DQAGST UTILIZANDO DQAGS EM FORTRAN-77

```
JATFOR-77 V3.0 Copyright WATCOM Systems Inc. 1984,1988  92/06/12 21:46:42

)ptions: list,disk,xtype,terminal,extensions,warnings,check,arraycheck

      1        INTEGER IER, KEY, NEVAL
      2        DOUBLE PRECISION A, ABSERR, B, EPSABS, EPSREL, F, RESULT
      3        DOUBLE PRECISION DABS, ERRABS
      4        EXTERNAL F
        C
      5        A = 0.D0
      6        B = 1.D0
        C
      7        EPSABS = 0.D0
      8        EPSREL = 1.D-3
        C
      9        CALL DQAGS (F, A, B, EPSABS, EPSREL, RESULT,
               X            ABSERR, NEVAL, IER)
        C
     10        ERRABS = DABS (-4.D0 - RESULT)
     11        WRITE (6,900) RESULT, ABSERR, ERRABS, NEVAL, IER
     12    900 FORMAT( ' APROXIMACAO DA INTEGRAL =', D24.16//
               X         ' ESTIMATIVA DO ERRO ABSOLUTO =', D9.2//
               X         ' ERRO ABSOLUTO REAL =', D9.2//
               X         ' NUMERO DA AVALIACOES DO INTEGRANDO =', I5//
               X         ' CODIGO DE RETORNO =', I2)
     13        STOP
     14        END

     15        DOUBLE PRECISION FUNCTION F(X)
     16        DOUBLE PRECISION X
        C
     17        DOUBLE PRECISION DLOG, DSQRT
        C
     18        F = DLOG (X) / DSQRT (X)
     19        RETURN
     20        END


     21        SUBROUTINE DQAGS (F, A, B, EPSABS, EPSREL, RESULT, ABSERR,
               X                 NEVAL, IER)
        C
     22        INTEGER NEVAL, IER
     23        DOUBLE PRECISION F, A, B, EPSABS, EPSREL, RESULT, ABSERR
        C
        C      CALCULA UMA APROXIMACAO DE
        C           I = INTEGRAL DE F(X) SOBRE (A,B),
        C      ESPERANDO QUE O ERRO COMETIDO SATISFACA
        C           ABS (I - RESULT) .LE. MAX (EPSABS, EPSREL*ABS(I)).
        C      A ROTINA E CHAMADA POR DQAG. CONTUDO, PODE SER
        C      CHAMADO DIRETAMENTE PELO USUARIO.
        C
        C      PARAMETROS
        C
        C      NA CHAMADA
        C         F        DOUBLE PRECISION FUNCTION
        C                  E A FUNCAO INTEGRANDO DEFINIDA PELO USUARIO EM UM
        C                  SUBPROGRAMA FUNCAO. O NOME REAL DE F PRECISA SER
        C                  DECLARADO NUM EXTERNAL NO PROGRAMA ATIVADOR.
        C          A       DOUBLE PRECISION
```

```
C                LIMITE INFERIOR DO INTERVALO DE INTEGRACAO
C          B     DOUBLE PRECISION
C                LIMITE SUPERIOR DO INTERVALO DE INTEGRACAO
C          EPSABS DOUBLE PRECISION
C                TOLERANCIA DE ERRO ABSOLUTO SOLICITADA
C          EPSREL DOUBLE PRECISION
C                TOLERANCIA DE ERRO RELATIVO SOLICITADA. SE EPSABS E
C                EPSREL FOREM NEGATIVOS DQAGS SINALIZARA A OCORRENCIA
C                DE ERRO.
C          RESULT DOUBLE PRECISION
C                APENAS DECLARADO
C          ABSERR DOUBLE PRECISION
C                APENAS DECLARADO
C          NEVAL  INTEGER
C                APENAS DECLARADO
C          IER    INTEGER
C                APENAS DECLARADO
C
C       NO RETORNO
C          RESULT APROXIMACAO DA INTEGRAL
C          ABSERR ESTIMATIVA DO MODULO DO ERRO ABSOLUTO QUE DEVE
C                SER MAIOR OU IGUAL A ABS(I - RESULT).
C          NEVAL  O NUMERO DE AVALIACOES DO INTEGRANDO.
C          IER    SE O SEU VALOR FOR
C                 = 0 TERMINO NORMAL DA ROTINA. SUPOE-SE QUE A
C                     TOLERANCIA DE ERRO SOLICITADA FOI SATISFEITA
C                 = 1 O MAXIMO NUMERO SOLICITADO DE SUBDIVISOES DO
C                     INTERVALO FOI ATINGIDO. PODE-SE PERMITIR MAIS
C                     SUBDIVISOES AUMENTANDO-SE O VALOR DE LIMIT
C                     (PRECISA AJUSTAR DIMENSOES). SE ESSE AUMENTO NAO
C                     ACARRETAR MELHORIA NO RESULTADO ACONSELHA-SE
C                     ANALISAR O INTEGRANDO A FIM DE AVERIGUAR AS
C                     DIFICULDADES DE INTEGRACAO. SE A DIFICULDADE PUDER
C                     SER LOCALIZADA (POR EXEMPLO, SINGULARIDADE, DES-
C                     CONTINUIDADE DENTRO DO INTERVALO) VALERA A PENA
C                     SUBDIVIDIR O INTERVALO NESSE PONTO E CHAMAR O
C                     DQAGS EM CADA SUBINTERVALO. SE POSSIVEL, ESCOLHER
C                     UMA ROTINA ESPECIALIZADA QUE SEJA CAPAZ DE CONTOR-
C                     NAR A DIFICULDADE DETETADA.
C                 = 2 FOI DETETADA A OCORRENCIA DE ERRO DE ARREDONDAMENTO
C                     QUE NAO PERMITE ATINGIR A TOLERANCIA DE ERRO
C                     SOLICITADA.
C                 = 3 UM MAU COMPORTAMENTO EXTREMO DO INTEGRANDO OCORRE
C                     EM ALGUNS PONTOS NO INTERVALO DE INTEGRACAO.
C                 = 4 O ALGORITMO NAO CONVERGE. ERRO DE ARREDONDAMENTO
C                     E DETETADO NA TABELA DE EXTRAPOLACAO. PRESUME-SE
C                     QUE A TOLERANCIA SOLICITADA NAO PODE SER ALCANCADA
C                     DEVIDO AO ERRO DE ARREDONDAMENTO NA TABELA E QUE
C                     O RESULTADO FORNECIDO E O MELHOR QUE SE PODE OBTER.
C                 = 5 A INTEGRAL E PROVAVELMENTE DIVERGENTE OU LENTAMEN-
C                     TE CONVERGENTE. DEVE-SE OBSERVAR QUE A DIVERGENCIA
C                     PODE OCORRER COM QUALQUER OUTRO VALOR DE IER DIFE-
C                     RENTE DE ZERO.
C                 = 6 HOUVE ERRO NOS DADOS DE ENTRADA (EPSABS  E EPSREL
C                     NEGATIVOS). OS VALORES DE RESULT, ABSERR E NEVAL
C                     SERAO NULOS.
C
C       SUBPROGRAMAS UTILIZADOS
C          DO FORTRAN - DABS, DMAX1
C          DA BIBLIOTECA - DQEXT, DQK21, DQSORT
```

```
C         DO USUARIO - F
C
C      REFERENCIA
C         PIESSENS, DE DONCKER-KAPENGA, UBERHUBER, KAHANER, QUADPACK - A
C           SUBROUTINE PACKAGE FOR AUTOMATIC INTEGRATION, SPRINGER-
C           VERLAG, BERLIN HEIDELBERG N. YORK TOKYO, 1983
C
C                    UNIVERSIDADE FEDERAL DA PARAIBA
C                    DEPARTAMENTO DE SISTEMAS E COMPUTACAO
C                    PROJETO BITAN
C                    MARIO T. HATTORI  ABR/90
C
      INTEGER ID, IERRO, IORD(500), IROFF1, IROFF2, IROFF3,
     X        JUPBND, K, KSGN, KTMIN, LAST, LIMIT, MAXERR,
     X        NRES, NRMAX, NUMRL2
      LOGICAL EXTRAP, NOEXT
      DOUBLE PRECISION ABSEPS, ALIST(500), AREA, AREA1, AREA12, AREA2,
     X                 A1, A2, BLIST(500), B1, B2, CORREC, DEFAB1,
     X                 DEFAB2, DEFABS, DRES, ELIST(500), EPMACH, ERLARG,
     X                 ERLAST, ERRBND, ERRMAX, ERROR1, ERR012,
     X                 ERROR2, ERRSUM, ERTEST, OFLOW, RESABS, RESEPS,
     X                 RES3LA(3), RLIST(500), RLIST2(52), SMALL, UFLOW
      DOUBLE PRECISION DABS, DMAX1, DMAQ
      EXTERNAL F
      DATA LIMIT /500/
C
      EPMACH = DMAQ (4)
      OFLOW = DMAQ (2)
      UFLOW = DMAQ (1)
C
C   TESTA VALIDADE DOS PARAMETROS
C
      IER = 0
      NEVAL = 0
      LAST = 0
      RESULT = 0.0E+00
      ABSERR = 0.0E+00
      ALIST(1) = A
      BLIST(1) = B
      RLIST(1) = 0.0E+00
      ELIST(1) = 0.0E+00
      IF (EPSABS .LT. 0.0E+00 .AND. EPSREL .LT. 0.0E+00)  THEN
         IER = 6
      ENDIF
      IF (IER .NE. 6) THEN
C
C   PRIMEIRA APROXIMACAO DA INTEGRAL
C
         IERRO = 0
         CALL DQK21 (F, A, B , RESULT, ABSERR, DEFABS, RESABS)
C
C   TESTA EXATIDAO
C
         DRES = DABS (RESULT)
         ERRBND = DMAX1 (EPSABS, EPSREL * DRES)
         LAST = 1
         RLIST(1) = RESULT
         ELIST(1) = ABSERR
         IORD(1) = 1
         IF (ABSERR .LE. 1.0E+02 * EPMACH * DEFABS .AND. ABSERR .GT.
```

```
     X       ERRBND)  THEN
55             IER = 2
56          ENDIF
57          IF (LIMIT .EQ. 1)  THEN
58             IER = 1
59          ENDIF
60          IF (IER .EQ. 0 .AND. (ABSERR .GT. ERRBND .OR. ABSERR .EQ.
     X       RESABS) .AND. ABSERR .NE. 0.0E+00) THEN
C
C    DEFINE CONDICOES INICIAIS
C
61             RLIST2(1) = RESULT
62             ERRMAX = ABSERR
63             MAXERR = 1
64             AREA = RESULT
65             ERRSUM = ABSERR
66             ABSERR = OFLOW
67             NRMAX = 1
68             NRES = 0
69             NUMRL2 = 2
70             KTMIN = 0
71             EXTRAP = .FALSE.
72             NOEXT = .FALSE.
73             IROFF1 = 0
74             IROFF2 = 0
75             IROFF3 = 0
76             KSGN = -1
77             IF (DRES .GE. (1.0E+00 - 5.0E+01 * EPMACH) * DEFABS)  THEN
78                KSGN = 1
79             ENDIF
C
C    CICLO PRINCIPAL
C
80             DO 90 LAST = 2,LIMIT
C
C    BISSECCIONA O SUBINTERVALO COM A ESTIMATIVA DE ERRO MAIOR
C
81                A1 = ALIST(MAXERR)
82                B1 = 5.0E-01 * (ALIST(MAXERR) + BLIST(MAXERR))
83                A2 = B1
84                B2 = BLIST(MAXERR)
85                ERLAST = ERRMAX
86                CALL DQK21 (F, A1, B1, AREA1, ERROR1, RESABS, DEFAB1)
87                CALL DQK21 (F, A2, B2, AREA2, ERROR2, RESABS, DEFAB2)
C
C    MELHORA AS APROXIMACOES ANTERIORES DA INTEGRAL E DO ERRO E
C       FAZ TESTE DE EXATIDAO
C
88                AREA12 = AREA1 + AREA2
89                ERRO12 = ERROR1 + ERROR2
90                ERRSUM = ERRSUM + ERRO12 - ERRMAX
91                AREA = AREA + AREA12 - RLIST(MAXERR)
C
92                IF (DEFAB1 .NE. ERROR1 .AND. DEFAB2 .NE. ERROR2) THEN
93                   IF (DABS (RLIST(MAXERR) - AREA12) .LE. 1.0E-05 *
     X                DABS (AREA12) .AND. ERRO12 .GE. 9.9E-01 *
     X                ERRMAX) THEN
94                      IF (EXTRAP)  THEN
95                         IROFF2 = IROFF2 + 1
96                      ELSE
```

```
                        IROFF1 = IROFF1 + 1
                     ENDIF
                  ENDIF
                  IF (LAST .GT. 10 .AND. ERR012 .GT. ERRMAX)  THEN
                     IROFF3 = IROFF3 + 1
                  ENDIF
               ENDIF
               RLIST(MAXERR) = AREA1
               RLIST(LAST) = AREA2
               ERRBND = DMAX1 (EPSABS, EPSREL * DABS (AREA))
C
C    TESTA ERRO DE ARREDONDAMENTO E EVENTUALMENTE SINALIZA ERRO
C
               IF (IROFF1 + IROFF2 .GE. 10 .OR. IROFF3 .GE. 20)  THEN
                  IER = 2
               ENDIF
               IF (IROFF2 .GE. 5)  THEN
                  IERRO = 3
               ENDIF
C
C    NUMERO DE SUBINTERVALOS ATINGIU LIMIT
C
               IF (LAST .EQ. LIMIT)  THEN
                  IER = 1
               ENDIF
C
C    INTEGRANDO TEM MAU COMPORTAMENTO EM UM PONTO DO INTERVALO
C       DE INTEGRACAO
C
               IF (DMAX1 (DABS (A1), DABS (B2)) .LE. (1.0E+00 +
     X            1.0E+03 * EPMACH) * (DABS (A2) + 1.0E+03 *
     X            UFLOW))  THEN
                  IER = 4
               ENDIF
C
C    ANEXA OS INTERVALOS RECEM-CRIADOS A LISTA
C
               IF (ERROR2 .LE. ERROR1) THEN
                  ALIST(LAST) = A2
                  BLIST(MAXERR) = B1
                  BLIST(LAST) = B2
                  ELIST(MAXERR) = ERROR1
                  ELIST(LAST) = ERROR2
               ELSE
                  ALIST(MAXERR) = A2
                  ALIST(LAST) = A1
                  BLIST(LAST) = B1
                  RLIST(MAXERR) = AREA2
                  RLIST(LAST) = AREA1
                  ELIST(MAXERR) = ERROR2
                  ELIST(LAST) = ERROR1
               ENDIF
C
C    MANTEM A ORDENACAO DESCENDENTE NA LISTA DE ESTIMATIVAS DE ERRO
C       E SELECIONA O SUBINTERVALO COM ESTIMATIVA DE ERRO MAIOR
C       (PARA SER BISSECCIONADA EM SEGUIDA)
C
               CALL DQSORT (LIMIT, LAST, MAXERR, ERRMAX, ELIST, IORD,
     X              NRMAX)
C
```

```
      C   DESVIO PARA FORA DO CICLO
      C
  135                 IF (ERRSUM .LE. ERRBND) THEN
  136                    GOTO 115
  137                 ENDIF
  138                 IF (IER .NE. 0) THEN
  139                    GOTO 100
  140                 ENDIF
  141                 IF (LAST .NE. 2) THEN
  142                    IF (NOEXT) THEN
  143                       GOTO 90
  144                    ENDIF
  145                    ERLARG = ERLARG - ERLAST
  146                    IF (DABS (B1 - A1) .GT. SMALL)  THEN
  147                       ERLARG = ERLARG + ERRO12
  148                    ENDIF
  149                    IF (.NOT. EXTRAP) THEN
      C
      C   TESTA SE O PROXIMO INTERVALO A SER BISSECCIONADO E O MENOR
      C
  150                       IF (DABS (BLIST(MAXERR) - ALIST(MAXERR)) .GT.
     X                       SMALL) THEN
  151                          GOTO 90
  152                       ENDIF
  153                       EXTRAP = .TRUE.
  154                       NRMAX = 2
  155                    ENDIF
  156                    IF (IERRO .NE. 3 .AND. ERLARG .GE. ERTEST) THEN
      C
      C   O MENOR INTERVALO TEM O MAIOR ERRO. ANTES DE BISSECCIONAR DIMINUI
      C      A SOMA DOS ERROS SOBRE OS INTERVALOS MAIORES (ERLARG)
      C      E EXECUTA EXTRAPOLACAO.
      C
  157                       ID = NRMAX
  158                       JUPBND = LAST
  159                       IF (LAST .GT. (2 + LIMIT / 2))  THEN
  160                          JUPBND = LIMIT + 3 - LAST
  161                       ENDIF
  162                       DO 50 K = ID,JUPBND
  163                          MAXERR = IORD(NRMAX)
  164                          ERRMAX = ELIST(MAXERR)
      C
      C   DESVIO PARA FORA DO CICLO
      C
  165                          IF (DABS (BLIST(MAXERR) - ALIST(MAXERR)) .GT.
     X                          SMALL) THEN
  166                             GOTO 90
  167                          ENDIF
  168                          NRMAX = NRMAX + 1
  169                50        CONTINUE
      C
      C   EXECUTA EXTRAPOLACAO
      C
  170                    ENDIF
  171                    NUMRL2 = NUMRL2 + 1
  172                    RLIST2(NUMRL2) = AREA
  173                    CALL DQEXT (NUMRL2, RLIST2, RESEPS, ABSEPS, RES3LA,
     X                      NRES)
  174                    KTMIN = KTMIN + 1
  175                    IF (KTMIN .GT. 5 .AND. ABSERR .LT. 1.0E-03 *
```

```
     X                ERRSUM)  THEN
176                   IER = 5
177                ENDIF
178                IF (ABSEPS .LT. ABSERR) THEN
179                   KTMIN = 0
180                   ABSERR = ABSEPS
181                   RESULT = RESEPS
182                   CORREC = ERLARG
183                   ERTEST = DMAX1 (EPSABS, EPSREL * DABS (RESEPS))
C
C   DESVIO PARA FORA DO CICLO
C
184                   IF (ABSERR .LE. ERTEST) THEN
185                      GOTO 100
186                   ENDIF
C
C   PREPARA BISSECCAO DO MENOR INTERVALO.
C
187                ENDIF
188                IF (NUMRL2 .EQ. 1)  THEN
189                   NOEXT = .TRUE.
190                ENDIF
191                IF (IER .EQ. 5) THEN
192                   GOTO 100
193                ENDIF
194                MAXERR = IORD(1)
195                ERRMAX = ELIST(MAXERR)
196                NRMAX = 1
197                EXTRAP = .FALSE.
198                SMALL = SMALL * 5.0E-01
199                ERLARG = ERRSUM
200             ELSE
201                SMALL = DABS (B - A) * 3.75E-01
202                ERLARG = ERRSUM
203                ERTEST = ERRBND
204                RLIST2(2) = AREA
205             ENDIF
206    90       CONTINUE
C
C   COMPUTA O RESULTADO FINAL
C
207   100       CONTINUE
208             IF (ABSERR .NE. OFLOW) THEN
209                IF (IER + IERRO .NE. 0) THEN
210                   IF (IERRO .EQ. 3)  THEN
211                      ABSERR = ABSERR + CORREC
212                   ENDIF
213                   IF (IER .EQ. 0)  THEN
214                      IER = 3
215                   ENDIF
216                   IF (RESULT .EQ. 0.0E+00 .OR. AREA .EQ. 0.0E+00) THEN
217                      IF (ABSERR .GT. ERRSUM) THEN
218                         GOTO 115
219                      ENDIF
220                      IF (AREA .EQ. 0.0E+00) THEN
221                         GOTO 130
222                      ENDIF
223                   ELSE
224                      IF (ABSERR / DABS (RESULT) .GT. ERRSUM /
     X                   DABS (AREA)) THEN
```

```
225                         GOTO 115
226                      ENDIF
      C
      C   TESTA DIVERGENCIA
      C
227                      ENDIF
228                   ENDIF
229                   IF (KSGN .EQ. (-1) .AND.
     X                   DMAX1 (DABS (RESULT), DABS (AREA)) .LE. DEFABS *
     X                   1.0E-02) THEN
230                      GOTO 130
231                   ENDIF
232                   IF (1.0E-02 .GT. (RESULT / AREA) .OR. (RESULT /
     X                   AREA) .GT. 1.0E+02 .OR. ERRSUM .GT. DABS (AREA))  THEN
233                      IER = 6
234                   ENDIF
235                   GOTO 130
236                ENDIF
237     115        CONTINUE
      C
      C   COMPUTA INTEGRAL
      C
238                RESULT = 0.0E+00
239                DO 120 K = 1,LAST
240                   RESULT = RESULT + RLIST(K)
241     120        CONTINUE
242                ABSERR = ERRSUM
243     130        CONTINUE
244                IF (IER .GT. 2)  THEN
245                   IER = IER - 1
246                ENDIF
247             ENDIF
248             NEVAL = 42 * LAST - 21
249          ENDIF
250          RETURN
251          END
```

```
      C     OBSERVE QUE A UNIDADE PERFURADORA, IMACH(3), RECEBEU O NUMERO 7, Q
      C     ADEQUADO PARA O SISTEMA UNIVAC-FOR. SE VOCE TEM OS SISTEMA
      C     UNIVAC-FTN, ATRIBUA O NUMERO 1.
      C
      C     DATA IMACH( 1) /           5/
      C     DATA IMACH( 1) /           5/
      C     DATA IMACH( 2) /           6/
      C     DATA IMACH( 3) /           7/
      C     DATA IMACH( 4) /           6/
      C     DATA IMACH( 5) /          36/
      C     DATA IMACH( 6) /           6/
      C     DATA IMACH( 7) /           2/
      C     DATA IMACH( 8) /          35/
      C     DATA IMACH( 9) / O377777777777 /
      C     DATA IMACH(10) /           2/
      C     DATA IMACH(11) /          27/
      C     DATA IMACH(12) /        -128/
      C     DATA IMACH(13) /         127/
      C     DATA IMACH(14) /          60/
      C     DATA IMACH(15) /       -1024/
      C     DATA IMACH(16) /        1023 /
      C
  504       IF (I .LT. 1 .OR. I.GT. 16) GO TO 20
  505          IMAQ = IMACH(I)
  506          RETURN
  507    20 CONTINUE
  508       WRITE(NOUT,9000)
  509  9000 FORMAT(33HERRO EM IMAQ - I FORA DOS LIMITES)
  510       RETURN
  511       END
```

```
ompile time:                    11.20  Execution time:                01.59
ize of object code:             10610  Number of extensions:             1
ize of local data area(s):       3615  Number of warnings:               0
ize of global data area:        18921  Number of errors:                 0
bject/Dynamic bytes free:  155847/43984  Statements Executed:           4488
```