PROGRAMAÇÃO FORTRAN

PARA ENGENHARIA

Fabiano A.N. Fernandes

1ª Edição
2003

SUMÁRIO

1. INTRODUÇÃO

O Fortran tem sido usado por cientistas e engenheiros por muitos anos, sendo uma das principais linguagens de programação científica especialmente devido a sua capacidade em fazer cálculos. Taxada de linguagem obsoleta pelas pessoas que desconhecem as novas atualizações na sua estrutura de programação, o Fortran hoje possui todos os elementos de linguagem que tornaram o C++ famoso.

O Fortran, abreviação de FORmula TRANslation (ou originalmente IBM Mathematical FORmula Translation System), é a mais velha das linguagens de *alto nível* e foi desenvolvida pelo grupo IBM no final da década de 1950. A linguagem ganhou popularidade na década de 1960 e ganhou sua primeira versão padronizada: Fortran 66.

Em meados da década de 1970, todo grande computador vinha com a linguagem Fortran embutida e era a linguagem mais robusta da época e tinha um processamento muito eficiente. Além disso o código podia ser compilado (transformado em um programa executável) em qualquer tipo de sistema de computador e portanto se tornou a linguagem mais usada no meio científico. O domínio do Fortran levou a uma nova atualização que ganhou o nome de Fortran 77 (pelo qual o Fortran é conhecido até hoje).

Infelizmente a revisão para o Fortran 77 foi muito conservadora mantendo uma estrutura de programação antiga. Com a popularização dos computadores pessoais, os jovens programadores da década de 1980 preferiam aprender Basic, Pascal e no final dos anos 80, o C; que eram linguagens que tinham uma estrutura de programação mais bem estruturada e moderna. Essa preferência dos jovens programadores levou no início da década de 1990 a uma mobilização para implantar o C++ como linguagem de programação preferencial no meio científico, aliando capacidade de cálculo com uma estrutura moderna de programação. A migração para o C++ só não foi maior porque muitas rotinas de métodos numéricos estavam em Fortran e daria muito trabalho e levaria muito tempo para traduzi-las para o C++.

Na mesma época (1991) o Fortran recebeu sua maior atualização, com a introdução do Fortran 90 que permitia o uso de muitos comandos e estrutura das linguagens mais modernas.

1.1. O Curso

Este curso, irá apresentar os principais comandos do Fortran 90 usados para fazer projetos de engenharia. Os exemplos e exercícios focam em problemas tradicionais e de utilização prática.

Ao final do curso, alguns métodos numéricos mais utilizados são abordados, mostrando como criar programas usando bibliotecas numéricas.

2. LÓGICA DE PROGRAMAÇÃO

Programar em Fortran, assim como em qualquer outra linguagem de programação é simples, o complicado é organizar o pensamento lógico e estruturar a resolução do problema para se atingir o objetivo que se deseja.

É um erro comum e grave para o iniciante em programação, escrever um programa sem ao menos esquematizar as ações que devem ser executadas pelo programa (algoritmo) de modo a solucionar o problema.

Nos primeiros programas, o algoritmo ajuda a organizar o pensamento lógico, principalmente quando decisões devem ser tomadas ou operações com vetores e matrizes são necessários.

Após algum tempo de experiência, o processo de organização da estrutura do programa passa de a ser lógico e fácil, não sendo necessário fazer um algoritmo muito detalhado. Porém se o programa for utilizado por mais de uma pessoa, o algoritmo ainda é necessário para facilitar o entendimento do programa por outras pessoas, uma vez que ler um algoritmo é bem mais fácil do que ler o código de um programa.

2.1. Algoritmo

Um algoritmo é uma sequência finita de passos que levam a execução de uma tarefa, ou seja, é a receita que deve ser seguida para se chegar a uma meta específica. O programa por sua vez, é nada mais do que um algoritmo escrito numa linguagem de computador.

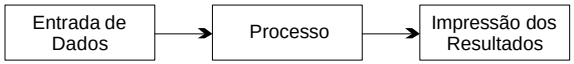
Regras Básicas para Construção de um Algoritmo

Para escrever um algoritmo deve-se descrever a sequência de instruções de maneira simples e objetiva, podendo-se utilizar algumas técnicas básicas:

- usar somente um verbo por frase
- usar frases curtas e simples
- ser objetivo
- usar palavras que não tenham sentido dúbio

Fases de um Algoritmo

O algoritmo deve conter as três fases fundamentais da resolução de um problema. Estas fases são a leitura de dados, cálculos (ou processo) e impressão dos resultados.



EXEMPLO 1
Um algoritmo para calcular a média de três números tomaria a forma:

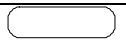
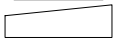
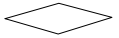
1. Ler N1, N2 e N3
2. Calcular Média pela equação: $Media = \frac{N1 + N2 + N3}{3}$
3. Imprimir Média

2.2. Fluxograma

O fluxograma é uma forma padronizada e eficaz para representar os passos lógicos de um determinado processo. Sua principal função é a de facilitar a visualização dos passos de um processo.

O fluxograma é constituído por diversos símbolos que representam estes elementos de programação (Tabela 2.1). No interior dos símbolos sempre existirá algo escrito denotando a ação a ser executada.

Tabela 2.1. Elementos do fluxograma

	início e fim
	leitura de dados
	impressão de dados
	ação
	decisão
	conexão

EXEMPLO 2
O fluxograma para o exemplo **1** tomaria a forma:

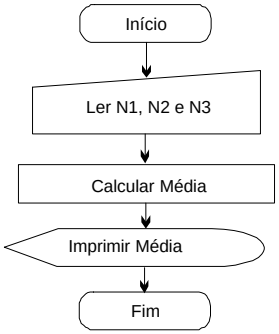


Figura 2.1. Fluxograma para cálculo da média de três números.

EXEMPLO 3
Uma aplicação simples em engenharia é o calculo do balanço de massa em um tanque de mistura, como o mostrado na Figura 2.2.

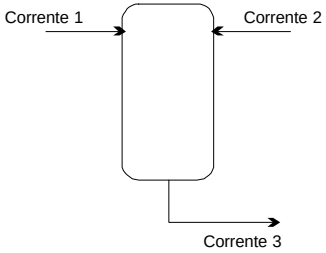


Figura 2.2. Tanque de mistura

Supondo que não há acúmulo volumétrico no interior do tanque, e que as densidade das correntes de entrada (1 e 2) são diferentes, o cálculo do fluxo volumétrico de saída do tanque (corrente 3), do fluxo mássico e da densidade no tanque pode ser feito usando as equações:

Fluxo volumétrico: $F3 = F1 + F2$

Fluxo Mássico: $M3 = F1 \cdot r1 + F2 \cdot r2$

Densidade: $r3 = \frac{F1 \cdot r1 + F2 \cdot r2}{F3}$

Fi	fluxo volumétrico da corrente i
Mi	fluxo mássico da corrente i
ρi	densidade da corrente i

O fluxograma a ser seguido para cálculo do tanque será:

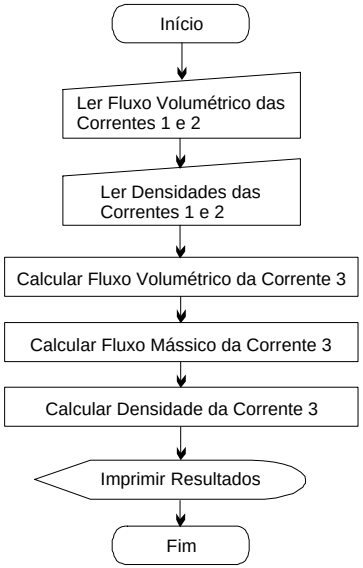


Figura 2.3. Fluxograma para cálculo do balanço de massa em um tanque agitado.

EXEMPLO 4
Se considerarmos os trocadores de calor, o coeficiente de troca térmica depende do tipo de escoamento (laminar ou turbulento) e pode ser calculado por meio de correlações que são definidas para cada faixa de número de Reynolds.
Um programa que calcule o coeficiente de troca térmica deve conter um processo decisório que utilize a correlação correta em função do valor do número de Reynolds, conforme as equações:

EQ1: $N_{Nu} = 0{,}153 \cdot N_{Re}^{\,0{,}8} \cdot N_{Pr}^{\,0{,}33} \cdot f^{\,0{,}14}$ para $N_{Re} < 2100$

EQ2: $N_{Nu} = 10{,}56 \cdot N_{Re}^{0{,}33} \cdot N_{Pr}^{0{,}33} \cdot \left(\frac{d}{L}\right)^{0{,}33} \cdot f^{0{,}14}$

para $N_{Re} > 2100$

d	diâmetro do tubo
L	comprimento do tubo
N_{Nu}	número de Nusselt
N_{Pr}	número de Prandtl
N_{Re}	número de Reynolds
ϕ	razão de viscosidade do fluido no centro e na parede do tubo

O fluxograma do programa para cálculo do coeficiente de transferência de calor será:

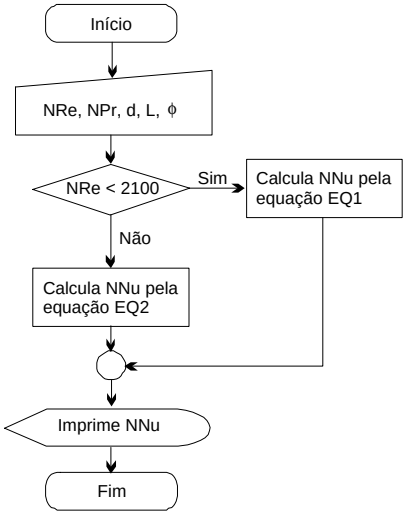


Figura 2.4. Fluxograma para cálculo do coeficiente de transferência de calor.

No fluxograma acima, após a leitura das variáveis necessárias, o programa deve decidir qual das duas equações será usada para o cálculo do número de Nusselt,. Esta decisão é feita comparando o número de Reynolds lido com o limite superior para a aplicação da equação EQ1. Dependendo do valor do número de Reynolds, o número de Nusselt será calculado pela EQ1 ou pela EQ2.

EXEMPLO 4

É muito comum em engenharia, termos que gerar dados para montar um gráfico de uma determinada função. A velocidade terminal de uma partícula é função do tamanho da partícula e das propriedades do fluido e do sólido e pode ser calculada pela equação:

$$u_t = \frac{0{,}524 \cdot D_p^{\,2} \cdot (r_s - r_f)}{m}$$

Se quisermos gerar 100 pontos para construir um gráfico da velocidade superficial em função do diâmetro de partícula, para partículas variando de 50 a 1000 μm poderemos usar o seguinte fluxograma:

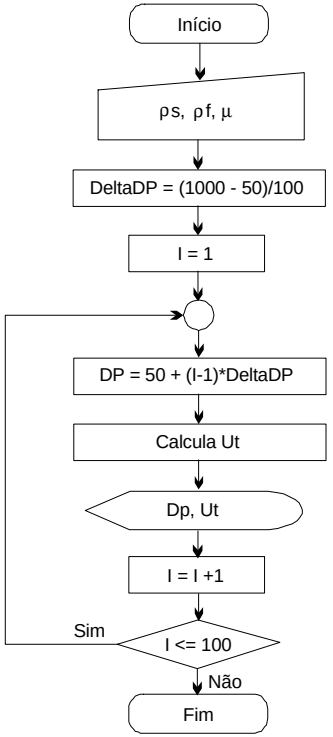


Figura 2.5. Fluxograma para cálculo do coeficiente de transferência de calor.

No fluxograma acima, um contador (**I**) é utilizado para fazer a iteração de 1 até 100 que é o número de pontos desejado para o gráfico. Um valor de incremento é definido para o diâmetro das partículas (**DeltaDP**) e este é usado no cálculo do diâmetro da partícula (**DP**). Após a velocidade terminal (**UT**) ser calculada, os valores de **DP** e **UT** são impressos. O contador é incrementado em uma unidade e o processo continua até que 100 pontos sejam impressos.

EXEMPLO 5

A tecnologia Pinch, usada para otimizar a troca de energia entre as diversas correntes de um processo, requer a organização das temperatura das diversas correntes em ordem decrescente, em uma de suas etapas.
As temperaturas das correntes são armazenadas em um vetor que deve ser organizado do maior valor para o menor valor.
Se a temperatura de 10 correntes tiverem de ser organizadas, o fluxograma a ser seguido será dado por:

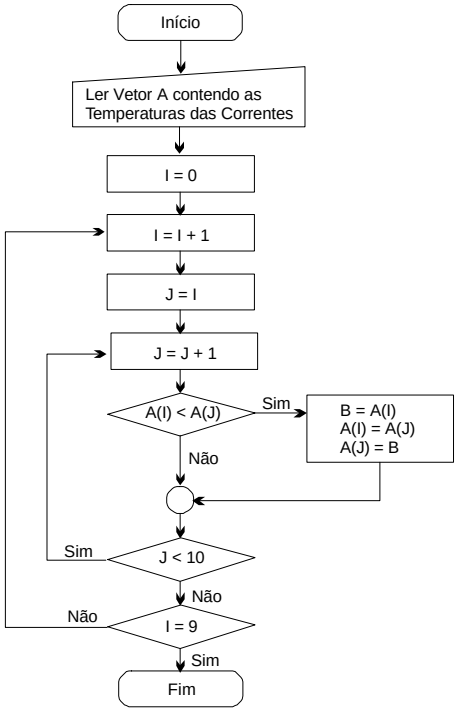


Figura 2.6. Fluxograma para organização de um vetor em ordem decrescente.

Neste fluxograma usamos o conceito de contadores (variáveis **I** e **J**), que servem para contar o número de iterações realizadas, ou simplesmente para marcar uma posição. Neste caso os contadores servem para indicar qual a posição no vetor **A** que contém as temperaturas.
Para organizar o vetor é necessário procurar pelo maior valor e colocá-lo na primeira posição do vetor, buscar pelo segundo maior valor e colocá-lo na segunda posição do vetor e assim por diante.
Se inicialmente o vetor estiver totalmente desorganizado o maior valor pode estar em qualquer posição no interior do vetor, como por exemplo:

Posição	1	2	3	4	5	6	7	8	9	10
Valor	12	9	25	11	22	12	15	3	7	18

Para achar o maior valor e colocá-lo na primeira posição do vetor, podemos usar o contador **I** e dar a ele o valor **1** referente à primeira posição no vetor **A**. Portanto a variável **A(I)** conterá o valor do primeiro valor do vetor, ou seja, **A(1)**.
Para colocar o maior valor do vetor nesta posição, devemos comparar o valor desta posição com os valores contidos nas outras posições do vetor, ou seja com as posições 2 até 10.
Para controlar qual a posição que será comparada com a posição **I**, podemos usar o controlador **J**, fazendo este variar de 2 até 10. Se o valor de **A(J)** for maior que o valor de **A(I)**, então trocamos estes valores de posição de forma que o maior valor fique na primeira posição:

Posição	1	2	3	4	5	6	7	8	9	10
Valor	12	9	25	11	22	12	15	3	7	18

Uma vez que a primeira posição do vetor foi preenchida corretamente com o maior valor do vetor, podemos repetir a mesma operação para achar o segundo maior valor e colocá-lo na segunda posição do vetor. Para tanto, o contador **I** é incrementado recebendo o valor **2** referente à segunda posição no vetor **A**. Para colocar o segundo maior valor do vetor nesta posição, devemos comparar o valor desta posição com os valores contidos nas posições restantes do vetor, ou seja com as posições, 3 até 10. Novamente, se o valor de **A(J)** for maior que o valor de **A(I)**, então trocamos estes valores de posição de forma que o maior valor fique na segunda posição:

Posição	1	2	3	4	5	6	7	8	9	10
Valor	25	9	12	11	22	12	15	3	7	18

Note que o contador **I** deve variar entre I+1 e 10 (última posição do vetor).
A operação é repetida para as outras posições do vetor. Para um vetor com 10 posições, o valor do contador **I** varia de 1 a 9 e não de 1 a 10 pois no final do processo, o valor contido na posição 10 já será o menor valor contido no vetor. Além disso não seria possível comparar o valor **A(10)** com o valor **A(11)** pois este último não existe.

Dos exemplos mostrados neste capítulo, o exemplo 5 é um dos problemas mais complicados que se tem em lógica de programação para engenharia, pois envolve operação com vetores, controle de vetores, loops e comparações.

Embora, a modelagem e a resolução dos problemas de engenharia sejam muitas vezes complexos, a lógica de programação a ser utilizada será na grande maioria dos casos muito parecida com os exemplos mostrados neste capítulo.

Nos próximos capítulo iremos abordar os comandos que nos permitem programar em Fortran.

EXERCÍCIOS

EXERCÍCIO 1
Um procedimento muito comum em programação para engenharia é a obtenção das raízes de uma função. Para uma função de segundo grau, como a mostrada no exemplo 4, em que a velocidade de terminação é uma função de segundo grau em relação ao diâmetro da partícula, podemos determinar de duas formas o diâmetro da partícula dado uma velocidade terminal. Diretamente, reorganizando a equação isolando o diâmetro da partícula em função da velocidade terminal:

$$Dp = \pm\sqrt{\frac{u_t \cdot m}{0{,}524 \cdot (r_s - r_f)}}$$

ou pela técnica de bisseção, buscando o zero da função:

$$0 = u_t - \frac{0{,}524 \cdot D_p{}^2 \cdot (r_s - r_f)}{m}$$

Desenvolva o fluxograma para calcular o diâmetro da partícula a partir de cada um destes dois processos.

EXERCÍCIO 2
A correlação a ser utilizada para calcular as propriedades fisicoquímicas depende da fase em que a substância se encontra: gás ou líquido. A decisão de qual correlação deve ser utilizada pode ser feita com base na comparação entre a temperatura de ebulição do composto e a temperatura do processo.
Desenvolva um fluxograma para calcular a capacidade calorífica de uma substância.
As correlações para o cálculo da calorífica são:

para gás:

$$Cp = A + B \cdot T + C \cdot T^2$$

para líquido:

$$Cp = \frac{A^2}{t} + B - 2 \cdot A \cdot C \cdot t - A \cdot D \cdot t^2$$

$$t = 1 - \frac{T}{T_c}$$

Cp	capacidade calorífica
T	temperatura
Tb	temperatura de ebulição
Tc	temperatura crítica
A, B, C, D	parâmetros

EXERCÍCIO 3
O exemplo 5 apresentou como se organiza um vetor (contendo 10 valores) em ordem decrescente. Desenvolva um algoritmo que organize um vetor, contendo *N* valores, em ordem crescente.

3. COMPILADOR FORTRAN

Compilador é o nome que se dá ao programa que irá transformar o seu código Fortran em um programa executável. Existem vários compiladores Fortran, como o Intel Fortran, Compaq Fortran, GCC, ProFortran, entre outros. Atualmente os compiladores mais usados são:

- **INTEL e COMPAQ FORTRAN**
  Devido a facilidade de sua interface, modernidade do código que compila, capacidade de gerar aplicativos com interface gráfica em Windows (QuickWin) e grande variedade de métodos já codificados em sua biblioteca numérica.

- **GNU FORTRAN (GCC)**
  Devido a ser um programa livre (grátis). É um compilador para Fortran 77 mas contém a maioria dos comandos do Fortran 90 além da possibilidade de formatação livre do código. Não cria aplicativos com interface gráfica e não contém módulo de bibliotecas numéricas.

Os programas a serem feitos neste curso poderão ser executados em qualquer compilador Fortran com capacidade de compilar Fortran 90. Somente alguns exemplos de capítulo 12 sobre métodos matemáticos irão requerer a biblioteca numérica IMSL.

As seções seguintes irão apresentar como iniciar um projeto no COMPAQ Fortran, que é a versão atual do antigo mas ainda popular MS Fortran PowerStation. O INTEL Fortran é a nova denominação do agora antigo COMPAQ Fortran (a diferença é a possibilidade de integração com a plataforma .NET da Microsoft)

3.1. Criando um Projeto

No COMPAQ Fortran, todo programa em Fortran está ligado a um projeto que irá conter o código fonte do programa que está sendo escrito. Para criar um projeto no Fortran, selecione *File* no menu principal e depois selecione *New* (Figura 3.1).

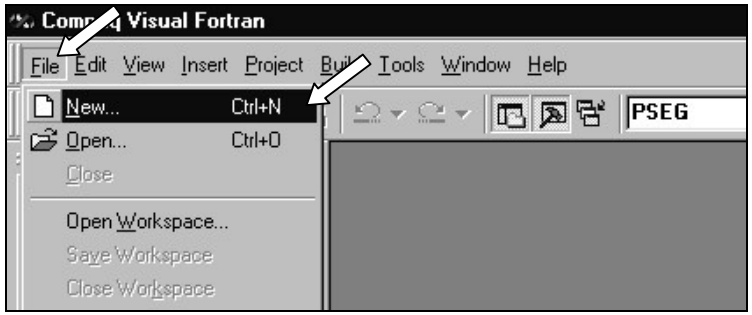


Figura 3.1. Abertura de um novo projeto no Fortran

Este compilador é capaz de criar vários tipos de programas (programa executável, subrotina DLL, programas com interface Windows, etc.). Neste curso abordaremos os programas executáveis, portanto escolha a opção *Fortran Console Application* (Figura 3.2).

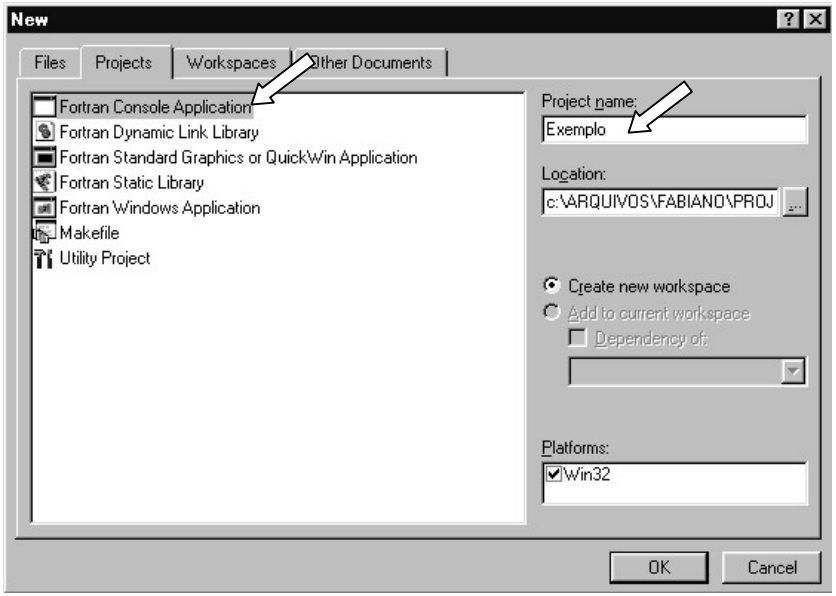


Figura 3.2. Abertura de um novo projeto no Fortran

Dê um nome para o projeto que estará sendo criado. Um novo diretório será criado com o nome deste projeto. Será neste diretório que os arquivos com o código do programa em Fortran deverão ser gravados (Figura 3.2).

Escolha para criar um projeto vazio (Figura 3.3). Finalize a abertura do projeto pressionando o botão *Finish*.

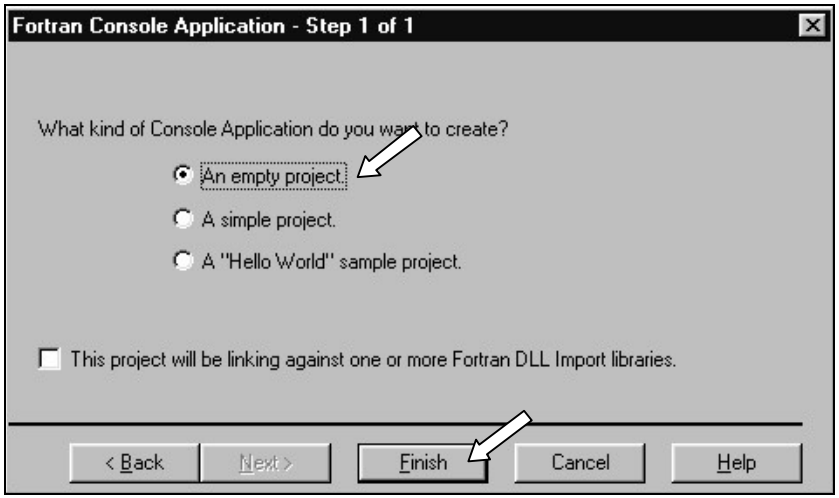


Figura 3.3. Abertura de um novo projeto no Fortran

Após criado o projeto, o arquivo que conterá o código em Fortran deverá ser criado. Este arquivo é um arquivo texto comum que posteriormente será gravado com a extenção **.f90**. Para criar o arquivo do código, pressione o botão *New Text File* (Figura 3.4).

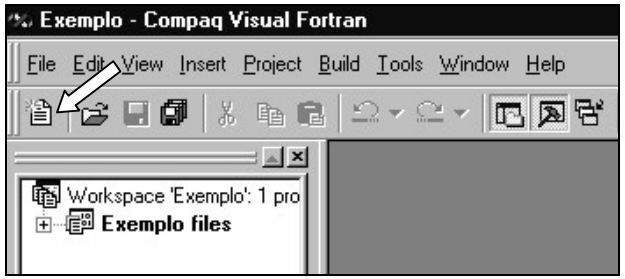


Figura 3.4. Abertura de um novo arquivo de código.

Este arquivo texto poderá ser editado e o código do programa poderá ser digitado nele. Após editado, este arquivo deve ser gravado com a extensão **.f90**. Para salvar o arquivo selecione *File* no menu principal e depois selecione a opção *Save*, ou simplesmente pressione o botão *Save* (Figura 3.5). O nome deste arquivo poderá ser igual ao nome do projeto (recomendável para não causar muita confusão).

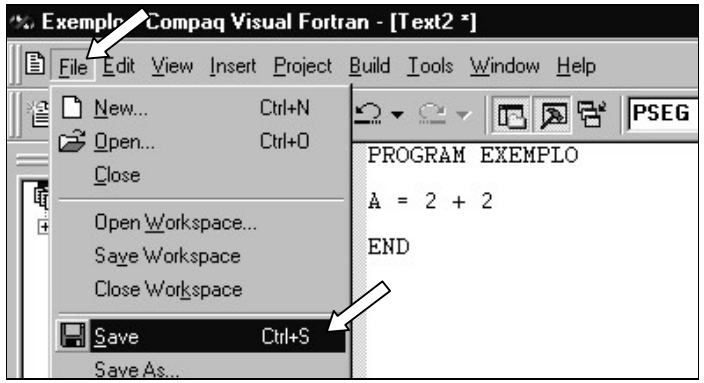


Figura 3.5. Gravação de um novo arquivo de código.

Não esqueça de gravar o arquivo com a extensão **.f90** (Figura 3.6).

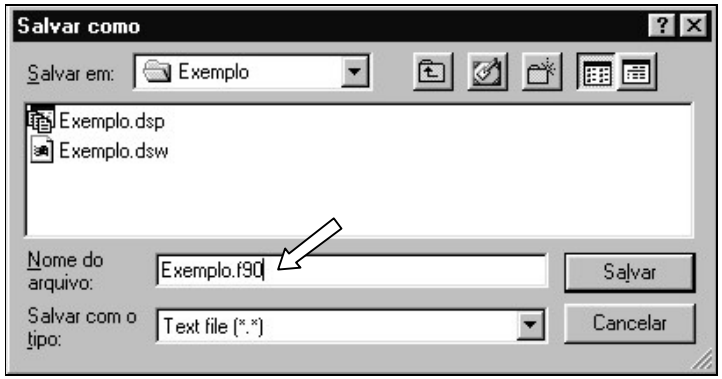


Figura 3.6. Gravação de um novo arquivo de código.

Este arquivo por sua vez deverá ser inserido no projeto. Para isto, selecione *Project* no menu principal e depois selecione a opção *Add To Project* e *Files* (Figura 3.7). Selecione o arquivo **f90** que foi criado.

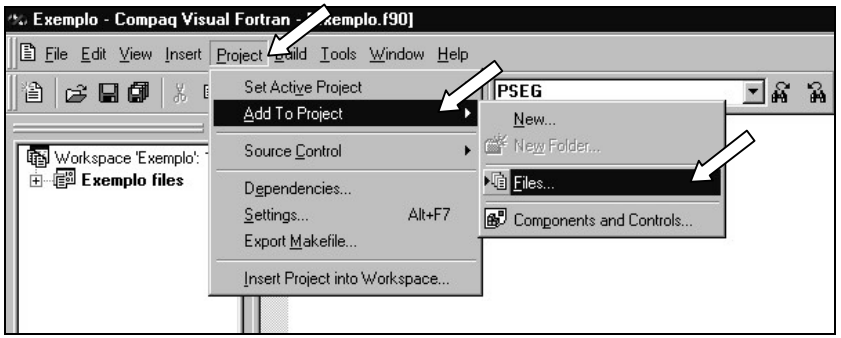


Figura 3.7. Vinculação do arquivo de código ao projeto.

*Atenção: não é porque o arquivo **f90** está aberto no compilador que ele está vinculado ao projeto. Isto só ocorre após o usuário fazer a inserção manual deste arquivo ao projeto.*

Depois de vincular o arquivo **f90** ao projeto, o projeto deve ser salvo para gravar este novo vínculo. Após este procedimento, o arquivo com o código Fortran pode ser editado, e o programa escrito.

Após pronto, o código deve ser compilado para então se tornar um programa executável. A compilação é feita selecionando *Build* no menu principal e depois a opção *Rebuild All* no menu principal (ou pressione o botão *Rebuild All*). Se o compilador encontrar erros no código do programa que impeçam a criação do programa executável, as mensagem de erro aparecerão na janela abaixo do código (Figura 3.8).

Figura 3.8. Compilação e criação do programa executável.

Selecionar *Rebuild All* como mostrado na Figura 3.8 evita o trabalho de ter que selecionar *Compile* e depois selecionar *Build*.

Para executar o programa, selecione a opção *Build* no menu principal e depois a opção *Execute*, ou simplesmente pressione o botão *Execute* (Figura 3.9).

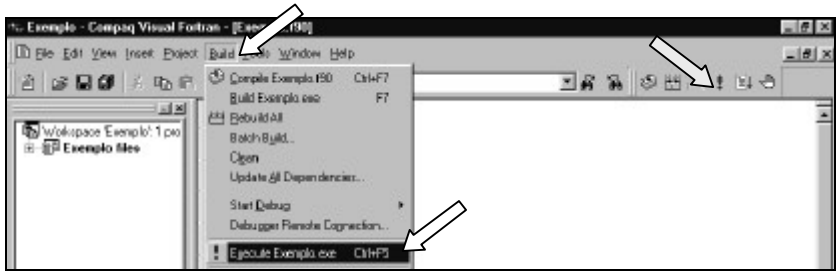


Figura 3.9. Execução de um programa.

3.1.1. Usando um Código Pronto em um Novo Projeto

Se quiser começar um novo projeto e importar um arquivo de código existente para este novo projeto siga o seguinte procedimento:

- Crie o novo projeto.
- Copie o arquivo de código para o diretório criado para o novo projeto.
- Vincule o arquivo de código ao novo projeto.

Se o arquivo de código não for copiado para o novo diretório, este código será compartilhado por dois ou mais projetos e uma modificação neste código implicará em mudanças no código para os dois projetos. Portanto, se quiser modificar o código do programa sem afetar a última versão, o procedimento acima deve ser seguido.

3.2. Código em FORTRAN 90

O código do programa em Fortran 90 tem formatação livre, com o código podendo ser escrito a partir da primeira coluna e não há limite de caracteres por linha.

O programa começa com o comando **PROGRAM** e termina com o comando **END**.

```
PROGRAM <nome>
:
código
:
END
```

onde <nome> é o nome dado ao programa

O comando **PROGRAM** é na verdade opcional, mas pode vir a ser importante para diferenciar o programa principal dos outros módulos, subrotinas e funções (veremos estas estruturas no Capítulo 11).

É possível inserir comentários ao longo do programa de forma a identificar as diversas partes do programa e descrever o que está sendo realizado em cada parte. O comentário começa com o caracter **!**

```
PROGRAMA EXEMPLO
! PROGRAMA PARA CALCULO DE 2 + 2
A = 2 + 2      ! EQUAÇÃO
END
```

Muitas vezes as equações são muito longa para caberem na tela, de forma que a linha do programa sairia do campo visual. Neste caso o caracter **&** pode ser usado para indicar que esta linha de código continua na linha seguinte. O **&** deve vir no final da linha.

```
PROGRAMA EXEMPLO
! CALCULO DE UM BALANÇO POPULACIONAL
A = (TAU + BETA)*(TAU + BETA/2.0*(TAU + BETA)*(R – 1.0))*R/ &
      (1.0 + TAU + BETA)**R
END
```

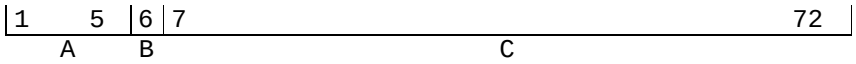
2.3. Código em FORTRAN 77

O Fortran 77 é a versão antiga da linguagem Fortran. Ainda hoje ela é bastante popular pois alguns programadores aprenderam a programar em Fortran 77 e escolheram não se atualizar para o usar o Fortran 90. Portanto é muito comum ver programas novos sendo escritos em Fortran 77.

As desvantagens do Fortran 77 em relação ao Fortran 90 são: não poder usar alguns comandos novos que foram criados com o Fortran 90; maior

dificuldade em fazer alguns tipos de operações com vetores e matrizes; impossibilidade de criar DLLs; e ter que conviver com regras mais rígidas para escrever o programa.

O Fortran 77 tem várias regras de escrita do código, sendo que as linhas de código são divididas por seções:



Zona A – contém comentários e números de linha de código (linhas 1 a 5).
Zona B – contém o caracter que indica a continuação da linha anterior (linha 6).
Zona C – código do programa (linhas 7 a 72).

Um programa em Fortran 77 teria a forma:

```
1   5 6 7                                                   72
      PROGRAM <NOME>
      :
      código
      :
      END
```

A inserção de comentários deve ser feita colocando a letra **C** na primeira coluna da linha:

```
1   5 6 7                                                   72
      PROGRAM EXEMPLO
C     PROGRAMA PARA CÁLCULO DE 2 + 2
      A = 2 + 2
      END
```

Qualquer linha de código deve ser escrito até a coluna 72. Após a coluna 72, nenhum código é lido pelo compilador. Se o texto do código chegar até a coluna 72, o restante da linha de código deverá continuar na coluna 7 da linha de baixo. Um caracter qualquer deve ser colocado na coluna 6 para identificar que aquela linha se trata da continuação da linha anterior.

```
1   5 6 7                                                   72
      PROGRAM EXEMPLO
C     CALCULO DE BALANÇO POPULACIONAL
      A = (TAU + BETA)*(TAU + BETA/2.0*(TAU + BETA)*(R – 1.0)*R
     * /(1.0 + TAU + BETA)**R
      END
```

4. TIPOS E DECLARAÇÃO DE VARIÁVEIS

As variáveis podem ser basicamente de quatro tipos: numéricas, caracteres ou lógicas. Os tipos de variáveis do Fortran são:

- **INTEGER**
  números inteiros

- **REAL**
  número real
  suporta valores entre 1.0×10^{-45} até 1.0×10^{45}

- **REAL*8**
  número real em dupla precisão
  suporta valores entre 1.0×10^{-300} até 1.0×10^{300}
  este tipo de variável é o tipo mais usado em engenharia e seu uso deve ser preferido dentre as duas formas de número reais
  programas mais antigos usavam a declaração: DOUBLE PRECISION para este tipo de variável

- **CHARACTER*i**
  sequência alfanumérica com um máximo de *i* caracteres
  não pode ser utilizada em operações matemáticas

- **COMPLEX**
  número complexo

- **LOGICAL**
  variável lógica
  possui dois valores: **.FALSE.** (falso) e **.TRUE.** (verdadeiro)
  este tipo de variável tem sido gradativamente substituído por número inteiros onde **0** se refere a falso e **1** a verdadeiro.

4.1. Declaração de Variáveis

As variáveis podem ser declaradas em grupo ou individualmente. Esta declaração deve vir logo no início do programa.

Individualmente, as variáveis são declaradas listando seus nomes após o tipo da variável, como por exemplo:

```
INTEGER A, B, C
REAL D, E
REAL*8 F, G, H
CHARACTER*10 I
COMPLEX J
```

É importante dar nomes representativos para as variáveis, de forma que se possa identificar facilmente sua função no programa.

EXEMPLO

```
REAL*8 DENS, VISC          para densidade e viscosidade
INTEGER IDX                para índice
```

É comum esquecermos de declarar variáveis no início do programa quando usamos a declaração individual das variáveis. Para evitar este problema, podemos usar a função **IMPLICIT** para declarar um grupo de variáveis baseados em sua letra inicial:

```
IMPLICIT REAL*8 (A-H,O-Z)
```

esta declaração irá fazer com que todas as variáveis iniciadas em A até H e em O até Z sejam número reais em dupla precisão. Como consequência, as variáveis iniciadas em I até N serão número inteiros.

Em geral, as letras I a N são utilizadas para denotar números inteiros e as demais são usadas para números reais (convenção estabelecida), porém isto não impede que se use as letras I a N para números reais e as outras para inteiros.

Utilizar o comando **IMPLICIT** não impede a declaração individual de outras variáveis, sendo que declarações individuais se sobrepõe à declaração feita pelo comando **IMPLICIT**.

EXEMPLO

```
IMPLICIT REAL*8 (A-H,O-Z)
REAL*8 NSA
INTEGER P1
CHARACTER*20 ARQUIVO
```

4.2. Atribuição de Valores

❖ Formas válidas para números inteiros:

I = 0
I = 134
I = -23

❖ Formas válidas para números reais:

A = 3.1415
A = -0.0012
A = .236
A = +5.0E3

A atribuição 5.0E3 quer dizer: 5.0×10^3

❖ Formas válidas para números reais em dupla precisão (REAL*8):

A = 3.1415D0
A = -0.0012D0
A = 2.4D-62
A = +5.0D2

A atribuição 5.0D3 quer dizer: 5.0×10^3

Mesmo para números pequenos é importante a colocação do D0 após o número, pois esta atribuição elimina o risco da variável conter "lixo" em seu final. A falta do D0 pode levar o número 5.0 a ser armazenado na variável como 5.000000342589485 ou mesmo 4.999999993748758, sendo que algumas vezes este "lixo" pode afetar operações com números muito pequenos.

❖ Formas válidas para números complexos:

A atribuição do número complexo deve ser sempre feito entre parênteses, onde o primeiro número é a parte real e o segundo número é a parte imaginária.

C = (1,2)
C = (1.70,-8.948)
C = (+502348E5,.999)

❖ Formas válidas para variável lógica:

L = .TRUE.
L = .FALSE.
estas são as duas únicas opções para a variável lógica

❖ Formas válidas para caracteres

O texto alfanumérico pode ser definido entre apostrofes ou entre aspas

S = "Texto"
S = 'texto'

No caso do apostrofe ser necessário no meio do texto, pode-se usar as formas:

S = "texto's texto"
S = 'texto''s texto'

5. CÁLCULOS MATEMÁTICOS

5.1. Operações Matemáticas Básicas

As operações básicas de adição, subtração, multiplicação, divisão e exponenciação são feitas usando os símbolos da Tabela 5.1.

Tabela 5.1. Símbolos usados para as operações matemáticas

Símbolo	Operação
+	adição
-	subtração
*	multiplicação
/	divisão
**	exponenciação

Uma hierarquia é imposta a estas operações:

1. parênteses
2. exponenciação
3. multiplicação e divisão (o que aparecer primeiro)
4. adição e subtração (o que aparecer primeiro)

EXEMPLO
As equações:

$$A = B + C \cdot D$$

$$A = B^{D} + E$$

$$A = \frac{B \cdot C + D^{E}}{F}$$

seriam programadas como:

```
A = B + C*D

A = B**D + E

A = (B*C + D**E)/F
```


Deve-se sempre ter o cuidado com a hierarquia entre as diferentes operações matemáticas, para se evitar erros de calculo.

EXEMPLO 1
A equação:

$$Z = \left[\frac{(B-C)^{E} + A \cdot B}{F} \right]^{G}$$

deve ser programada como:

```
Z = (((B-C)**E + A*B)/F)**G
```

Se esta mesma equação fosse programada como:

```
Z = (B-C)**E + A*B/F**G
```

a equação que estaria sendo calculada seria:

$$Z = (B-C)^{E} + \frac{A \cdot B}{F^{G}}$$

que por sua vez resultaria num valor muito diferente do que o valor desejado inicialmente.

5.2. Funções Matemáticas

O Fortran possui um conjunto de funções matemáticas para cálculo de logaritmo, seno, tangente, e muitas outras. As principais funções estão listadas abaixo.

ABS(A) — calcula o número absoluto de A
A pode ser um inteiro, real ou complexo

ACOS(A) — calcula o arco coseno de A (resultado em radianos)
A pode ser somente real

ACOSD(A) — calcula o arco coseno de A (resultado em graus)
A pode ser somente real

ASIN(A) — calcula o arco seno de A (resultado em radianos)
A pode ser somente real

ASIND(A) calcula o arco seno de *A* (resultado em graus)
A pode ser somente real
Alguns compiladores podem não aceitar este comando

ATAN(A) calcula o arco tangente de *A* (resultado em radianos)
A pode ser somente real

ATAND(A) calcula o arco tangente de *A* (resultado em graus)
A pode ser somente real
Alguns compiladores podem não aceitar este comando

CEILING(A) retorna o menor número inteiro maior ou igual à *A*
A pode ser somente real
CEILING(4.8) retorna 5.0
CEILING(-2.5) retorna –2.0

COS(A) calcula o coseno de *A* (*A* em radianos)
A pode ser somente real

COSD(A) calcula o coseno de *A* (*A* em graus)
A pode ser somente real
Alguns compiladores podem não aceitar este comando

COSH(A) calcula o coseno hiperbólico de *A*
A pode ser somente real

COTAN(A) calcula a cotangente de *A* (resultado em radianos)
A pode ser somente real

COTAND(A) calcula a cotangente de *A* (resultado em graus)
A pode ser somente real
Alguns compiladores podem não aceitar este comando

EXP(A) calcula a exponencial de *A*
A pode ser somente real

INT (A) converte o valor de *A* em um número inteiro
A pode ser real ou complexo
INT(7.8) retorna o valor 7

LEN(S) retorna o número de caracteres de um texto
S pode ser somente um campo alfanumérico

LOG(A) calcula o logaritmo natural de *A*
A pode ser real ou complexo

LOG10(A) calcula o logaritmo de *A*
A pode ser real ou complexo

SIN(A) calcula o seno de *A* (*A* em radianos)
A pode ser real ou complexo

SIND(A) calcula o seno de *A* (*A* em graus)
A pode ser real ou complexo
Alguns compiladores podem não aceitar este comando

SINH(A) calcula o seno hiperbólico de *A*
A pode ser somente real

TAN(A) calcula a tangente de *A* (*A* em radianos)
A pode ser real ou complexo

TAND(A) calcula a tangente de *A* (*A* em graus)
A pode ser real ou complexo
Alguns compiladores podem não aceitar este comando

TANH(A) calcula a tangente hiperbólica de *A*
A pode ser somente real

Quando o resultado desejado é um numero real em dupla precisão (REAL*8), as funções acima devem ser precedidas por um D, ou seja, a função tangente será **DTAN**(A), a exponencial será **DEXP**(A) e assim por diante.

EXEMPLO 2
A distribuição granulométrica pode ser representada pela equação:

$$X = 1 - \exp\left[-\left(\frac{D}{D^*}\right)^N\right]$$

A programação desta equação é dada por:

```
X = 1.0D0 – DEXP(-(D/DSTAR)**N)
```

6. LEITURA E IMPRESSÃO DE DADOS

A leitura e impressão de dados é uma parte fundamental de muitos programas. Em Fortran, a leitura de dados é feita pelo comando **READ** e a impressão de dados é feito pelo comando **WRITE**.

Tanto o comando **WRITE** quanto o comando **READ** podem seguir um padrão (formato) ou ser livres de formato. Em geral usa-se o formato somente para a impressão de dados.

O comando **READ** tem a forma:

READ(<unidade>,<formato>) <variáveis>

<unidade> é um índice que indica de onde a leitura de dados será feita:
se *****, ela será feita pelo teclado
se um número, ela será feita a partir de um arquivo de dados

<formato> são as regras da formatação da leitura de dados
se *****, o formato é livre (*forma preferencial*)
se uma linha de comando (número da linha), o formato será o que estiver definido na linha de comando especificada
se um formato, seguirá o formato que estiver especificado

<variáveis> lista de variáveis a serem lidas (separadas por vírgulas)

EXEMPLO

READ (*,*) A,B,C	lê as variáveis A, B e C a partir do teclado
READ(2,*) A,B	lê as variáveis A, B a partir do arquivo especificado na unidade 2 (veremos a especificação de arquivos no capítulo 10)

O comando **WRITE** tem a forma:

WRITE(<unidade>,<formato>) <variáveis>

<unidade> é um índice que indica de onde a impressão dos dados será feita:
se *****, imprime as variáveis na tela
se um número, imprime as variáveis em um arquivo de dados

<formato> são as regras da formatação da impressão dos dados
se *****, o formato é livre
se uma linha de comando (número da linha), o formato será o que estiver definido na linha de comando especificada
se um formato, seguirá o formato que estiver especificado

<variáveis> lista de variáveis a serem impressos (separadas por vírgulas)

EXEMPLO

WRITE(*,*) A,B,C	escreve as variáveis A, B e C na tela
WRITE(2,*) A,B	escreve as variáveis A, B no arquivo especificado na unidade 2
WRITE(6,100) A,B	escreve as variáveis A, B no arquivo especificado na unidade 6, seguindo o formato especificado na linha de comando 100. *esta forma de especificação usando linhas de comando numerados tem caído em desuso e seu uso não é mais recomendado*
WRITE(*,'(2F5.2)') A,B	escreve as variáveis A, B na tela, seguindo o formato especificado (2F5.2).

6.1. Formatação dos Dados

O formato de impressão ou leitura é especificado diretamente no comando **WRITE** ou **READ** ou através do comando **FORMAT**.

Os formatos podem ser:

Ix inteiro, onde *x* é o número de caracteres a ser impresso/lido
I3 inteiro com três algarismos
I5 inteiro com cinco algarismos

Fx.y real com *x* algarismos, sendo *y* algarismos reservados para as casas decimais
x deve ser pelo menos igual à *y+1*, uma vez que o ponto decimal também conta como um caracter

F5.2 número real com 2 casas decimais e 2 algarismos antes da virgula
F10.4 número real com 4 casas decimais e 5 algarismos antes da virgula
F5.5 forma não válida, pois não há espaço para as 5 casas decimais mais a virgula

Ex.y número real escrito em notação científica com *x* caracteres, sendo *y* algarismos reservados para as casas decimais. A parte exponencial terá a forma E±00, ocupando 4 caracteres
x deve ser pelo menos igual à *y+5*, uma vez que o ponto decimal e a parte exponencial também contam como um caracteres
E9.2 número real escrito na forma: aa.bbE±cc
E10.1 número real escrito na forma: aaaa.bE±cc

Ax campo alfanumérico com *x* caracteres
A5 campo alfanumérico com 5 caracteres

yX *y* espaços

Forma de Uso

Incorporado ao comando **WRITE**:

```
WRITE(*,'(I2,3X,F5.2,3X,F5.2)') N, A, B
```

neste exemplo, o formato 3X,F5.2 ocorre duas vezes na sequência, e portanto um parênteses pode ser usado para suprimir a repetição do texto:

```
WRITE(*,'(I2,2(3X,F5.2))') N, A, B
```

Usando o comando **FORMAT**:

```
WRITE(*,100) N, A, B
100 FORMAT(I2,3X,F5.2,3X,F5.2)
```

EXEMPLO
Sendo: I = 100
N = 5
A = 1030.56
B = 5.55667
C = 12.563
S = 'MEDIA'


```
WRITE(*,'(I3,2X,F5.2,2X,E8.2)') I,C,A
Imprimiria: 100__12.56__1.03E+03
```
(onde _ se refere a um espaço)

```
WRITE(*,'(A8,F10.3,F10.1)') S,A,B
Imprimiria: MEDIA_____1030.560_______5.6
```

```
WRITE(*,'(I2,3F5.2)') N,A,B,C
Imprimiria: _5XXXXX_5.5612.56
```
(a variável A não será impressa pois o tamanho de sua parte inteira é maior do que o reservado para ela)

EXERCÍCIOS

EXERCÍCIO 1
No controle de qualidade, alguns gráficos de controle se baseiam na média de três valores. Escreva um programa para ler três valores números reais, calcular sua média e imprimir o resultado com duas casas decimais.

EXERCÍCIO 2
Escreva um programa para ler dois números reais, calcular o logaritmo do primeiro número, o coseno do segundo e imprimir o resultado destas duas operações e o produto dos dois resultados.

7. PROCESSOS DECISÓRIOS

7.1. Operadores Relacionais

Toda decisão no Fortran depende de uma comparação entre dois valores ou de um conjunto de comparações.

Os operadores que podem ser usados para comparar duas variáveis são mostradas na Tabela 7.1.

Tabela 7.1. Operadores Relacionais

Símbolo	Operador
==	igual
>	maior que
>=	maior ou igual que
<	menor que
<=	menor ou igual que
/=	diferente

Estes operadores servem para decidir o que será feito dependendo do resultado da comparação. O comando mais utilizado para o processo decisório é o **IF..THEN** e o **IF..THEN..ELSE**.

7.2. IF..THEN

O comando **IF..THEN** tem a seguinte estrutura lógica:

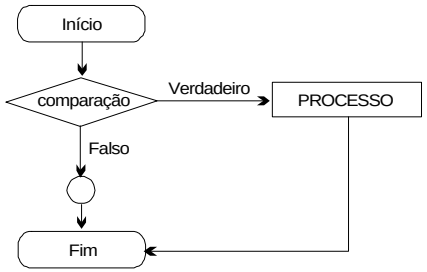


Figura 7.1. Fluxograma lógico do comando **IF..THEN**

No comando **IF..THEN** uma comparação é feita entre dois valores. Se a comparação for verdadeira, um determinado processo é executado, caso contrário o processo não é executado.

Em termos de programação, a estrutura é:

```
IF (<comparação>) THEN
      :
      PROCESSO
      :
END IF
```

onde <comparação> é a expressão usada para testar a condição a ser verificada.

Caso o *PROCESSO* consista somente de uma linha de comando, o comando **IF..THEN** pode ser escrito como:

```
IF (<comparação>) PROCESSO
```

EXEMPLO 1

O coeficiente de arraste (C_D) de partículas sólidas pode ser calculado pela equação:

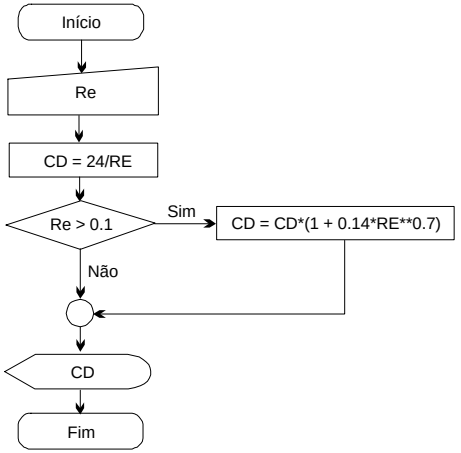
$$C_D = \frac{24}{\mathrm{Re}} \qquad \text{válida para Re < 0,1}$$

Para valores maiores do número de Reynolds (Re), a equação para cálculo do coeficiente de arraste é dado pela equação:

$$C_D = \frac{24}{\mathrm{Re}} \cdot \left(1 + 0{,}14 \cdot \mathrm{Re}^{0{,}7}\right) \qquad \text{válido para Re > 0,1}$$

C_D	coeficiente de arraste
Re	número de Reynolds

O fluxograma de deve ser seguido para este processo é:



cálculo do coeficiente de arraste baseado na fórmula para Re < 0,1. Uma

a execução do programa é desviada para calcular o coeficiente de arraste baseado na segunda equação.
O programa em Fortran para cálculo do coeficiente de arraste será:

```
PROGRAM ARRASTE
IMPLICIT REAL*8 (A-      -Z)
! LEITURA DAS VARIÁVEIS



CD = 24.0D0/RE
IF (RE > 0.1D0) CD = CD*(1.0D0 + 0.14D0*RE**0.7D0)

! IMPRESSÃO DO RESULTADO

END
```


. IF..THEN..ELSE

A estrutura do tem a seguinte lógica:

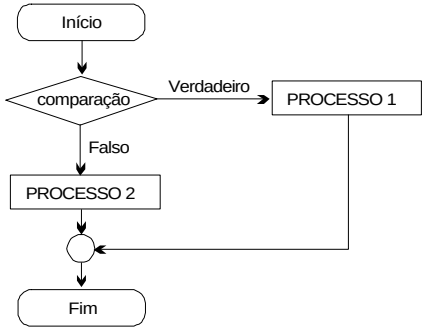


Figura 7.3. Fluxograma lógico do comando **IF..THEN..ELSE**

No comando **IF..THEN..ELSE**, se a comparação for verdadeira, o processo **1** é executado, caso contrário o processo **2** é executado.

Em termos de programação, a estrutura é a seguinte:

```
IF (<comparação>) THEN
      :
      PROCESSO 1
      :
ELSE
      :
      PROCESSO 2
      :
END IF
```

EXEMPLO 2
No cálculo da perda de carga, o fator de atrito é calculado de acordo com o número de Reynolds (Re). Se o número de Reynolds for < 2100, a equação **1** é usada (regime laminar), caso contrário, a equação **2** é utilizada (regime turbulento).

EQ1: $f = \frac{64}{\mathrm{Re}}$ [eq. Dorey-Weisbach]

EQ2: $$f = \left(\frac{1}{2 \cdot \log \frac{e}{D} + 1{,}74} \right)^2$$ [eq. Von Karman]

O fluxograma de deve ser seguido para este processo é:

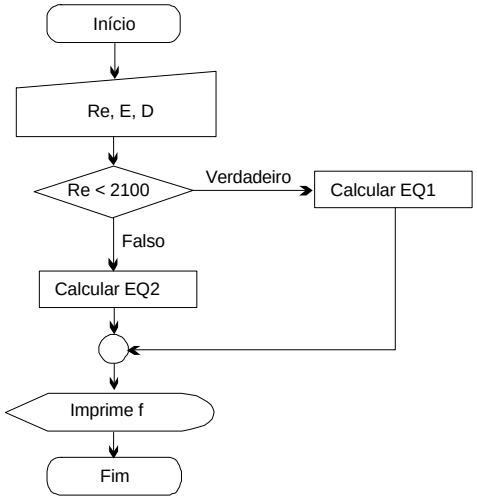


Figura 7.4. Fluxograma lógico para cálculo do fator de atrito

Segundo o fluxograma, após a leitura do número de Reynolds (Re) é feita uma comparação para verificar o se Re é menor do que 2100 (região de escoamento laminar). Caso a condição for verdadeira, o fator de atrito é calculado usando a equação 1, caso contrário o fator será calculado usando a equação 2. Posteriormente, o fator de atrito é impresso.
O programa em Fortran para cálculo do fator de atrito será:

```
PROGRAM FATRITO
IMPLICIT REAL*8 (A-H,O-Z)
! LEITURA DAS VARIÁVEIS
READ(*,*) RE, E, D

! CÁLCULO DO FATOR DE ATRITO
IF (RE < 2100.0D0) THEN
      ! RE < 2100 (ESCOAMENTO LAMINAR)
      FATR = 64.0D0/RE
```



```
ELSE
      ! RE > 2100 (ESCOAMENTO TURBULENTO)
      FATR = (1.0D0/(2.0D0*LOG10(E/D) + 1.74D0))**2.0D0
ENDIF

! IMPRESSÃO DOS RESULTADOS
WRITE(*,*) FATR
END
```

Note que para melhor visualização e entendimento do comando **IF..THEN..ELSE**, o Processo **1** e o Processo **2** estão indentados, ou seja estão uma tabulação a frente do comando **IF**. A indentação do programa é importante para melhor visualizar o fluxo de informações no programa, e é útil principalmente quando o tamanho do código é grande.

7.3.1. Forma Antiga

O Fortran77 não aceitava as declarações dos operadores relacionais na forma de símbolos (==, >, >=, <, <= e /=) e usava palavras chaves para estes operadores. A Tabela 7.2 mostra a equivalência entre os símbolos e as palavras chaves para os operadores relacionais.

Tabela 7.2. Equivalência entre os operadores relacionais no Fortran 90 (forma atual) e Fortran 77 (forma antiga)

Fortran 90	Fortran 77
==	.EQ.
>	.GT.
>=	.GE.
<	.LT.
<=	.GE.
/=	.NE.

A forma usando palavras chaves tem caído em desuso e os novos compiladores tendem a não mais aceitar esta forma.

7.4. Comparação em Conjunto

Algumas vezes, um processo só é executado se duas ou mais condições forem verdadeira (caso **E**) ou se pelo menos uma das condições for verdadeira

(caso **OU**). No primeiro caso, o operador **.AND.** é usado e no segundo caso, o operador **.OR.** é usado.

A tabela 7.2. mostra quais serão os resultados finais das comparações em função dos resultados das comparações individuais.

Tabela 7.2. Resultado das Comparações

Comparação	Resultado
Verdadeiro **.AND.** Verdadeiro	Verdadeiro
Verdadeiro **.AND.** Falso	Falso
Falso **.AND.** Falso	Falso
Verdadeiro **.OR.** Verdadeiro	Verdadeiro
Verdadeiro **.OR.** Falso	Verdadeiro
Falso **.OR.** Falso	Falso
.NOT. Verdadeiro	Falso
.NOT. Falso	Verdadeiro

Em termos de programação, a estrutura é a seguinte:

IF ((<comparação>)**.AND.**(<comparação>)) THEN

e

IF ((<comparação>)**.OR.**(<comparação>)) THEN

EXEMPLO 3

Um dos pontos que mais gera erro de execução em programas é a divisão por zero. Um programa bem estruturado deve prevenir a ocorrência de erros antes do erro ocorrer, alertando o usuário para o problema.

O cálculo da área de troca térmica necessária em trocadores de calor é dado pela equação:

$$Ac = \frac{Q}{Uc \cdot \Delta T}$$

Ac	área de troca térmica
Q	calor trocado
Uc	coeficiente de troca térmica
ΔT	diferença de temperatura

No caso, uma divisão por zero pode ocorrer se Uc ou ΔT forem iguais a zero. Um programa bem feito deve prever esta possibilidade e impedir o erro antes

que o mesmo ocorra. Um fluxograma lógico para o cálculo da área de troca térmica com predição de erros teria a estrutura:

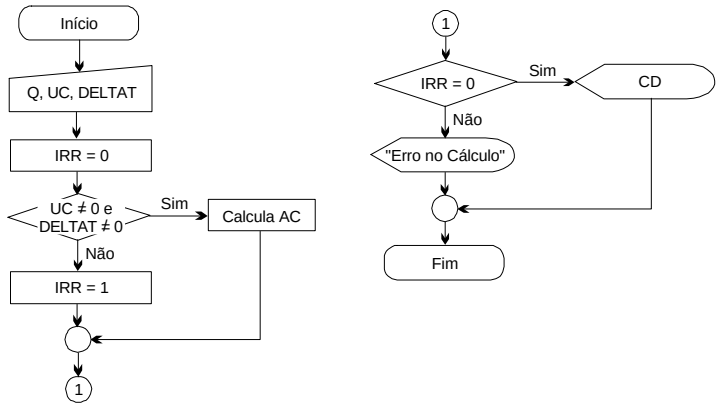


Figura 7.5. Fluxograma lógico para cálculo do fator de atrito

Segundo o fluxograma, após a leitura das variáveis, uma variável de controle de erro (**IRR**) é introduzida e inicializada. Esta variável é definida com o valor 0 (zero) para sem erro de execução, e pode vir a receber um valor qualquer durante a execução do programa se um possível erro ocorreu ou poderia ocorrer (e foi impedido).

Seguindo o fluxograma, uma comparação para verificar se Uc ou ΔT (**DELTAT**) são diferentes de zero é feita. Caso a condição seja verdadeira, a área de troca térmica é calculada, caso contrário a variável de controle de erro recebe um valor diferente de zero, indicando a ocorrência de um erro. Posteriormente, uma nova comparação é feita, verificando o valor da variável de controle de erro. Se o valor desta variável for 0 (zero), a área de troca térmica é impressa, caso contrário uma mensagem de erro é apresentada.

O programa em Fortran para o cálculo da área de troca térmica será:

```
PROGRAM TROCTERM
IMPLICIT REAL*8 (A-H,O-Z)
! LEITURA DAS VARIÁVEIS
READ(*,*) Q,UC,DELTAT

! DEFINIÇÃO DA VARIÁVEL DE ERRO
IRR = 0

! CÁLCULO DA ÁREA DE TROCA TÉRMICA
IF ((UC /= 0.0D0).AND.(DELTAT /= 0.0D0)) THEN
      AC = Q/(UC*DELTAT)
```

```
ELSE
        ! PODERIA OCORRER DIVISÃO POR ZERO
        IRR = 1
ENDIF

! IMPRESSÃO DOS RESULTADOS
IF (IRR == 0) THEN
        WRITE(*,*) AC
ELSE
        WRITE(*,*) 'ERRO NO CÁLCULO – DIVISÃO POR ZERO'
ENDIF
END
```

7.5. Processo Decisório por Faixa ou Classes

Índices podem ser usados para desviar a execução do programa para diferentes processos, dependendo do valor deste índice. Para esta forma de processo decisório usamos o comando **SELECT CASE** que tem a seguinte estrutura lógica:

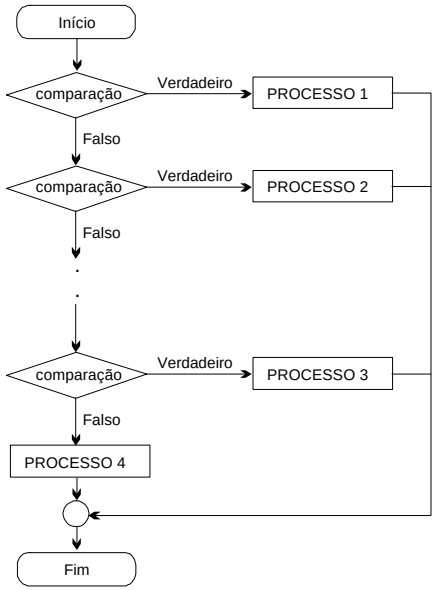


Figura 7.6. Fluxograma lógico do comando **SELECT CASE**.

No comando **SELECT CASE**, uma variável é comparada com vários valores. Quando a comparação resultar em verdadeiro o processo relativo àquela condição é executado. Quando nenhuma comparação resultar em verdadeiro, o processo relativo a condição **CASE ELSE** é executado.

Em termos de programação, a estrutura é a seguinte:

```
SELECT CASE (<variável>)
        CASE (a)
                PROCESSO 1
        CASE (b)
                PROCESSO 2
        :
        CASE (n)
                PROCESSO 3
        CASE ELSE
                PROCESSO 4
END SELECT
```

É importante notar que o **SELECT CASE** só pode ser usado com número inteiros. Tanto a variável, quanto os valores de **a, b, n** devem ser número inteiros.

Uma faixa de valores pode ser usada nos Cases, como por exemplo: CASE (1:5) significa uma faixa de valores de 1 a 5.

A condição **CASE ELSE** é opcional e pode ser omitida do comando **SELECT CASE**.

EXEMPLO 4

O projeto de equipamentos de adsorção requer a seleção de um adsorvente e informações relacionados à transferência de massa para a superfície do adsorvente. A seleção do adsorvente requer informações para descrever a capacidade de equilíbrio do adsorvente à temperatura constante (isoterma de adsorção). Vários tipos de isotermas de adsorção existem e um programa genérico ou que irá testar vários tipos de isotermas deve ter um sistema de seleção da isoterma que será usada.

EQ1: $q_{ADS} = \dfrac{Q \cdot K \cdot C}{1 + K \cdot C}$ [eq. Langmuir]

EQ2: $q_{ADS} = K \cdot C^{-n}$ [eq. Freundlich]

EQ3: $q_{ADS} = \dfrac{Q \cdot K \cdot p}{(1 + K \cdot p + p/P) \cdot (1 - p/P)}$ [eq. BET]

O fluxograma para a escolha da isoterma depende da escolha do tipo de isoterma pelo usuário. Esta escolha é armazenada em uma variável de controle (**IDX**) que será usada na decisão para selecionar a equação que será usada.

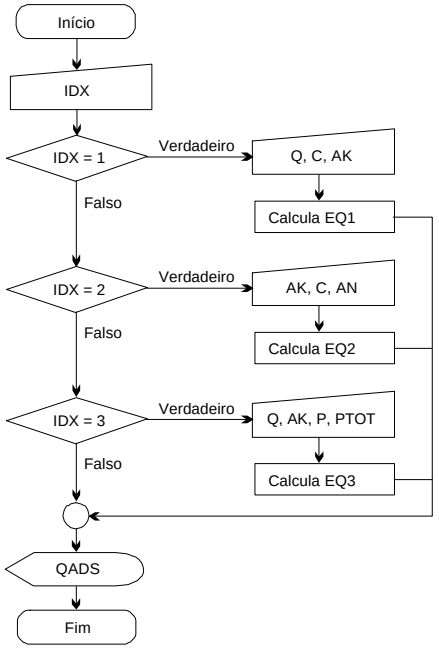


Figura 7.7. Fluxograma lógico para calculo da isoterma de adsorção

O programa em Fortran para cálculo da isoterma será:

```
PROGRAM ISOTERMA
IMPLICIT REAL*8 (A-H,O-Z)

! LEITURA DO TIPO DE ISOTERMA
READ(*,*) IDX

! CÁLCULO DA ISOTERMA
SELECT CASE (IDX)
```



```
      CASE (1)    ! ISOTERMA DE LANGMUIR
            READ(*,*) Q, C, AK
            QADS = Q*C*AK/(1.0D0 + C*AK)
      CASE (2)    ! ISOTERMA DE FREUNDLICH
            READ(*,*) AK, C, AN
            QADS = AK*C**(-AN)
      CASE (3)    ! ISOTERMA BET
            READ(*,*) Q, AK, P, PTOT
            QADS = Q*P*AK/((1.0D0 + AK*P + P/PTOT)*(1.0D0 – P/PTOT))
END SELECT

! IMPRESSÃO DO RESULTADO
WRITE(*,*) QADS
END
```

EXERCÍCIOS

EXERCÍCIO 1
Desenvolva um programa para calcular a perda de carga usando as fórmulas de Fair-Whipple-Hsiao.

EQ1: $PC = 0{,}00086 \dfrac{Q^{1{,}75}}{D^{4{,}75}}$ [para água fria]

EQ2: $PC = 0{,}0007 \dfrac{Q^{1{,}75}}{D^{4{,}75}}$ [para água quente]

D	diâmetro do tubo
L	comprimento do tubo
PC	perda de carga
Q	vazão de água

EXERCÍCIO 2
Refaça o Exemplo 2 inserindo no programa um sistema para detecção de erros devido a divisão por zero. Crie um sistema para apresentar ao usuário uma mensagem de erro indicando qual variável apresentou o problema.

EXERCÍCIO 3

Desenvolva um programa para calcular a pressão de vapor de uma substância onde o usuário seleciona a equação pela qual a pressão de vapor será calculada.
Equações:

EQ1: $$P_{vap} = P_C \cdot \exp\left[\frac{A \cdot X + B \cdot X^{1,5} + C \cdot X^3 + D \cdot X^6}{1 - X}\right]$$

$$X = 1 - \frac{T}{T_C}$$

EQ2: $$P_{vap} = \exp\left[A - \frac{B}{T + C}\right]$$

EQ3: $$P_{vap} = \frac{P_C}{10^{Z3}}$$

$$Z1 = 5{,}808 + 4{,}93 \cdot v$$

$$Z2 = \frac{36}{T_R} - 35{,}0 - T_R{}^6 + 42 \cdot \ln(T_R)$$

$$Z3 = 0{,}118 \cdot Z2 - 7 \cdot \log(T_R) + (Z1 - 7{,}0) \cdot \left(0{,}0364 \cdot Z2 - \log(T_R)\right)$$

$$T_R = \frac{T}{T_C}$$

A,B,C,D	parâmetros da equação
Pvap	pressão de vapor
P_C	pressão crítica
T_C	temperatura crítica
T_R	temperatura relativa
ω	fator acêntrico

8. LOOPS

Loops são rotinas cíclicas nas quais um processo é executado por um número pré-determinado de vezes ou enquanto uma condição de permanência no *loop* continue sendo satisfeita.

8.1. Loops Limitados

Um processo pode ser executado por um número limitado de vezes usando o comando **DO..ENDDO**.

Este comando tem a seguinte estrutura lógica:

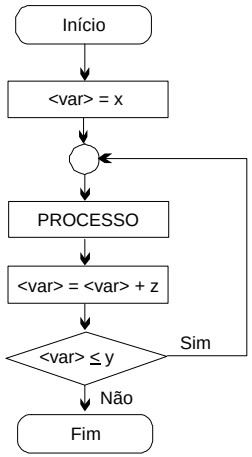


Figura 8.1. Fluxograma lógico do comando **DO..ENDDO**.

No comando **DO..ENDDO**, a variável de controle (<var>) é iniciada com um valor *x*. Após a execução do processo, a variável de controle tem seu valor incrementado com o valor *z*. Uma comparação é feita para ver se a variável de controle atingiu o valor máximo definido para ela (*y*). Se o valor máximo ainda não foi atingido, o processo é executado novamente, até que a variável de controle seja maior que *y*.

Em termos de programação, a estrutura é:


```
DO <var> = x,y,z
      :
      PROCESSO
      :
ENDDO
```

x	valor inicial de <var>
y	valor final de <var>
z	incremento em <var> a cada iteração

O passo de incremento da variável de controle (<var>) pode ser maior que 1 ou até mesmo negativo, porém deve ser um número inteiro. Caso o passo seja negativo, *x* deve ser maior do que *y*.

Se o incremento for igual a 1, o valor de *z* pode ser omitido e o comando **DO..ENDDO** toma a forma:

```
DO <var> = x,y
      :
      PROCESSO
      :
ENDDO
```

EXEMPLO 1

Quando são produzidos, os polímeros apresentam uma distribuição de pesos moleculares. A distribuição pode ser calculada pela função:

$$W(r) = (t+b)\cdot\left[t + \frac{b}{2}\cdot(t+b)\cdot(r-1)\right]\cdot\frac{r}{(1+t+b)^r}$$

$$t = \frac{ktd}{kp\cdot[M]} + \frac{ktm}{kp} + \frac{ktx\cdot[X]}{kp\cdot[M]}$$

$$b = \frac{ktc}{kp\cdot[M]}$$

kfm	constante de transferência para monômero
kfx	constante de transferência para CTA
kp	constante de propagação
ktc	constante de terminação por combinação
ktd	constante de term. por desproporcionamento
r	comprimento de cadeia

W	fração de cadeias produzidas
[M]	concentração de monômero
	CM – concentração de monômero (no programa)
[X]	concentração de CTA
	CX – concentração de CTA (no programa)

Para obter dados para imprimir a distribuição de pesos moleculares, pode-se usar um *loop* para gerar os dados da fração de cadeias formadas em função do comprimento de cadeia do polímero. O fluxograma a ser seguido será:

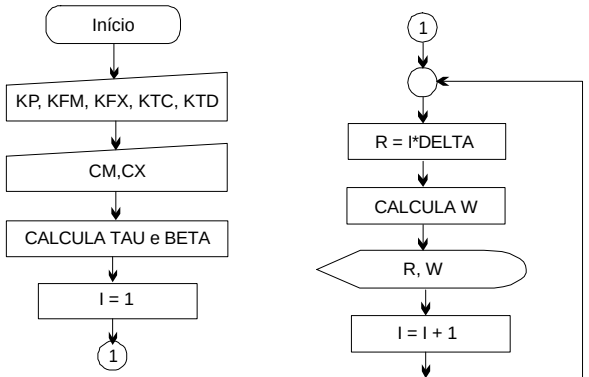


Figura 8.2. Fluxograma lógico para geração de dados para a distribuição de pesos moleculares de polímeros.

Segundo o fluxograma, primeiramente os parâmetros cinéticos e as concentrações são lidas. Os parâmetros τ e β da equação são calculados e inicia-se o *loop* para calculo da fração de pesos moleculares (**W**) em função do comprimento de cadeia (**R**). Cem pontos devem ser gerados, e portanto a variável de controle **I** deve variar entre 1 e 100.
No interior do *loop* o valor de **R** é calculado em função do valor de **I** e portanto **R** pode ser incrementado 100 vezes (assim como **I**), porém seu incremento poderá ser maior ou menor do que 1 dependendo do valor de **DELTA**. Calcula-se a fração de pesos moleculares (**W**) e **R** e **W** são impressos.

Em termos de programação, a estrutura é a seguinte:


```
PROGRAM DPM
IMPLICIT REAL*8 (A-H,K,O-Z)

! LEITURA DAS VARIÁVEIS
READ(*,*) KP, KFM, KFX, KTC, KTD
READ(*,*) CM, CX

! CÁLCULO DOS PARÂMETROS TAU E BETA
TAU = KTD/(KP*CM) + KTM/KP + KTX*CX/(KP*CM)
BETA = KTC/(KP*CM)

! CÁLCULO DA DISTRIBUIÇÃO DE PESOS MOLECULARES
DO I = 1,100
      R = I*1000.0D0
      W = (TAU + BETA)*(TAU + BETA/2.0D0*(TAU + BETA)*(R – 1.0D0))* &
            (R/(1.0D0 + TAU + BETA)**R)
      WRITE(*,*) R,W
ENDDO
END
```

8.1.1. Forma Antiga

No Fortran 77, os *loops* eram controlados pelo comando **DO..CONTINUE**, que tem como estrutura de programação:

```
DO <linha> <var> = x,y,z
      :
      PROCESSO
      :
<linha> CONTINUE
```

onde <linha> é o número da identificação de linha onde o **CONTINUE** está localizado

O *loop* do Exemplo 1 seria programado como:

```
      DO 100 I = 1,100
            R = I*1000.0D0
            W =
            WRITE(*,*) R,W
100   CONTINUE
```

Esta forma de controle de loop caiu em desuso e não deve ser mais utilizada.

8.2. Loops por Decisão

Os *loops* podem ocorrer enquanto uma condição continue sendo atendida, usando o comando **DO WHILE**. Este comando tem como estrutura lógica:

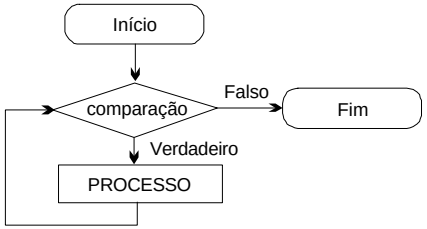


Figura 8.3. Fluxograma lógico do comando **DO WHILE**.

No comando **DO WHILE**, o programa entra e continua em *loop* até que a condição responsável pelo *loop* continue sendo atendida.

Em termos de programação, a estrutura é:

```
DO WHILE (<comparação>)
      :
      PROCESSO
      :
ENDDO
```

EXEMPLO 2

Sistemas de busca por mínimos e zeros de funções podem usar o esquema de *loop* por desição para determinar quando parar a busca do mínimo e/ou zero de função.

A equação de Colebrook é uma das melhores equações para calcular o fator de atrito em tubulações industriais, porém esta equação é uma função intrínseca e requer um sistema iterativo para calcular o fator de atrito. O método de bisseção pode ser usado para esta operação.

$$\frac{1}{f^{0,5}} = -2 \cdot \log\left[\frac{e}{3{,}7065 \cdot D} + \frac{2{,}5226}{\mathrm{Re} \cdot f^{0,5}}\right] \qquad \text{[eq. Colebrook]}$$

Esta equação pode ser rearranjada para:

$$e = 0 = \frac{1}{f^{0,5}} + 2 \cdot \log\left[\frac{e}{3{,}7065 \cdot D} + \frac{2{,}5226}{\mathrm{Re} \cdot f^{0,5}}\right] \qquad \text{[eq. 2]}$$

f	fator de atrito
ε/D	rugosidade relativa
Re	número de Reynolds

Analisando a equação tem-se que se “chutarmos” um valor inicial para **f**, se a equação 2 for negativa, **f** estará superestimado, caso contrário estará subestimado. Portanto pode-se fazer um sistema de bisseção que leve esta informação em conta para calcular o fator de atrito.

É difícil achar **e** = 0,0 para a equação 2 e portanto pode-se parar a iteração de busca por **f** quando **e** estiver dentro de uma tolerância, por exemplo **e** ± 0,001.

Como limites da busca na bisseção, pode-se usar os limites do fator de atrito apresentado no gráfico de Moody, e portanto **f** deverá estar entre 0,007 e 0,1.

O fluxograma a ser seguido será:

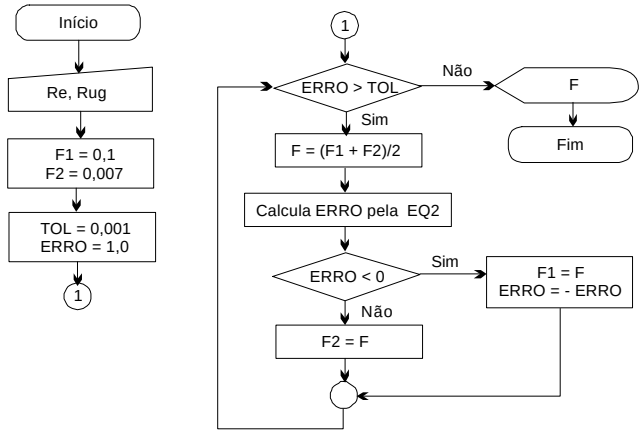


Figura 8.4. Fluxograma lógico para cálculo do fator de atrito.

Segundo o fluxograma, após a leitura e inicialização das variáveis, o algoritmo entra no *loop* para cálculo do fator de atrito. No *loop*, o fator de atrito é calculado baseado na teoria do método da bisseção. O erro é calculado e dependendo do seu valor, pode-se inferir se o valor atual do fator de atrito está

super ou subestimado e baseado neste resultado, define-se os novos limites superior e inferior para o valor do fator de atrito.

Em termos de programação, a estrutura é a seguinte:

```
PROGRAM FATRITO
IMPLICIT REAL*8 (A-H,O-Z)

! LEITURA DAS VARIÁVEIS
READ(*,*) RE, RUG

! INICIALIZAÇÃO DAS VARIÁVEIS
F1 = 0.1D0
F2 = 0.007D0
ERRO = 1.0D0
TOL = 0.001D0

! CÁLCULO DO FATOR DE ATRITO VIA MÉTODO DA BISSEÇÃO
DO WHILE (ERRO > TOL)
      F = (F1 + F2)/2.0D0
      ERRO = F**(-0.5D0) + 2.0D0*DLOG10(RUG/3.7065D0 &
            + 2.5226/(RE*F**0.5D0))
      IF (ERRO < 0.0D0) THEN
            F1 = F
            ERRO = - ERRO
      ELSE
            F2 = F
      ENDIF
ENDDO

! IMPRESSÃO DOS RESULTADOS
WRITE(*,*) F
END
```

Note que no programa, a variável **ERRO** é inicializada com um valor maior do que **TOL**, de forma que o programa entre no *loop*. Se a variável **ERRO** não fosse inicializada, ou fosse inicializada com zero, o programa não entraria no *loop* uma vez que a condição de entrada e permanência no *loop* não seria satisfeita.

8.3. Loops Infinitos

O uso de *loops* infinitos é uma opção alternativa aos *loops* decisórios, onde a decisão de saída é feita por um **IF** interno e a saída do *loop* é feita pelo comando **EXIT**.

Este tipo de *loop* tem estrutura lógica:

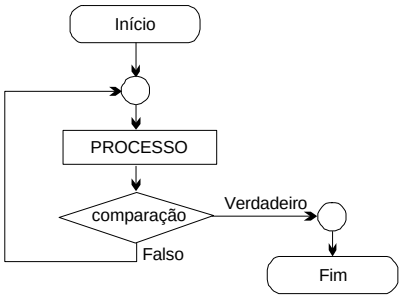


Figura 8.5. Fluxograma lógico do *loop* infinito.

Em termos de programação, a estrutura é:

```
DO
      :
      PROCESSO
      :
      IF (<comparação>) EXIT
ENDDO
```

EXEMPLO 3

Se o exemplo 2 for utilizado com um loop infinito, tem-se o seguinte fluxograma:

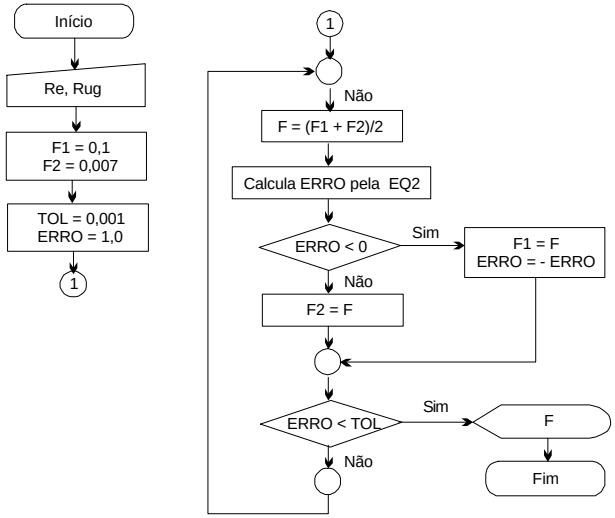


Figura 8.6. Fluxograma lógico para cálculo do fator de atrito.

Segundo o fluxograma, após a leitura e inicialização das variáveis, o algoritmo entra no *loop* para cálculo do fator de atrito. No *loop*, o fator de atrito é calculado pelo método de bisseção. O erro é calculado e dependendo do valor do erro, pode-se inferir se o valor atual do fator de atrito está super ou subestimado e baseado neste resultado, define-se os novos limites superior e inferior para o valor do fator de atrito.
A comparação entre os valores do erro (**ERRO**) e da tolerância requerida (**TOL**) é feita no final do *loop*. Caso o erro seja menor do que a tolerância, a execução do programa sai do *loop* usando o comando **EXIT**.

Em termos de programação, a estrutura é a seguinte:

```
PROGRAM FATRITO2
IMPLICIT REAL*8 (A-H,O-Z)

! LEITURA DAS VARIÁVEIS
READ(*,*) RE, RUG

! INICIALIZAÇÃO DAS VARIÁVEIS
F1 = 0.1D0
F2 = 0.007D0
TOL = 0.001D0

! CÁLCULO DO FATOR DE ATRITO VIA MÉTODO DA BISSEÇÃO
```



```
DO
        F = (F1 + F2)/2.0D0
        ERRO = F**(-0.5D0) + 2.0D0*DLOG10(RUG/3.7065D0 &
                + 2.5226/(RE*F**0.5D0))
        IF (ERRO < 0.0D0) THEN
                F1 = F
                ERRO = - ERRO
        ELSE
                F2 = F
        ENDIF
        IF (ERRO < TOL) EXIT
ENDDO

! IMPRESSÃO DOS RESULTADOS
WRITE(*,*) F
END
```

8.4. CYCLE

O comando **CYCLE** interrompe a execução do restante do ciclo (*loop*), retornando a execução do programa para o início do *loop*. Este comando tem como estrutura lógica:

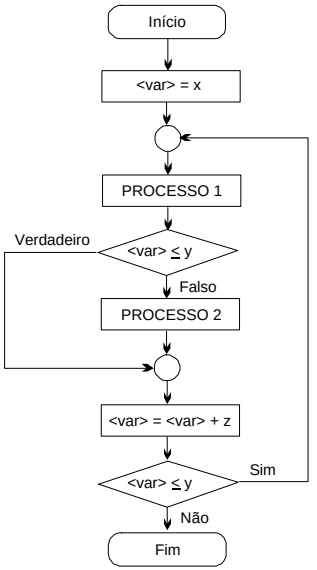


Figura 8.7. Fluxograma lógico de um *loop* usando o comando **CYCLE**.

Segundo a estrutura lógia do comando **CYCLE**, após a execução do *PROCESSO 1*, uma comparação é feita e se esta condição for satisfeita a variável de controle <var> é incrementada e a execução do programa volta para o início do *loop*. Caso a condição não seja satisfeita, o *PROCESSO 2* é executado.

Em termos de programação, a estrutura é:

```
DO <var> = x,y,z
      :
      PROCESSO 1
      :
      IF (<comparação>) CYCLE
      :
      PROCESSO 2
      :
ENDDO
```

EXERCÍCIOS

EXERCÍCIO 1

Os ciclones são projetados para remover partículas até um certo tamanho de corte. Para o projeto do equipamento, a granulometria das partículas a serem processadas deve ser conhecida.

A equação de granulometria é dada pela equação de Rosin-Rammler-Bennett:

$$E(X) = 1 - \exp\left[-\left(\frac{X}{D^*}\right)^n\right]$$

D*	diâmetro de corte (constante)
E	distribuição acumulada do tamanho de partículas
n	parâmetro da equação
X	diâmetro da partícula (variável)

Desenvolva um algoritmo e programa para gerar 50 pontos para a curva de granulometria do sistema (gráfico **E** em função de **X**).

EXERCÍCIO 2

Desenvolva um algoritmo e programa para gerar pontos para o gráfico de performance de um sedimentador, obtendo pontos de 50 em 50 unidades da variável **X**. Sendo que o fim da geração de dados para o gráfico for quando o valor do fluxo de sedimentação (G) for 1000 vezes menor do que o valor máximo da função (G_{max}).

A equação do sedimentador é dada pela equação:

$$G = X \cdot 10^{(A \cdot X + B)}$$

onde A = -0.0142
B = 0.370

G	fluxo de sedimentação
X	concentração de sólidos

O programa deve obter G_{max} (maior valor da função) usando a mesma função acima.

9. VETORES E MATRIZES

Em Fortran, os vetores e matrizes começam a ser contados a partir da posição 1.

Um vetor com 5 posições tem forma:

1	2	3	4	5

Uma matriz 3x5 tem forma:

1,1	1,2	1,3	1,4	1,5
2,1	2,2	2,3	2,4	2,5
3,1	3,2	3,3	3,4	3,5

9.1. Tipos de Vetores e Matrizes

Todos os tipos de variáveis podem ser usados como vetores ou matrizes. Portanto podemos ter os tipos: INTEGER, REAL, REAL*8, COMPLEX.

9.2. Declaração de Vetores

Os vetores ou matrizes podem ser declarados de duas formas: através das palavras chave de declaração de tipos de variáveis, ou através do comando **DIMENSION**.

- Através de declaração de tipo

```
REAL*8 A(10)        vetor com 10 campos
REAL*8 B(3,5)       matriz 3x5
```

- Através do comando **DIMENSION**

```
IMPLICIT REAL*8 (A-H,O-Z)
DIMENSION A(10), B(3,5)
```

9.3. Atribuição de Valores

Valores podem ser atribuídos aos vetores e matrizes de forma individual, por faixa e total.

- forma individual

```
A(2) = 10          atribui o valor 10 ao campo 2
```

- por faixa

```
A(2:5) = 10        atribui o valor 10 aos campos 2 até 5, ou seja,
                       A(2) = A(3) = A(4) = A(5) = 10
B(1:3,3:4) = 10    atribui o valor 10 aos campos
                       B(1,3)   B(2,3)   B(3,3)
                       B(1,4)   B(2,4)   B(3,4)
```

- total

```
A = 10             atribui o valor 10 a todos os campos da variável A,
                       ou seja, A(1) = A(2) = ... = A(n) = 10
```

9.4. Operações com Vetores e Matrizes

Vetores e matrizes podem ser somados, subtraídos, multiplicados e divididos entre si, desde que sejam de mesmo tamanho:

```
REAL A(10), B(10), C(10)
A = B                  é o mesmo que fazer A(1) = B(1),
                       A(2) = B(2), etc...
A = B + C              é o mesmo que fazer A(1) = B(1) + C(1),
                       A(2) = B(2) + C(2), etc...
```

Quando os vetores ou matrizes forem e tamanhos diferentes, uma faixa comum poderá ser somada ou subtraída:

```
REAL A(10), B(5), C(20)
A(1:5) = B(1:5)            é o mesmo que fazer A(1) = B(1),
                           A(2) = B(2), até A(5) = B(5).
A(3:5) = B(3:5) + C(3:5)   é o mesmo que fazer A(3) = B(3) + C(3),
                           A(4) = B(4) + C(4) e A(5) = B(5) + C(5)
```

Se a faixa for diferente ou maior do que o tamanho do vetor ou matriz, ocorrerá um erro na execução do programa:

```
REAL A(10), B(5), C(20)
A(1:3) = B(3:5)            faixas diferentes
A(1:10) = B(1:10) + C(1:10)  B não tem 10 campos
```

Os vetores e matrizes também podem ser somados, subtraídos, divididos e multiplicados por número escalares:

```
REAL A(5), B(5)
A = B + 5                  é o mesmo que fazer A(1) = B(1) + 5,
                           A(2) = B(2) + 5, etc...
A(1:3) = 2*B(1:3)          é o mesmo que fazer A(1) = 2*B(1),
                           A(2) = 2*B(2), A(3) = 2*B(3)
```

Os campos individuais, por sua vez, podem sofrer qualquer tipo de operação:

```
A(1) = B(3) + 2
A(2) = B(3)*C(5)*3 + C(1)
D(1,3) = E(1,3) + F(2,7)
```

9.5. Funções Intrínsecas

O Fortran possui um conjunto de funções matemáticas para cálculo com matrizes. As principais funções estão listadas abaixo.

DOT_PRODUCT(A,B) — calcula o produto vetorial entre A e B
A e B são dois vetores numéricos de igual tamanho

MATMUL (A,B) — calcula o produto matricial entre A e B
A e B são duas matrizes numéricas
se A tem tamanho (n,m) e B tem tamanho (m,k), a matriz resultante terá tamanho (n,k)

MAXVAL (A) — retorna o maior valor do vetor A
A é um vetor numérico

MINVAL (A) — retorna o menor valor do vetor A
A é um vetor numérico

SUM(A) — calcula o somatório dos valor do vetor A
A é um vetor numérico

9.6. Loops com Vetores e Matrizes

Os *loops* podem ser usados com vetores e matrizes da mesma forma como foi abordado no Capítulo 8. A diferença é que os *loops* podem ser usados para controlar o campo corrente do vetor ou matriz que será usado em algum processo ou calculo.

EXEMPLO 1
Um uso simples dos *loops* com vetores e matrizes pode ser para controlar o campo do vetor ou matriz que será usado em algum cálculo.

```
DO I = 1,10
      A(I) = B(I,2)*C(I)
      SUM = SUM + C(I)
ENDDO
```

EXEMPLO 2
Quando se trabalha com análise estatística de dados experimentais, é comum ser necessário calcular a média e desvio padrão destes dados. O cálculo da média e desvio padrão de um conjunto de dados pode ser feito seguindo o fluxograma:

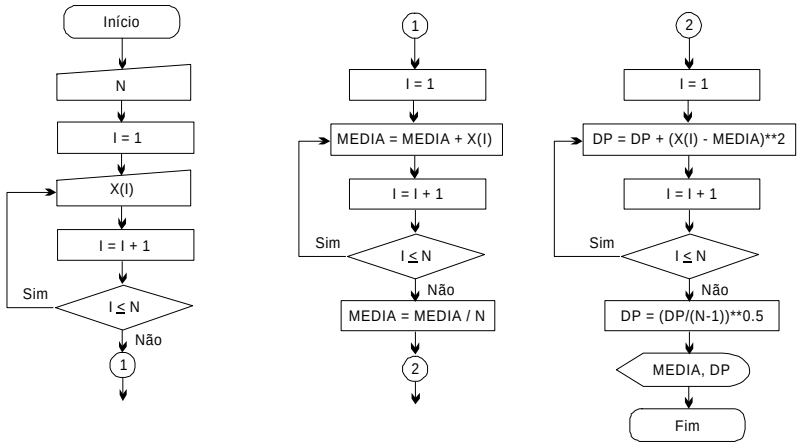


Figura 9.1. Fluxograma para cálculo da média e desvio padrão de um conjunto de dados

O programa para em Fortran para realizar o cálculo teria a forma:

```
PROGRAM ESTAT
IMPLICIT REAL*8 (A-H,O-Z)
DIMENSION X(10)

! LEITURA DE VARIÁVEIS
WRITE(*,*) 'NUMERO DE DADOS A SEREM LIDOS: (MAXIMO = 10)'
READ(*,*) N
WRITE(*,*) 'ENTRE COM OS DADOS DO CONJUNTO'
DO I = 1,N
      READ(*,*) X(I)
ENDDO

! CALCULO DA MÉDIA
DO I = 1,N
      MEDIA = MEDIA + X(I)
ENDDO
MEDIA = MEDIA/N

! CALCULO DO DESVIO PADRÃO
DO I = 1,N
      DP = DP + (X(I) – MEDIA)**2.0D0
ENDDO
DP = (DP/(N – 1.0D0))**0.5D0

! IMPRESSÃO DOS RESULTADOS
WRITE(*,*) 'MEDIA = ', MEDIA
WRITE(*,*) 'DESVIO PADRAO = ', DP
END
```

9.7. Processos Decisórios com Vetores e Matrizes

Assim como para variáveis escalares, os campos individuais dos vetores e matrizes podem ser usados pelos comandos **IF..THEN..ELSE** e **SELECT CASE**.

EXEMPLO 3
Em processos de estimativa de parâmetros, os valores dos parâmetros podem ser armazenados em vetores e estes valores podem estar sujeitos a limites superiores e inferiores.
No caso, os valores dos parâmetros estão armazenados no vetor **THETA**, os limites superiores no vetor **TMAX** e os limites inferiores no vetor **TMIN**.

O fluxograma a ser seguido tem a seguinte estrutura:

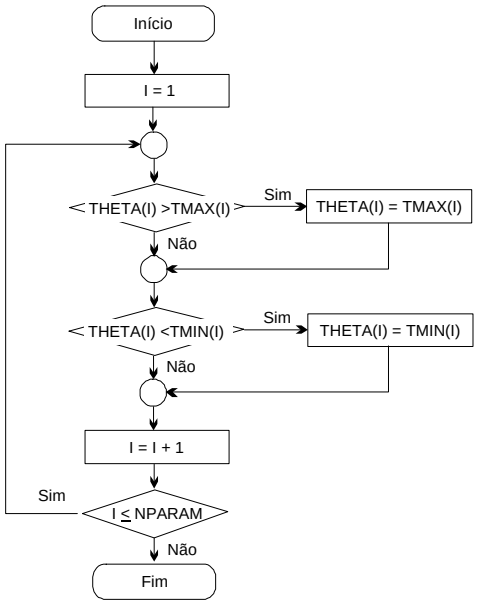


Figura 9.2. Fluxograma lógico para controle de parâmetros.

Segundo o fluxograma, primeiramente o valor de **THETA(I)** é comparado com **TMAX(I)**, recebendo seu valor se **THETA(I)** for maior do que **TMAX(I)**. Posteriormente o valor de **THETA(I)** é comparado com **TMIN(I)**, recebendo seu valor se **THETA(I)** for menor do que **TMIN(I)**. Esta operação é repetida para todos os parâmetros (de 1 até **NPARAM** – onde **NPARAM** é o número total de parâmetros).
Em termos de programação, a estrutura é:

```
DO I = 1,NPARAM
      IF (THETA(I) > TMAX(I)) THETA(I) = TMAX(I)
      IF (THETA(I) < TMIN(I)) THETA(I) = TMIN(I)
ENDDO
```

Como apresentado no exemplo, o processo decisório usa os valores individuais dos campos do vetor e geralmente a ação também afeta somente os valores individuais do vetor ou matriz. Isto ocorre com os comando **IF..THEN..ELSE** e **SELECT CASE**.

9.7.1. WHERE

O comando **WHERE** afeta todo o vetor ou matriz e geralmente é usado para realizar alguma operação matemática com o vetor ou matriz.

Este comando tem a seguinte estrutura lógica:

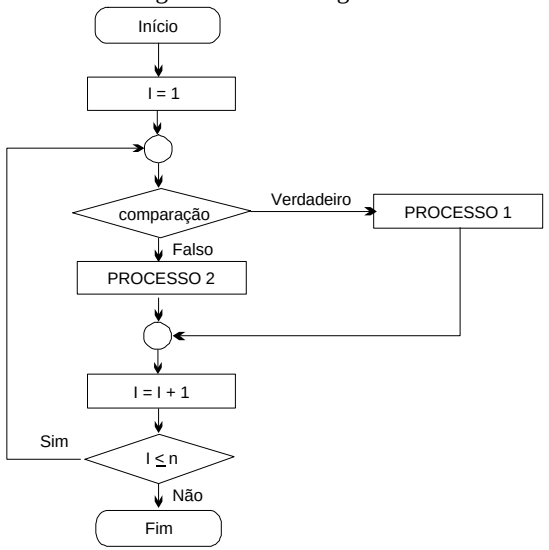


Figura 9.3. Fluxograma lógico do comando **WHERE**. No fluxograma, *n* é o número de campos no vetor ou matriz.

O comando **WHERE** tem uma lógica parecida com a do comando **IF..THEN..ELSE**. A diferença é que a comparação afeta todos os campos do vetor ou matriz e não somente um único campo (como ocorreria com o **IF..THEN..ELSE**).

Em termos de programação, a estrutura é:

```
WHERE (<comparação>)
      :
      PROCESSO 1
      :
ELSEWHERE
      :
      PROCESSO 2
      :
ENDWHERE
```


onde <comparação> é a expressão usada para testar a condição a ser verificada.

Caso o *PROCESSO* consista de somente uma linha de comando, o comando **WHERE** pode ser escrito como:

```
WHERE (<comparação>) PROCESSO
```

O comando **WHERE** é equivalente ao uso de um comando **DO..ENDDO** com a comparação sendo feita por um comando **IF..THEN..ELSE**:

```
DO I = 1,N
      IF (<comparação>) THEN
            :
            PROCESSO 1
            :
      ELSE
            :
            PROCESSO 2
            :
      ENDIF
ENDDO
```

Neste caso, a variável **I** deve controlar o campo do vetor/matriz.

EXEMPLO 4

No caso de divisão dos elementos de dois vetores, a divisão não pode ocorrer se o valor em alguma posição do vetor divisor for zero. O comando **WHERE** pode ser usado para executar a divisão somente quando o valor divisor for zero.

Em termos de programação, a estrutura é:

```
PROGRAM DIVVET
IMPLICIT REAL*8 (A-H,O-Z)
DIMENSION A(5), B(5), C(5)

! LEITURA DAS VARIÁVEIS
DO I = 1,5
      READ(*,*) A(I), B(I)
ENDDO

! CÁLCULO DA DIVISÃO
C = 0.0D0
WHERE (A /= 0.0D0) C = B/A
```

```
! IMPRESSÃO DOS RESULTADOS
DO I = 1,5
      WRITE(*,*) I, C(I)
ENDDO
END
```

Se **A** e **B** fossem:

B	20	5	12	18	5
A	2	0	2	3	-1

O resultado da divisão seria:

C	10	0	6	6	-5

No caso, a divisão de B(2) com A(2) não ocorreria e C(2) permaneceria com o seu valor inicial.

9.7.2. FORALL

O comando **FORALL** funciona como um *loop* **DO..ENDDO**, porém pode ser usado com mais de uma variável de controle, sendo útil com operações com matrizes. Este comando tem a seguinte estrutura lógica:

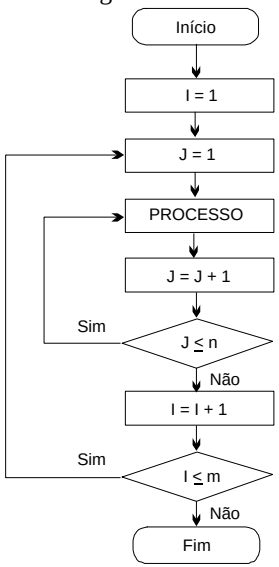


Figura 9.4. Fluxograma lógico do comando **FORALL**. No fluxograma, *n* e *m* se referem ao número de campos da matriz (linha e coluna).

Em termos de programação, a estrutura é:

```
FORALL (I = x:y, J = w:z)
      :
      PROCESSO
      :
END FORALL
```

EXERCÍCIOS

EXERCÍCIO 1

Muitos programas para engenharia envolvem a normalização de parte dos dados de entrada, como por exemplo, programas para redes neurais. Desenvolva um programa para normalizar um conjunto de dados. A normalização é feita usando a fórmula:

$$X_{NORM} = \frac{X - X_{MIN}}{X_{MAX} - X_{MIN}}$$

X	dado original
X_{MAX}	maior valor do conjunto de dados
X_{MIN}	menor valor do conjunto de dados
X_{NORM}	dado normalizado

O programa deve perguntar ao usuário o número de valores que serão lidos.

10. ARQUIVOS DE DADOS

As operações com arquivos no Fortran, em geral, são simples necessitando da abertura do arquivo, gravação ou leitura dos dados e o fechamento do arquivo.

Quando trabalhando com arquivos, deve-se ter em mente que o tempo de leitura e gravação em arquivos é uma operação relativamente lenta se comparada com as operações matemáticas. Portanto se um arquivo deve ser lido várias vezes durante a execução do programa, uma boa idéia é ler todo o arquivo de uma só vez, armazenando os dados em variáveis.

10.1. Operações com Arquivos

Arquivos são abertos usando o comando **OPEN** que tem forma:

```
OPEN (<unit>, FILE = <arquivo>)
```

<unit> unidade de referência para o arquivo
pode ser qualquer número inteiro

<arquivo> nome do arquivo a ser criado ou aberto.
o nome do arquivo deve vir entre aspas.

Para escrever dados no arquivo deve-se usar o comando **WRITE** usando a unidade do arquivo:

```
WRITE (<unit> , <formato>) <variáveis>
```

Para ler o arquivo de dados deve-se usar o comando **READ**, também usando a unidade do arquivo:

```
READ (<unit> , <formato>) <variáveis>
```

Antes do programa acabar deve-se fechar o arquivo de dados usando o comando **CLOSE**:

```
CLOSE (<unit>)
```


Estes tipos de arquivo usados pelo Fortran são arquivos texto simples e podem ser editados em qualquer editor de texto (desde que gravados no formato texto). Em geral se opta pela extensão **.TXT** ou **.DAT** para os arquivos de dados.

EXEMPLO 1
Para abrir o arquivo DATA01.DAT que contém dois números reais, calcular o produto destes dois número e gravar o resultado no arquivo RES01.DAT, podemos usar o seguinte programa em Fortran:

```
PROGRAM PRODUTO
IMPLICIT REAL*8 (A-H,O-Z)
! ABERTURA DE ARQUIVOS
OPEN (2,FILE = 'DATA01.DAT')
OPEN (3,FILE = 'RES01.DAT')

! LEITURA DAS VARIÁVEIS
READ(2,*) A, B

! CÁLCULO
C = A*B

! IMPRESSÃO DO RESULTADO
WRITE(3,*) C

CLOSE(2)
CLOSE(3)
END
```

10.2. Arquivos de Dados – Leitura

Deve-se tomar o devido cuidado para ler corretamente os dados do arquivo. É muito comum erros de arquivos com menos dados do que variáveis a serem lidas, ou de leitura errada dos dados (ler linha errada, ou deixar de ler alguma variável).

O comando **READ** lê uma linha de arquivo por vez. Portanto se um arquivo com três linha de dados tiver que ser lido, serão necessários 3 comandos **READ** para ler todo o arquivo. Se quatro **READ** forem usados, um erro indicando fim de arquivo será gerado e a execução do programa será terminada.

Em cada linha de dados, cada valor deverá ser separado por um espaço ou tabulações.

EXEMPLO 2
Um arquivo de dados pode ser usado para armazenar os dados de um reator químico, das condições iniciais de sua operação e dados cinéticos da reação. Uma linha do arquivo pode conter as dimensões do reator: altura e diâmetro; uma segunda linha pode conter os parâmetros cinéticos da reação: k (constante cinética) e Ea (energia de ativação) e uma terceira linha pode conter as condições operacionais iniciais do reator e reagentes: temperatura, concentração de reagente A e concentração de reagente B, como mostrado abaixo:

```
2.58        0.54
510.0       30100.5
342.5       0.015       9.3D-2
```

Um programa para ler estes dados do arquivo REAT.DAT seria:

```
PROGRAM LEARQ
IMPLICIT REAL*8 (A-H,O-Z)
OPEN(2, FILES = 'REAT.DAT')
READ(2,*) H,D
READ(2,*) AK, EA
READ(2,*) TEMP, CA, CB
END
```

10.2.1. EOF

O comando **EOF** (end of file) pode ser usado para auxiliar a leitura de arquivos grandes. Este comando indica se a última linha do arquivo já foi lida ou não. Se **EOF** for igual a verdadeiro, o final do arquivo foi atingido. Se for igual a falso, o final do arquivo ainda não foi atingido.

O uso deste comando tem a forma:

```
EOF(<unit>)
```

onde <unit> é a unidade do arquivo sendo lido.

EXEMPLO 3
Se o arquivo DADOS.TXT que contém duas colunas de números reais, porém com um número desconhecido de linhas tiver de ser lido, o comando EOF pode ser usado para controlar quando o programa deve parar de ler o arquivo.

```
PROGRAM READDATA
IMPLICIT REAL*8 (A-H,O-Z)
```



```
DIMENSION A(1000)

! LEITURA DOS DADOS
OPEN(2, FILE = 'DADOS.TXT')
N = 0
DO WHILE(.NOT.EOF(2))
      N = N + 1
      READ(*,*) A(I)
ENDDO
END
```

10.3. Arquivos de Dados – Impressão

Podemos escrever dados em um arquivo usando o comando **WRITE** podendo escolher entre escrever os valores com ou sem formato específico.

Caso os dados sejam gravados sem especificar um formato, serão gravados de dois a três valores por linha. Se mais de 3 variáveis forem escritas por **WRITE**, esta impressão ocupará mais de uma linha, o que pode comprometer posteriormente o entendimento da sequência dos dados que forem gravados.

A melhor opção para gravação de dados em arquivos é usar o comando **WRITE** com formato, de forma a ter uma melhor organização dos dados no arquivo. Neste caso não há o limite de até três valores por linha.

EXERCÍCIOS

EXERCÍCIO 1
Desenvolva um programa que leia o arquivo DATA01.TXT abaixo:

```
8.12D0
0.15D0
4.88D3
1030.4D0
```

Os dados devem ser lidos na variável **X**. O programa deve calcular X^2 e X^3 e imprimir na tela e em um arquivo os valores de X, X^2 e X^3, para cada um dos quatro valores contidos no arquivo de leitura.

EXERCÍCIO 2

Desenvolva um programa para calcular o progresso da reação química de decomposição do tolueno:

Concentração de tolueno: $C_A = C_{A0} \cdot \exp(-k \cdot T)$

$$k = A \cdot \exp\left(-\frac{Ea}{R \cdot T}\right)$$

C_A	concentração de tolueno
C_{A0}	concentração inicial de tolueno
A	fator pré-exponencial
Ea	energia de ativação
k	taxa de reação
R	constante dos gases
T	temperatura

Conversão de tolueno: $X_A = \frac{C_{A0} - C_A}{C_{A0}}$

X_A	conversão

O arquivo de entrada contém as condições operacionais iniciais, os parâmetros cinéticos da reação (A e Ea) e a constante dos gases, na seguinte sequência:

C_{A0} T
A Ea
R

Com os seguintes dados:

8,0 313,0
2.10^{49} 77500,0
1,987

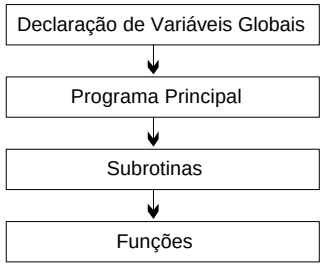
O programa deve calcular a concentração de tolueno e a conversão de tolueno para a reação, para tempos entre 0 e 200 minutos (20 pontos igualmente

espaçados) e imprimir o tempo e a conversão no arquivo RES1.DAT, e a concentração de reagente e produtos no arquivo RES2.DAT.

11. ORGANIZAÇÃO DE PROGRAMAS EXTENSOS

Conforme a complexidade de um programa aumenta, o programa necessita também de uma organização mais complexa, visando uma melhor organização do código e o compartilhamento de códigos comuns a várias etapas do algoritmo.

Desta forma podemos dividir o programa em um módulo de declaração de variáveis globais, programa principal, subrotinas e funções:



11.1. Módulo de Variáveis Globais

O módulo de variáveis globais deve conter as variáveis que serão utilizadas nas demais partes do programa. Declarar as variáveis num módulo de variáveis ajuda a não ter que passar as variáveis entre o programa principal e as subrotinas e funções.

A programação do módulo tem estrutura:

```
MODULE <nome>
      :
      VARIÁVEIS
      :
END MODULE
```

EXEMPLO 1

Um módulo de variáveis pode ser criado para resolver problemas de cálculo de reatores. Este tipo de problema geralmente necessita ser integrado e os dados relativos ao processo deve ser compartilhados entre o programa principal e a subrotina de integração numérica.


```
MODULE GLOBAL
      REAL*8 DENS, VISC, COND
      REAL*8 TEMP, PRES
      REAL*8 CONCA, CONCB
      INTEGER NPARAM
END MODULE
```

11.2. Programa Principal

Um programa Fortran deve ter um programa principal em sua estrutura, sendo ele tem a forma:

```
PROGRAM <nome>
      :
      PROCESSO
      :
END
```

onde <nome> é o nome que identifica o programa.

Deve-se ter o cuidado de não especificar nenhuma variável no programa contendo o mesmo nome do programa principal.

O programa principal controla todo o algoritmo que será seguido pelo programa, como declaração e inicialização de variáveis, leitura de dados, chamada de subrotinas e impressão dos resultados.

11.2.1. Use

As variáveis globais definidas no módulo de variáveis poderão ser acessíveis ao programa principal e às subrotinas e funções através do comando **USE**.

```
PROGRAM <nome>
IMPLICIT REAL*8 (A-H,O-Z)
USE <modulo>
:
END
```

O comando **USE** deve vir sempre depois da declaração de variáveis do programa principal ou subrotina.

11.3. Subrotinas

As subrotinas são subprogramas que executam procedimentos específicos. Uma subrotina pode ser chamada em vários pontos do programa de forma que ajuda a evitar a duplicação do mesmo código em pontos diferentes do programa.

```
SUBROUTINE <nome> (<variáveis>)
      :
      PROCESSO
      :
END SUBROUTINE
```

onde <nome> é o nome que identifica a subrotina. Deve-se ter o cuidado de não especificar nenhuma variável no programa contendo o mesmo nome da subrotina.

<variáveis> é a lista de variáveis que são passadas do programa principal ou outra subrotina para esta subrotina.

11.3.1. Call

As subrotinas são chamadas através do comando **CALL**, que tem a forma:

```
CALL <nome da subrotina> (<variáveis>)
```

onde <nome da subrotina> é o nome que identifica a subrotina.

<variáveis> é a lista de variáveis que são passadas para a subrotina que está sendo chamada.

EXEMPLO 1
Um exemplo simples para ilustrar aplicação de subrotinas é a criação de uma subrotina para calcular o produto entre dois números reais.

```
PROGRAM PROD1
IMPLICIT REAL*8 (A-H,O-Z)
READ(*,*) A,B
! CHAMADA DA SUBROTINA:
CALL PRODUTO (A,B,C)
```



```
WRITE(*,*) C
END

SUBROUTINE PRODUTO (A,B,C)
      IMPLICIT REAL*8 (A-H,O-Z)
      C = A*B
END SUBROUTINE
```

EXEMPLO 2
Se o mesmo exemplo fosse utilizado para multiplicar os campos de dois vetores, teríamos:

```
PROGRAM PROD2
IMPLICIT REAL*8 (A-H,O-Z)
DIMENSION A(10), B(10), C(10)
DO I = 1,10
      READ(*,*) A(I), B(I)
ENDDO
CALL PRODUTO (A,B,C)
END

SUBROUTINE PRODUTO (A,B,C)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(10), B(10), C(10)
      DO I = 1,10
            C(I) = A(I)*B(I)
      ENDDO
END SUBROUTINE
```

Note que os vetores ou matrizes são passados usando somente seu nome. A subrotina, porém, deve também dimensionar os vetores e matrizes que estão sendo passados.
Para poder generalizar a subrotina para aceitar qualquer tamanho de vetor, podemos passar na chamada da subrotina o tamanho do vetor:

```
PROGRAM PROD3
IMPLICIT REAL*8 (A-H,O-Z)
DIMENSION A(10), B(10), C(10)
N = 10
DO I = 1,N
      READ(*,*) A(I), B(I)
ENDDO
CALL PRODUTO (N,A,B,C)
END
```

```
SUBROUTINE PRODUTO (N,V1,V2,V3)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION V1(N), V2(N), V3(N)
      DO I = 1,N
            V3(I) = V1(I)*V2(I)
      ENDDO
END SUBROUTINE
```

Neste caso, o tamanho do vetor (N) também é passado para a subrotina e o comando **DIMENSION** se utiliza deste valor para dimensionar o tamanho do vetor.
Note também que as variáveis declaradas na subrotina (V1,V2,V3) podem ter nomes diferentes do que as variáveis que são passadas pelo programa principal (A,B,C). Porém quando a subrotina é chamada V1 recebe o valor de A, V2 recebe o valor de B e V3 recebe o valor de C; e quando a subrotina acaba, A recebe o valor de V1, B recebe o valor de V2 e C recebe o valor de V3.

O mesmo exemplo pode ser feito passando os valores das variáveis do programa principal para a subrotina definindo as variáveis a serem usadas em um módulo de variáveis globais:

```
MODULE GLOBAL
      INTEGER N
      REAL*8 A(10), B(10), C(10)
END MODULE

PROGRAM PROD4
      USE GLOBAL
      IMPLICIT REAL*8 (A-H,O-Z)
      DO I = 1,N
            READ(*,*) A(I), B(I)
      ENDDO
      CALL PRODUTO
END

SUBROUTINE PRODUTO ( )
      USE GLOBAL
      IMPLICIT REAL*8 (A-H,O-Z)
      DO I = 1,N
            C(I) = A(I)*B(I)
      ENDDO
END SUBROUTINE
```


Neste caso, as variáveis do programa principal e da subrotina devem ter o mesmo nome (A,B,C) pois estas duas partes do programa utilizam-se das variáveis definidas no modulo GLOBAL.
Esta forma de passar variáveis é muito útil quando subrotinas de métodos números são usadas (veja no Capítulo 12).

11.4. Funções

As funções são subprogramas que executam procedimentos específicos e retornam um valor único. Uma função pode ser chamada em vários pontos do programa de forma que ajuda a evitar a duplicação do mesmo código em pontos diferentes do programa.

```
<tipo> FUNCTION <nome> (<variáveis>)
      :
      PROCESSO
      :
END FUNCTION
```

onde <nome> é o nome que identifica a subrotina.

<variáveis> é a lista de variáveis que são passadas do programa principal ou outra subrotina para esta subrotina.

<tipo> é o tipo de valor que será retornado pela função: REAL, REAL*8, INTEGER, COMPLEX ou CHARACTER.

11.4.1. Chamando Funções

As funções são chamadas da seguinte forma:

```
<var> = <nome da função> (<variáveis>)
```

onde <nome da função> é o nome que identifica a função

<variáveis> é a lista de variáveis que são passadas para a função que está sendo chamada.

<var> é a variável que irá receber o valor retornado pela função

Esta forma de chamada da função é semelhante ao uso de qualquer função matemática intrínseca do Fortran, como mostrado no Capítulo 5.

EXEMPLO 3
Novamente, podemos usar o primeiro exemplo apresentado para multiplicar dois números reais e desenvolver um programa que se utilize de uma função para fazer este cálculo.

```
PROGRAM PROD5
IMPLICIT REAL*8 (A-H,O-Z)
READ(*,*) A,B
! CHAMADA DA FUNÇÃO:
C = PRODUTO (A,B)
WRITE(*,*) C
END

REAL*8 FUNCTION PRODUTO (A,B)
      IMPLICIT REAL*8 (A-H,O-Z)
      PRODUTO = A*B
END FUNCTION
```

Note que o nome da função (**PRODUTO**) deve ser igual ao nome da variável (**PRODUTO**) que terá o valor retornado para o programa principal.

12. MÉTODOS MATEMÁTICOS

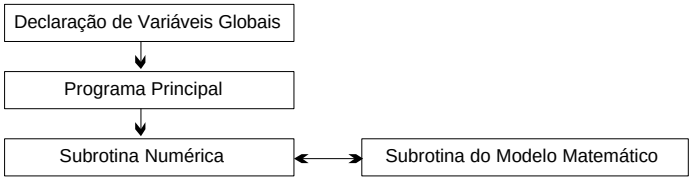
A resolução de modelos matemáticos de engenharia recai na utilização de métodos numéricos, como integração numérica, regressão, obtenção de raízes de funções, estimativa de parâmetros, entre outros.

Durante muito anos, vários pesquisadores e empresas desenvolveram subrotinas para resolução de métodos numéricos. Portanto, atualmente, cabe ao profissional fazer uso destas subrotinas prontas, dedicando maior atenção ao sistema a ser resolvido do que à programação de métodos numéricos.

12.1. Organização Geral do Programa

Quando bibliotecas numéricas ou subrotinas numéricas são utilizadas, a estrutura do programa segue uma forma semelhante à estrutura do programa apresentada no Capítulo 11.

Sendo assim devemos dividir o programa em um módulo de declaração de variáveis globais, programa principal, subrotinas numérica e subrotina que conterá o modelo matemático a ser resolvido:



Quando usamos uma subrotina numérica, esta subrotina é chamada pelo programa principal, que por sua vez chama a subrotina do modelo matemático.

Módulo de Variáveis Globais

O módulo de variáveis globais é muito útil quando se utiliza bibliotecas numéricas, pois é a forma mais fácil e eficiente de passar os valores das variáveis entre o programa principal e a subrotina que contém o modelo matemático.

A programação do módulo tem estrutura:


```
MODULE GLOBAL
      :
      VARIÁVEIS
      :
END MODULE
```

Programa Principal e Chamada da Subrotina Numérica

O programa principal deve conter a leitura e/ou inicialização das variáveis as serem usadas, e a chamada para a subrotina do método numérico:

Muitas subrotinas numéricas tem como um dos parâmetros de chamada, o nome da subrotina que contém o modelo matemático. Neste caso o nome da subrotina do modelo deve ser declarada no comando **EXTERNAL**:

```
PROGRAM <nome>
      USE <biblioteca>
      USE GLOBAL
      EXTERNAL <subrotina do modelo>
      :
      INICIALIZAÇÕES
      :
      CALL <subrotina numérica>
      :
END
```

onde <nome> é o nome que identifica o programa.
<biblioteca> é o nome da biblioteca numérica usada. Este comando é usado somente se o código da subrotina com o método numérico for intrínseco à biblioteca numérica. O comando não deve ser usado se o código da subrotina numérica for inserido ao programa.
<subrotina do modelo> é o nome da subrotina que contém o modelo matemático.
<subrotina numérica> é o nome da subrotina do método numérico e os parâmetros a serem passados para esta subrotina

Subrotina do Modelo Matemático

A subrotina do modelo matemático deve conter as equações que descrevem o modelo e cálculos auxiliares necessário para o cálculo das equações do modelo.

```
SUBROUTINE <nome> (<variáveis>)
      :
      EQUAÇÕES DO MODELO MATEMÁTICO
      :
END SUBROUTINE
```

onde <nome> é o nome que identifica a subrotina. Deve-se ter o cuidado de não especificar nenhuma variável no programa contendo o mesmo nome da subrotina.
<variáveis> é a lista de variáveis que são passadas do programa principal ou outra subrotina para esta subrotina.

12.1.1. Bibliotecas Numéricas

As bibliotecas numéricas são um conjunto de subrotinas contendo vários tipos de métodos numéricos. Estas bibliotecas podem vir na forma de módulos ou na forma de códigos individuais.

Quando a biblioteca está na forma de módulo, não é possível visualizar o código da subrotina, e para usar uma subrotina em específico deve-se declarar o uso do módulo (usando o comando **USE**) e depois chamar a subrotina usando o comando **CALL**.

Quando a biblioteca está na forma de código, deve-se copiar o código da subrotina para o programa ou deve-se adicionar o arquivo com a subrotina para o projeto sendo desenvolvido. Neste caso não se utiliza o comando **USE** para declarar o uso da biblioteca. Somente é necessário chamar a subrotina.

Bibliotecas na forma de módulo:
- IMSL (acompanha vários compiladores Fortran)
- NAG

Bibliotecas na forma de código:
- Numerical Recipes (pode ser lido em www.nr.com)
- Outras

12.1.2. Usando Bibliotecas Numéricas – IMSL

A biblioteca numérica IMSL é uma das bibliotecas mais usadas pois acompanha os compiladores Fortran: Compaq Fortran e Intel Fortran; e vem como opcionais em vários outros compiladores.

A estrutura geral de um programa que use alguma subrotina numérica do IMSL é:

```
MODULE GLOBAL
!       DECLARAÇÃO DAS VARIÁVEIS GLOBAIS
        INTEGER <variáveis>
        REAL*8 <variáveis>
END MODULE

! PROGRAMA PRINCIPAL
PROGRAM <nome>
        USE IMSL                        ! USA SUBROTINAS NUMÉRICAS DO IMSL
        USE GLOBAL                      ! USA VARIÁVEIS GLOBAIS
        IMPLICIT REAL*8(A-H,O-Z)
        EXTERNAL <subrotina do modelo>          ! SUBROTINA DO MODELO

!       INICIALIZAÇÃO DAS VARIÁVEIS DO MODELO
        <variável> = <valor>

!       INICIALIZAÇÃO DOS PARÂMETROS DA SUBROTINA
        <parâmetros> = <valor>

!       CHAMA A SUBROTINA DO MÉTODO NUMÉRICO
        CALL <subrotina do método numérico>

!       IMPRIME OS RESULTADOS PARCIAIS
        WRITE <variáveis>
END

! SUBROTINA QUE CONTÉM O MODELO MATEMÁTICO
SUBROUTINE <subrotina do modelo>
        USE GLOBAL                      ! USA VARIÁVEIS GLOBAIS
        IMPLICIT REAL*8(A-H,O-Z)

!       EQUAÇÕES DO MODELO
        <equações>

END SUBROUTINE
```

12.1.3. Usando Bibliotecas Numéricas – Outras

Quando bibliotecas numéricas que vem na forma de código são usadas, o código desta subrotina deve ser copiado para o final do programa. A estrutura geral de um programa que use este tipo de subrotina numérica é:

```
MODULE GLOBAL
!       DECLARAÇÃO DAS VARIÁVEIS GLOBAIS
        INTEGER <variáveis>
        REAL*8 <variáveis>
END MODULE

! PROGRAMA PRINCIPAL
PROGRAM <nome>
        USE GLOBAL                              ! USA VARIÁVEIS GLOBAIS
        IMPLICIT REAL*8(A-H,O-Z)
        EXTERNAL <subrotina do modelo>          ! SUBROTINA DO MODELO

!       INICIALIZAÇÃO DAS VARIÁVEIS DO MODELO
        <variável> = <valor>

!       INICIALIZAÇÃO DOS PARÂMETROS DA SUBROTINA
        <parâmetros> = <valor>

!       CHAMA A SUBROTINA DO MÉTODO NUMÉRICO
        CALL <subrotina do método numérico>

!       IMPRIME OS RESULTADOS PARCIAIS
        WRITE <variáveis>
END

! SUBROTINA QUE CONTÉM O MODELO MATEMÁTICO
SUBROUTINE <subrotina do modelo>
        USE GLOBAL                      ! USA VARIÁVEIS GLOBAIS
        IMPLICIT REAL*8(A-H,O-Z)

!       EQUAÇÕES DO MODELO
        <equações>

END SUBROUTINE

! SUBROTINA QUE CONTÉM O MÉTODO NUMÉRICO
! A SUBROTINA DEVE SER COPIADA NESTE PONTO DO PROGRAMA
! NÃO DEVE-SE FAZER NENHUMA ALTERAÇÃO NESTA SUBROTINA
SUBROUTINE <subrotina do método numérico>
        <código da subrotina>
END SUBROUTINE
```


12.2. Função de Zero

Grande parte das equações que descrevem fenômenos químicos, físicos e biológicos são equações não lineares, e portanto a resolução deste tipo de equações é parte integrante dos problemas de engenharia.

Diferentemente das equações lineares em que é possível achar uma solução algébrica, nem sempre é possível obter uma solução algébrica para equações não-lineares. Em geral é de interesse resolver f(x) = 0 e portanto deve-se achar as raízes da equação. Os principais métodos para obtenção das raizes da equação são: método da bisseção, método da secante, método de Newton ou métodos que combinem as características de dois deste métodos.

Uma das principais subrotinas numéricas para o cálculo das raízes de uma equação (**FZERO**) utiliza de um misto do método da bisseção com o método da secante, aliando a certeza de resposta da primeira com a rapidez da segunda.

12.2.1. Usando IMSL

A subrotina mais comum para obtenção de raízes do IMSL é a DZREAL.

A chamada desta subrotina tem a seguinte estrutura:

```
DZREAL (<modelo>,ATOL,RTOL,EPS,ETA,NRAIZ,ITMAX,XGUESS,X,INFO)
```

onde	<modelo>	nome da função que contém a equação.
	ATOL	erro absoluto (primeiro critério de parada)
	RTOL	erro relativo (segundo critério de parada)
	EPS	distância mínima entre os zeros da função
	ETA	critério de distanciamento. Se a distância entre dois zeros da função for menor do que a distância mínima definida em EPS, então um novo "chute" é dado a uma distância: última raiz encontrada + ETA.
	NRAIZ	número de raízes que devem ser obtidas
	ITMAX	número máximo de iterações
	XGUESS	vetor que deve conter os "chutes" iniciais dos valores das raízes (tamanho do vetor = NRAIZ)
	X	vetor que conterá as raízes da função (tamanho do vetor = NRAIZ)
	INFO	vetor que conterá o número de iterações necessárias para obter as raízes (tamanho do vetor = NRAIZ)

Função da Equação Matemática

A função que contém o equação matemática que se deseja obter as raízes tem a seguinte estrutura:

```
REAL*8 FUNCTION <modelo> (X)
```

onde X valor do ponto em que a função esta sendo avaliada.

EXEMPLO 1
Considerando que se deseja obter as raízes da equação:

$$f(X) = X^2 + 2 \cdot X - 6$$

A equação que será programada será a seguinte:

```
<modelo> = X**2 + 2*X -6
```

Estrutura Geral do Programa

A estrutura geral de um programa de integração usando a **DZREAL** tem a forma:

```
MODULE GLOBAL
!   DECLARAÇÃO DAS VARIÁVEIS GLOBAIS
    INTEGER <variáveis>
    REAL*8 <variáveis>
END MODULE

! PROGRAMA PRINCIPAL
PROGRAM <nome>
    USE IMSL                    ! USA SUBROTINAS NUMÉRICAS DO IMSL
    USE GLOBAL                  ! USA VARIÁVEIS GLOBAIS
    IMPLICIT REAL*8(A-H,O-Z)
    DIMENSION XGUESS(<nraiz>), X(<nraiz>), INFO(<nraiz>)
    EXTERNAL <modelo>           ! FUNÇÃO DO MODELO

!   INICIALIZAÇÃO DAS VARIÁVEIS DO MODELO
    NRAIZ = <nraiz>             ! DEFINE O NÚMERO DE RAIZES PROCURADAS
    <variável> = <valor>

!   DEFINIÇÃO DOS CHUTES INICIAS PARA AS RAÍZES
!   DEVEM SER DEFINIDOS nraiz CHUTES
    XGUESS(<campo>) = <valor>

!   INICIALIZAÇÃO DOS PARÂMETROS DA SUBROTINA
```



```
    EPS = 1.0D-5
    ETA = 1.0D-2
    ATOL = 1.0D-5
    RTOL = 1.0D-5
    ITMAX = 1000

!   CHAMA A SUBROTINA DE OBTENÇÃO DAS RAIZES
    CALL DZREAL(<modelo>,ATOL,RTOL,EPS,ETA,NRAIZ,ITMAX,XGUESS,X,INFO)

!   IMPRIME AS RAIZES
    WRITE <X>
END

! FUNÇÃO QUE CONTÉM A EQUAÇÃO
REAL*8 FUNCTION <modelo> (X)
    USE GLOBAL
    IMPLICIT REAL*8(A-H,O-Z)

!   EQUAÇÃO
    <modelo> = <equações>

END FUNCTION
```

EXEMPLO 2
Se desejarmos obter as duas raízes da equação apresentada no Exemplo 1, devemos utilizar o seguinte programa:

```
! PROGRAMA PRINCIPAL
PROGRAM RAIZES01
    USE IMSL                    ! USA SUBROTINAS NUMÉRICAS DO IMSL
    IMPLICIT REAL*8(A-H,O-Z)
    DIMENSION XGUESS(2), X(2), INFO(2)
    EXTERNAL FCN                ! FUNÇÃO DO MODELO

!   INICIALIZAÇÃO DAS VARIÁVEIS DO MODELO
    NRAIZ = 2                   ! DEFINE O NÚMERO DE RAIZES PROCURADAS

!   DEFINIÇÃO DOS CHUTES INICIAS PARA AS RAÍZES
!   DEVEM SER DEFINIDOS nraiz CHUTES
    XGUESS(1) = 4.5D0
    XGUESS(2) = -100.0D0

!   INICIALIZAÇÃO DOS PARÂMETROS DA SUBROTINA
    EPS = 1.0D-5
    ETA = 1.0D-2
    ATOL = 1.0D-5
    RTOL = 1.0D-5
    ITMAX = 1000
```

```
!   CHAMA A SUBROTINA DE OBTENÇÃO DAS RAIZES
    CALL DZREAL(FCN,ATOL,RTOL,EPS,ETA,NRAIZ,ITMAX,XGUESS,X,INFO)

!   IMPRIME AS RAIZES
    WRITE(*,*) X(1), X(2)
END

! FUNÇÃO QUE CONTÉM A EQUAÇÃO
REAL*8 FUNCTION FCN(X)
    IMPLICIT REAL*8(A-H,O-Z)
!   EQUAÇÃO
    FCN = X**2.0D0 + 2.0D0*X - 6.0D0
END FUNCTION
```

Note que neste programa, não foi passado nenhuma variável do programa principal para a função **FCN**. Portanto o módulo de variáveis globais não foi necessário.
Apenas a variável **X** é passada para **FCN**, mas esta variável é passada do programa principal para a subrotina **DZREAL** e da subrotina para a função **FCN**.

12.2.2. Usando Numerical Recipes

No Numerical Recipes encontram-se listadas várias subrotinas para obtenção de zeros de função. Abaixo mostramos o uso da subrotina **RTBIS** (adaptada do Numerical Recipes), que usa o método da bisseção para encontrar a raiz de uma função.

A chamada desta função tem a seguinte estrutura:

```
RTBIS (<modelo>,X1,X2,TOL)
```

onde	<modelo>	nome da função que contém a equação.
	X1	valor inicial da faixa de valores onde a raiz será procurada
	X2	valor final da faixa de valores onde a raiz será procurada
	TOL	erro absoluto (critério de parada)
	INFO	vetor que conterá o número de iterações necessárias para obter as raízes (tamanho do vetor = NRAIZ)

Nesta subrotina, a raiz da função é procurada entre os valores de X1 e X2.

Função da Equação Matemática

A função que contém o equação matemática que se deseja obter as raízes tem a seguinte estrutura:

```
REAL*8 FUNCTION <modelo> (X)
```

onde X valor do ponto em que a função esta sendo avaliada.

Estrutura Geral do Programa

A estrutura geral de um programa de integração usando a **RTBIS** tem a forma:

```
MODULE GLOBAL
!   DECLARAÇÃO DAS VARIÁVEIS GLOBAIS
    INTEGER <variáveis>
    REAL*8 <variáveis>
END MODULE

! PROGRAMA PRINCIPAL
PROGRAM <nome>
    USE GLOBAL                      ! USA VARIÁVEIS GLOBAIS
    IMPLICIT REAL*8(A-H,O-Z)
    EXTERNAL <modelo>               ! FUNÇÃO DO MODELO

!   INICIALIZAÇÃO DAS VARIÁVEIS DO MODELO
    <variável> = <valor>

!   DEFINIÇÃO DOS CHUTES INICIAS PARA AS RAÍZES
!   DEVEM SER DEFINIDOS OS LIMITES INFERIOR E SUPERIOR DE BUSCA
    X1 = <valor inferior>
    X2 = <valor superior>

!   INICIALIZAÇÃO DOS PARÂMETROS DA FUNÇÃO
    TOL = 1.0D-5

!   CHAMA A FUNÇÃO DE OBTENÇÃO DA RAIZ
    XRAIZ = RTBIS(<modelo>,X1,X2,TOL)

!   IMPRIME A RAIZ
    WRITE <XRAIZ>
END

! FUNÇÃO QUE CONTÉM A EQUAÇÃO
REAL*8 FUNCTION <modelo> (X)
    USE GLOBAL
```

```
    IMPLICIT REAL*8(A-H,O-Z)

!   EQUAÇÃO
    <modelo> = <equação>

END FUNCTION

! FUNÇÃO COM O MÉTODO DA BISSEÇÃO PARA
! OBTENÇÃO DA RAIZ DE UMA FUNÇÃO
REAL*8 FUNCTION RTBIS(FUNC,X1,X2,XACC)
    IMPLICIT REAL*8(A-H,O-Z)

    JMAX = 1000
    FMID = FUNC(X2)
    F = FUNC(X1)
    IF (F*FMID >= 0.0D0) THEN
        WRITE(*,*) ' NAO EXISTE RAIZ ENTRE ', X1, ' E ', X2
        RETURN
    ENDIF
    IF (F < 0.0D0) THEN
        RTBIS = X1
        DX = X2 - X1
    ELSE
        RTBIS = X2
        DX = X1 - X2
    ENDIF
    DO J = 1,JMAX
        DX = DX*0.5D0
        XMID = RTBIS + DX
        FMID = FUNC(XMID)
        IF (FMID <= 0.0D0) RTBIS = XMID
        IF ((ABS(DX) < XACC).OR.(FMID == 0.0D0)) RETURN
    ENDDO
    WRITE(*,*) ' NUMERO MAXIMO DE ITERACOES FOI ULTRAPASSADO '
END FUNCTION
```

EXEMPLO 3

A função **RTBIS** apenas retorna uma única raiz no intervalo especificado. Se duas ou mais raízes tiverem de ser obtidas, o programa deve chamar a função **RTBIS**, especificando um intervalo de busca diferente.

Se desejarmos obter as duas raízes da equação apresentada no Exemplo 1, devemos utilizar o seguinte programa:

```
! PROGRAMA PRINCIPAL
! OBTENÇÃO DE RAIZES PELO MÉTODO DA BISSEÇÃO
PROGRAM RAIZES03
    IMPLICIT REAL*8(A-H,O-Z)
    EXTERNAL FCN                ! FUNÇÃO DO MODELO
```



```
!   DEFINIÇÃO DOS CHUTES INICIAS PARA AS RAÍZES
!   DEVEM SER DEFINIDOS OS LIMITES INFERIOR E SUPERIOR DE BUSCA
    X1 = 0.0D0
    X2 = 5.0D0

!   INICIALIZAÇÃO DOS PARÂMETROS DA SUBROTINA
    TOL = 1.0D-4

!   CHAMA A FUNÇÃO DE OBTENÇÃO DAS RAIZES
    XRAIZ1 = RTBIS(FCN,X1,X2,TOL)

!   OBTENÇÃO DA SEGUNDA RAIZ
    X1 = -10.0D0
    X2 = 0.0D0
    XRAIZ2 = RTBIS(FCN,X1,X2,TOL)

!   IMPRIME AS RAIZES
    WRITE(*,*) XRAIZ1, XRAIZ2
END

! FUNÇÃO QUE CONTÉM A EQUAÇÃO
REAL*8 FUNCTION FCN(X)
    IMPLICIT REAL*8(A-H,O-Z)
!   EQUAÇÃO
    FCN = X**2.0D0 + 2.0D0*X - 6.0D0
END FUNCTION

! INSERIR NESTE PONTO A FUNÇÃO DO MÉTODO NUMÉRICO (RTBIS)
```

12.3. Integração Numérica

Programa que envolvem integração numérica são muito comuns em engenharia, principalmente em aplicações de controle de processos, dinâmica de processos, cálculo de reatores, leitos fixos e fluidizados, processos de absorção e adsorção, filtração, secagem, entre outros.

As subrotinas mais utilizadas para integração numérica são as subrotinas baseadas no método de Runge-Kutta e DASSL.

12.3.1. Usando IMSL

A subrotina mais comum para integração numérica do IMSL é a DIVPRK, baseada no método de Runge-Kutta.

A chamada desta subrotina tem a seguinte estrutura:

DIVPRK(IDO, NEQ, <modelo>, T, TOUT, ATOL, PARAM, Y)

onde	<modelo>	nome da subrotina que contém o modelo matemático.
	IDO	variável de controle da integração e de erro
	NEQ	número de equações do modelo matemático
	T	tempo inicial de integração
	TOUT	tempo final de integração
	ATOL	tolerância
	PARAM	vetor com as opções de configuração da subrotina
	Y	variável sendo integrada

A variável **IDO** controla a entrada e saída da subrotina, modificando seu valor dependendo se ocorreu algum erro durante a execução da subrotina. A variável **IDO** deve ser inicializada com o valor **1** antes de entrar pela primeira vez na subrotina. Quando a execução da subrotina **DIVPRK** termina sua execução, a variável **IDO** pode conter os valores: **2** quando não houve erro de execução, ou **4, 5** ou **6** quando houve algum erro. Em caso de erro, a integração deve ser interrompida. Se não houve erro (**IDO = 2**) a subrotina de integração pode ser chamada novamente dando continuidade à integração. Após o termino do uso da subrotina **DIVPRK**, a variável **IDO** deve receber o valor **3** e a subrotina deve ser chamada pela última vez, para liberar memória e indicar o fim da integração.

A variável **PARAM** é um vetor com 50 campos, que contém opções de como a subrotina **DIVPRK** deve conduzir a integração. Se a variável **PARAM** for inicializada com o valor **0.0D0** em todos os seus campos, isto indicará que a subrotina deve ser conduzida em sua forma padrão (funcionamento bom para a grande maioria dos casos). Para integrações mais complicadas (*stiff*), é necessário modificar algumas opções da integração:

PARAM(1)	passo inicial da integração
PARAM(2)	passo mínimo de integração
PARAM(3)	passo máximo de integração
PARAM(4)	aumenta o número de iterações (normal: 500)

Subrotina do Modelo Matemático

A subrotina que contém o modelo matemático a ser integrado tem a seguinte estrutura:

SUBROUTINE <modelo> (N,T,Y,YPRIME)

onde	N	número de equações diferenciais
	T	tempo
	Y	variável
	YPRIME	derivada de Y

EXEMPLO 4
Considerando um sistema de equações diferencias contendo três equações:

$$\frac{dC}{dt} = 3 \cdot X + 2 \cdot X^2$$

$$\frac{dT}{dt} = 2 \cdot X \cdot C$$

$$\frac{dX}{dt} = 3 \cdot \exp(-5 / T)$$

Para construir um modelo matemático para ser usado com a subrotina **DIVPRK**, temos que renomear as variáveis C, T e X tornando-as campos da variável Y. Portanto Y(1) = C, Y(2) = T e Y(3) = X.
A mesma analogia deve ser seguida para as derivadas de C, T e X, que se tornaram campos da variável YPRIME. Portanto YPRIME(1) = dC/dt, YPRIME(2) = dT/dt e YPRIME(3) = dX/dt.
Atenção: A variável YPRIME(1) deve conter a derivada da variável Y(1) e assim por diante.
O sistema de equações que será programado será o seguinte:

$$YPRIME(1) = 3 \cdot Y(3) + 2 \cdot Y(3)^2$$

$$YPRIME(2) = 2 \cdot Y(3) \cdot Y(1)$$

$$YPRIME(3) = 3 \cdot \exp(-5 / Y(2))$$

Estrutura Geral do Programa

A estrutura geral de um programa de integração usando a **DIVPRK** tem a forma:

```
MODULE GLOBAL
!   DECLARAÇÃO DAS VARIÁVEIS GLOBAIS
    INTEGER <variáveis>
    REAL*8 <variáveis>
END MODULE
```

```
! PROGRAMA PRINCIPAL
PROGRAM <nome>
    USE IMSL                    ! USA SUBROTINAS NUMÉRICAS DO IMSL
    USE GLOBAL                  ! USA VARIÁVEIS GLOBAIS
    IMPLICIT REAL*8(A-H,O-Z)
    ! Y = VARIÁVEL, YPRIME = DERIVADA DE Y
    REAL*8 Y(<tamanho>), YPRIME(<tamanho>)
    DIMENSION PARAM(50)         ! OBRIGATÓRIO PARA DIVPRK
    EXTERNAL <modelo>           ! SUBROTINA DO MODELO

!   INICIALIZAÇÃO DAS VARIÁVEIS DO MODELO
    NEQ = <tamanho>                 ! NÚMERO DE EQUAÇÕES DIFERENCIAIS
    Y(<campo>) = <valor>            ! VALORES INICIAIS DAS VARIÁVEIS A
                                    ! SEREM INTEGRADAS
    <variável> = <valor>

!   DEFINIÇÃO DA FAIXA DE INTEGRAÇÃO
    T     = 0.0D0               ! TEMPO INICIAL
    TFINAL = <tempo>            ! TEMPO FINAL
    TIMPR  = <intervalo>        ! INVERVALO DE IMPRESSÃO
                                ! DEVE SER MENOR OU IGUAL A TFINAL

!   INICIALIZAÇÃO DOS PARÂMETROS DA SUBROTINA
    IDO = 1                 ! INICIALIZAÇÃO DA SUBROTINA
    ATOL = 1.0D-4           ! TOLERÂNCIA
    PARAM = 0.0D0           ! USA A SUBROTINA DIVPRK NA SUA CONFIG.PADRÃO
    PARAM(4) = 1000000      ! AUMENTA O NÚMERO DE ITERAÇÕES POSSÍVEIS

!   IMPRIME AS CONDIÇÒES INICIAIS
    WRITE(*,*) <variáveis>

    DO WHILE (T < TFINAL)
!       FAZ INTEGRAÇÃO ATÉ O PROXIMO PONTO DE IMPRESSÃO
        TOUT = T + TIMPR

!       CHAMA A SUBROTINA DE INTEGRAÇÃO
        CALL DIVPRK (IDO, NEQ, <modelo>, T, TOUT, ATOL, PARAM, Y)

!       IMPRIME OS RESULTADOS PARCIAIS
        WRITE(*,*) <variáveis>
    ENDDO

!   TERMINA A INTEGRAÇÃO E LIBERA ESPAÇO NA MEMÓRIA
!   (OBRIGATÓRIO PARA DIVPRK)
    CALL DIVPRK (3, NEQ, FCNMOD, T, TOUT, ATOL, PARAM, C)
END

! SUBROTINA QUE CONTÉM O MODELO MATEMÁTICO
SUBROUTINE <modelo> (NEQ, T, Y, YPRIME)
    USE GLOBAL                  ! USA VARIÁVEIS GLOBAIS
    IMPLICIT REAL*8(A-H,O-Z)
    DIMENSION Y(NEQ), YPRIME(NEQ)

!   EQUAÇÕES DIFERENCIAIS DO MODELO
    YPRIME(<campo>) = <equação>

END SUBROUTINE
```


EXEMPLO 5

Compostos clorados derivados do benzeno são produzidos, geralmente, em reatores do tipo semi-batelada, que é um reator em que parte dos reagentes é introduzida antes do início da reação e outra parte dos reagentes é continuamente alimentada ao longo do processo.

No caso da cloração do benzeno, uma carga inicial de benzeno é introduzida no reator e cloro é alimentado à um fluxo contínuo no reator de forma que a concentração de cloro no reator seja igual a sua concentração de saturação no benzeno e seus derivados.

Três reações ocorrem simultaneamente no reator, produzindo três diferentes derivados do benzeno: monoclorobenzeno, diclorobenzeno e triclorobenzeno.

$C_6H_6 + Cl_2 \rightarrow C_6H_5Cl + HCl$

$C_6H_5Cl + Cl_2 \rightarrow C_6H_4Cl_2 + HCl$

$C_6H_4Cl_2 + Cl_2 \rightarrow C_6H_3Cl_3 + HCl$

No reator a concentração de benzeno e seus derivados variam constantemente com relação ao tempo de reação, de acordo com as equações:

$$\frac{dC_B}{dt} = -k_1 \cdot C_B \cdot C_{Cl}$$

$$\frac{dC_{MCB}}{dt} = +k_1 \cdot C_B \cdot C_{Cl} - k_2 \cdot C_{MCB} \cdot C_{Cl}$$

$$\frac{dC_{DCB}}{dt} = +k_2 \cdot C_{MCB} \cdot C_{Cl} - k_3 \cdot C_{DCB} \cdot C_{Cl}$$

$$\frac{dC_{TCB}}{dt} = +k_3 \cdot C_{DCB} \cdot C_{Cl}$$

C_B	concentração de benzeno
C_{MCB}	concentração de monoclorobenzeno
C_{DCB}	concentração de diclorobenzeno
C_{TCB}	concentração de triclorobenzeno
C_{Cl}	concentração de cloro

k_1	constante de reação = 24,08 L/mol.min
k_2	constante de reação = 3,02 L/mol.min
k_3	constante de reação = 0,10 L/mol.min

A carga inicial de benzeno no reator é de 8850 kg (peso molecular 78 g/mol e densidade 0.8731 kg/L). A concentração de cloro permanece constante em 0.12 mol de cloro por mol de benzeno alimentado inicialmente (devido a alimentação contínua de cloro no reator). A concentração de HCl é praticamente zero, pois o HCl vaporiza e deixa o reator.

O perfil de concentrações do benzeno e derivados do benzeno em função do tempo de reação pode ser obtido pelo seguinte programa:

```
MODULE GLOBAL
    REAL*8 B0,ANB0,AK1,AK2,AK3
    REAL*8 CL,V
END MODULE


! PROGRAMA PARA CÁLCULO DE UM REATOR PARA PRODUÇÃO DE CLOROBENZENOS
PROGRAM CLBENZ
    USE IMSL                    ! USA SUBROTINAS NUMÉRICAS DO IMSL
    USE GLOBAL                  ! USA VARIÁVEIS GLOBAIS
    IMPLICIT REAL*8(A-H,O-Z)
    REAL*8 Y(4), YPRIME(4)      ! Y = VARIÁVEL, YPRIME = DERIVADA DE Y
    DIMENSION PARAM(50)         ! OBRIGATÓRIO PARA DIVPRK
    EXTERNAL FCNMOD             ! SUBROTINA DO MODELO

!   INICIALIZAÇÃO DOS PARÂMETROS E CONSTANTES DO MODELO
    B0 = 8850.0D0
    ANB0 = B0/0.078D0

    AK1 = 28.08D0
    AK2 = 3.02D0
    AK3 = 0.10D0

    V = 0.8731D0*B0

!   INICIALIZAÇÃO DAS VARIÁVEIS DO MODELO
    NEQ = 4
    Y(1) = ANB0/V               ! CONC.BENZENO
    Y(2) = 0.0D0                ! CONC.CLOROBENZENO
    Y(3) = 0.0D0                ! CONC.DICLOROBENZENO
    Y(4) = 0.0D0                ! CONC.TRICLOROBENZENO
    CL = 0.012D0*ANB0/V         ! CONC.CLORO

!   DEFINIÇÃO DA FAIXA DE INTEGRAÇÃO
    T     = 0.0D0               ! TEMPO INICIAL
```



```
    TFINAL = 10.0D0             ! TEMPO FINAL
    TIMPR  = 0.5D0              ! INVERVALO DE IMPRESSÃO

!   INICIALIZAÇÃO DOS PARÂMETROS DA SUBROTINA
    IDO = 1                     ! INICIALIZAÇÃO DA SUBROTINA
    ATOL = 1.0D-4               ! TOLERÂNCIA
    PARAM = 0.0D0               ! USA A SUBROTINA DIVPRK NA SUA CONFIG.PADRÃO
    PARAM(4) = 1000000          ! AUMENTA O NÚMERO DE ITERAÇÕES POSSÍVEIS

!   IMPRIME AS CONDIÇÕES INICIAIS
    WRITE(*,*) 'TEMPO  BENZENO   CLBENZ  DICLBENZ  TRICLBENZ'
    WRITE(*,'(F4.1,4(F10.2))') T,Y(1),Y(2),Y(3),Y(4)

    DO WHILE (T < TFINAL)
!       FAZ INTEGRAÇÃO ATÉ O PROXIMO PONTO DE IMPRESSÃO
        TOUT = T + TIMPR

!       CHAMA A SUBROTINA DE INTEGRAÇÃO
        CALL DIVPRK (IDO, NEQ, FCNMOD, T, TOUT, ATOL, PARAM, Y)

!       IMPEDE CONCENTRAÇÕES NEGATIVAS
        WHERE(Y < 0.0D0) Y = 0.0D0

!       IMPRIME OS RESULTADOS PARCIAIS
        WRITE(*,'(F4.1,4(F10.2))') T,Y(1),Y(2),Y(3),Y(4)

    ENDDO

!   TERMINA A INTEGRAÇÃO E LIBERA ESPAÇO NA MEMÓRIA
!   (OBRIGATÓRIO PARA DIVPRK)
    CALL DIVPRK (3, NEQ, FCNMOD, T, TOUT, ATOL, PARAM, C)

END


! SUBROTINA QUE CONTÉM O MODELO MATEMÁTICO A SER INTEGRADO
SUBROUTINE FCNMOD(NEQ,T,Y,YPRIME)
    USE GLOBAL
    IMPLICIT REAL*8(A-H,O-Z)
    DIMENSION Y(NEQ), YPRIME(NEQ)

    YPRIME(1) = - AK1*Y(1)*CL
    YPRIME(2) = AK1*Y(1)*CL - AK2*Y(2)*CL
    YPRIME(3) = AK2*Y(2)*CL - AK3*Y(3)*CL
    YPRIME(4) = AK3*Y(3)*CL

END SUBROUTINE
```

12.3.2. Usando Numerical Recipes

No Numerical Recipes encontram-se listadas várias subrotinas para integração numérica. Abaixo mostramos o uso da subrotina **RK4** (adaptada do Numerical Recipes), que usa o método de Runge-Kutta para integração numérica

A chamada desta função tem a seguinte estrutura:

```
RK4(NEQ,Y,YPRIME,T,H,YOUT,<modelo>)
```

onde	<modelo>	nome da subrotina que contém o modelo matemático.
	NEQ	número de equações do modelo
	Y	valor inicial da variável sendo integrada
	YPRIME	valor da derivada da variável sendo integrada
	T	tempo inicial de integração
	H	passo de integração. Recomenda-se que seja pelo igual ou menor do que 10^{-3}.TIMPR (onde TIMPR é o intervalo de impressão dos valores de Y)
	YOUT	valor final da variável sendo integrada (após ser integrada entre T e T+H)

Subrotina do Modelo Matemático

A subrotina que contém o modelo matemático a ser integrado tem a seguinte estrutura:

```
SUBROUTINE <modelo> (NEQ, T, Y, YPRIME)
```

onde	NEQ	número de equações diferenciais
	T	tempo
	Y	variável sendo integrada
	YPRIME	derivada de Y

A forma de programar o modelo matemático é igual ao mostrado no Exemplo 4.

Estrutura Geral do Programa

A estrutura geral de um programa de integração usando a **RK4** tem a forma:


```
MODULE GLOBAL
!   DECLARAÇÃO DAS VARIÁVEIS GLOBAIS
    INTEGER <variáveis>
    REAL*8 <variáveis>
END MODULE

! PROGRAMA PRINCIPAL
PROGRAM <nome>
    USE GLOBAL                          ! USA VARIÁVEIS GLOBAIS
    IMPLICIT REAL*8(A-H,O-Z)
    ! Y = VARIÁVEL, YPRIME = DERIVADA DE Y, YOUT = VARIÁVEL AUXILIAR
    REAL*8 Y(<tamanho>), YPRIME(<tamanho>), YOUT(<tamanho>)
    EXTERNAL <modelo>                   ! SUBROTINA DO MODELO

!   INICIALIZAÇÃO DAS VARIÁVEIS DO MODELO
    NEQ = <tamanho>                 ! NÚMERO DE EQUAÇÕES DIFERENCIAIS
    Y(<campo>) = <valor>            ! VALORES INICIAIS DAS VARIÁVEIS A
                                    ! SEREM INTEGRADAS
    <variável> = <valor>

!   DEFINIÇÃO DA FAIXA DE INTEGRAÇÃO
    T      = 0.0D0                  ! TEMPO INICIAL
    TFINAL = <tempo>                ! TEMPO FINAL
    TIMPR  = <intervalo>            ! INVERVALO DE IMPRESSÃO
                                    ! DEVE SER MENOR OU IGUAL A TFINAL

!   INICIALIZAÇÃO DOS PARÂMETROS DA SUBROTINA
    H =     1.0D-3                  ! PASSO DE INTEGRAÇÃO

    DO WHILE (T < TFINAL)
!       FAZ INTEGRAÇÃO ATÉ O PROXIMO PONTO DE IMPRESSÃO
        TOUT = T + TIMPR

!       CHAMA A SUBROTINA DE INTEGRAÇÃO
        DO WHILE (T < TOUT)
            CALL RK4(NEQ,Y,YPRIME,T,H,YOUT,<modelo>)
            Y = YOUT
            T = T + H
        ENDDO

!       IMPRIME OS RESULTADOS PARCIAIS
        WRITE(*,*) <variáveis>
    ENDDO

END

! SUBROTINA QUE CONTÉM O MODELO MATEMÁTICO
SUBROUTINE <modelo> (NEQ, T, Y, YPRIME)
    USE GLOBAL                      ! USA VARIÁVEIS GLOBAIS
    IMPLICIT REAL*8(A-H,O-Z)
```

```
    DIMENSION Y(NEQ), YPRIME(NEQ)

!   EQUAÇÕES DIFERENCIAIS DO MODELO
    YPRIME(<campo>) = <equação>
END SUBROUTINE


! SUBROTINA QUE CONTÉM O MÉTODO DE INTEGRAÇÃO NUMÉRICA
SUBROUTINE RK4(N,Y,DYDX,X,H,YOUT,DERIVS)
    IMPLICIT REAL*8(A-H,O-Z)
    DIMENSION Y(N), YOUT(N), DYDX(N)
    DIMENSION DYM(N), DYT(N), YT(N)
    HH = 0.5D0*H
    H6 = H/6.0D0
    XH = X + HH
    DO I=1,N
        YT(I) = Y(I) + HH*DYDX(I)
    ENDDO
    CALL DERIVS(N,XH,YT,DYT)
    DO I = 1,N
        YT(I) = Y(I) + HH*DYT(I)
    ENDDO
    CALL DERIVS(N,XH,YT,DYM)
    DO I = 1,N
        YT(I) = Y(I) + H*DYM(I)
        DYM(I) = DYT(I) + DYM(I)
    ENDDO
    CALL DERIVS(N,X+H,YT,DYT)
    DO I = 1,N
        YOUT(I) = Y(I) + H6*(DYDX(I) + DYT(I) + 2.0D0*DYM(I))
    ENDDO
END SUBROUTINE
```

EXEMPLO 6

Se desejarmos integrarmos o modelo matemático apresentado no Exemplo 5, usando a subrotina **RK4**, devemos utilizar o seguinte programa:

```
MODULE GLOBAL
!   DECLARAÇÃO DAS VARIÁVEIS GLOBAIS
    REAL*8 B0,ANB0,AK1,AK2,AK3
    REAL*8 CL,V
END MODULE

! PROGRAMA PRINCIPAL
PROGRAM CLBENZ02
    USE GLOBAL                          ! USA VARIÁVEIS GLOBAIS
    IMPLICIT REAL*8(A-H,O-Z)
    ! Y = VARIÁVEL, YPRIME = DERIVADA DE Y, YOUT = VARIÁVEL AUXILIAR
    REAL*8 Y(4), YPRIME(4), YOUT(4)
```



```
    EXTERNAL FCNMOD                 ! SUBROTINA DO MODELO

!   INICIALIZAÇÃO DOS PARÂMETROS E CONSTANTES DO MODELO
    B0 = 8849.5D0
    ANB0 = B0/0.078D0
    AK1 = 24.08D0
    AK2 = 3.02D0
    AK3 = 0.10D0
    V = 0.8731D0*B0

!   INICIALIZAÇÃO DAS VARIÁVEIS DO MODELO
    NEQ = 4
    Y(1) = ANB0/V           ! CONC.BENZENO
    Y(2) = 0.0D0            ! CONC.CLOROBENZENO
    Y(3) = 0.0D0            ! CONC.DICLOROBENZENO
    Y(4) = 0.0D0            ! CONC.TRICLOROBENZENO
    CL = 0.012D0*ANB0/V     ! CONC.CLORO

!   DEFINIÇÃO DA FAIXA DE INTEGRAÇÃO
    T     = 0.0D0           ! TEMPO INICIAL
    TFINAL = 10.0D0         ! TEMPO FINAL
    TIMPR  = 0.5D0          ! INVERVALO DE IMPRESSÃO

!   INICIALIZAÇÃO DOS PARÂMETROS DA SUBROTINA
    H =     1.0D-3          ! PASSO DE INTEGRAÇÃO

!   IMPRIME AS CONDIÇÕES INICIAIS
    WRITE(*,*) 'TEMPO  BENZENO   CLBENZ  DICLBENZ  TRICLBENZ'
    WRITE(*,'(F4.1,4(F10.2))') T,Y(1),Y(2),Y(3),Y(4)

    DO WHILE (T < TFINAL)
!       FAZ INTEGRAÇÃO ATÉ O PROXIMO PONTO DE IMPRESSÃO
        TOUT = T + TIMPR

!       CHAMA A SUBROTINA DE INTEGRAÇÃO
        DO WHILE (T < TOUT)
            CALL RK4(NEQ,Y,YPRIME,T,H,YOUT,FCNMOD)
            Y = YOUT
            T = T + H
!           IMPEDE CONCENTRAÇÕES NEGATIVAS
            WHERE(Y < 0.0D0) Y = 0.0D0
        ENDDO

!       IMPRIME OS RESULTADOS PARCIAIS
        WRITE(*,'(F4.1,4(F10.2))') T,Y(1),Y(2),Y(3),Y(4)
    ENDDO
END

! SUBROTINA QUE CONTÉM O MODELO MATEMÁTICO
SUBROUTINE FCNMOD (NEQ, T, Y, YPRIME)
    USE GLOBAL                  ! USA VARIÁVEIS GLOBAIS
    IMPLICIT REAL*8(A-H,O-Z)
```

```
    DIMENSION Y(NEQ), YPRIME(NEQ)

    YPRIME(1) = - AK1*Y(1)*CL
    YPRIME(2) = AK1*Y(1)*CL - AK2*Y(2)*CL
    YPRIME(3) = AK2*Y(2)*CL - AK3*Y(3)*CL
    YPRIME(4) = AK3*Y(3)*CL
END SUBROUTINE
```

12.4. Regressão Não-Linear

A regressão não-linear é muito usado em engenharia, pois muitas vezes é necessário achar os coeficientes (ou parâmetros) de uma equação ou modelo para um conjunto de observações.

As subrotinas mais utilizadas para integração numérica são as subrotinas baseadas no método de

12.4.1. Usando IMSL

A subrotina mais comum para integração numérica do IMSL é a DRNLIN, baseada no método de

A chamada desta subrotina tem a seguinte estrutura:

```
DRNLIN (<modelo>, NPRM, IDERIV, THETA, R, LDR, IRANK, DFE, SSE)
```

onde	<modelo>	nome da subrotina que contém o modelo matemático
	NPRM	número de parâmetros a serem estimados
	IDERIV	opção de derivada da função: **0** – derivadas são calculadas por diferenças finitas, **1** – derivada fornecida pelo usuário
	THETA	vetor com os *NPRM* parâmetros
	R	matriz triangular NPRM x NPRM que contém a matriz resultante da decomposição do jacobiano.
	LDR	dimensão de R
	IRANK	ordem da matriz de R
	DFE	grau de liberdade do erro
	SSE	soma dos quadrados do erro

Subrotina do Modelo Matemático

A subrotina que contém o modelo matemático a ser integrado tem a seguinte estrutura:


```
SUBROUTINE <modelo> (NPRM,THETA,IOPT,IOBS,FRQ,WT,E,DE,IEND)
```

onde	NPRM	número de parâmetros
	THETA	vetor com os parâmetros
	IOPT	opção de avaliação: **0** – calcula a função, **1** – calcula a derivada
	IOBS	número da observação
	FRQ	frequência para a observação
	WT	peso da observação
	E	erro da observação IOBS
	DE	vetor que contém as derivadas parciais do resíduo para a observação IOBS
	IEND	indicador de finalização: **0** – menor ou igual ao número de observações, **1** – maior que o número de observações.

Esta subrotina deve ser programa de forma a retornar o erro da observação sendo analisada (**E**), ou seja a diferença entre o valor observado e o valor estimado pelo modelo.

EXEMPLO 7
A taxa de reação química é geralmente dada pela lei de Arrhenius:

$$k = A \cdot \exp\left(\frac{-Ea}{R \cdot T}\right)$$

A	fator pré-exponencial (parâmetro)
Ea	energia de ativação (parâmetro)
k	taxa de reação
R	constante dos gases
T	temperatura

Pode-se estimar o valor do fator pré-exponencial (**A**) e da energia de ativação (**Ea**) obtendo-se valores experimentais de k e T. Por uma questão de maior facilidade matemática, o logaritmo desta equação é avaliado.

$$\ln k = \ln A - \left(\frac{Ea}{R \cdot T}\right)$$

Para ser usada na subrotina, o erro entre o **k** observado e o **k** calculado deve ser obtido, e portanto a subrotina deve calcular a seguinte equação:

$$E = \ln k - \left[\ln A - \left(\frac{Ea}{R \cdot T}\right)\right]$$

Neste caso, **A** e **Ea** são os parâmetros da equação e **k** e **T** são as variáveis conhecidas. Podemos transformar **A** em **THETA(1)** e **Ea** em **THETA(2)** de forma que possam ser avaliadas pela subrotina. **k** e **T**, por sua vez serão transformados em **Y(1)** e **Y(2)**. Como existem várias observações, cada par de observações de **k** e **T**, serão armazenadas em **Y(1,IOBS)** e **Y(2,IOBS)**.
Portanto a equação a ser programada deverá ser:

$$E = \ln\{Y(1,IOBS)\} - \left[\ln\{THETA(1)\} - \left(\frac{THETA(2)}{R \cdot Y(2,IOBS)}\right)\right]$$

ou em Fortran:

```
E    = DLOG(Y(1,IOBS)) - (DLOG(THETA(1)) - THETA(2)/(R*Y(2,IOBS)))
```

Estrutura Geral do Programa

A estrutura geral de um programa de integração usando a **DRNLIN** tem a forma:

```
MODULE GLOBAL
    REAL*8 Y(10,1000), <variáveis>
!       SE TIVER MAIS QUE 10 VARIÁVEIS, MUDAR O NÚMERO DE
!       Y(<campo>,1000)
!       SE TIVER MAIS QUE 1000 OBSERVAÇÕES, MUDAR O NÚMERO DE
!       Y(10,<observações>)
    INTEGER NOBS, <variáveis>
END MODULE

! PROGRAMA REGRESSÃO NÃO LINEAR USANDO BIBLIOTECA IMSL
PROGRAM <programa>
    USE GLOBAL
    USE IMSL
    IMPLICIT REAL*8 (A-H,O-Z)
    PARAMETER (NPRM = <número de parâmetros>)
    DIMENSION THETA(NPRM), R(NPRM,NPRM)
    EXTERNAL FCNMOD

!   ABERTURA DE ARQUIVO DE DADOS
    OPEN(2,FILE='<arquivo de dados>')

!   INICIALIZAÇAO DAS VARIÁVEIS
    <variáveis> = <valor>
```



```
!   LEITURA DE DADOS DO ARQUIVO
    NOBS = 0
    DO WHILE (.NOT.(EOF(2)))
        NOBS = NOBS + 1
        READ(2,*) Y(1,NOBS), Y(2,NOBS), Y(<campo>,NOBS)
!       Y(<campo>,i) = SÃO AS VARIÁVEIS CUJOS VALORES SÃO CONHECIDOS
!       PARA CADA OBSERVAÇÃO
    ENDDO

!   "CHUTE" INICIAL PARA OS PARÂMETROS
    THETA(<parametro>) = <valor>          ! PARÂMETRO DA EQUAÇÃO

!   INICIALIZAÇÃO DAS CONDIÇÕES DA REGRESSÃO LINEAR
    IOPT = 0                              ! OPÇÃO DE AVALIAÇÃO DA FUNÇÃO

    CALL DRNLIN(FCNMOD, NPRM, IOPT, THETA, R, NPRM, IRANK, DFE, SSE)

!   IMPRESSÃO DOS RESULTADOS
    WRITE(*,*) <parametros>
END

! SUBROTINA COM A FUNÇÃO DA TAXA DE REAÇÃO
SUBROUTINE FCNMOD(NPRM, THETA, IOPT, IOBS, FRQ, WT, E, DE, IEND)
    USE GLOBAL
    IMPLICIT REAL*8 (A-H,O-Z)
    DIMENSION THETA(NPRM), R(NPRM,NPRM)

    IEND = 0
    IF (IOBS <= NOBS) THEN
        WT   = 1.0D0          ! PESO DA OBSERVAÇÃO
        FRQ  = 1.0D0
        E    = <função>
    ELSE
!       ACABARAM-SE AS OBSERVAÇÕES
        IEND = 1
    END IF
END   SUBROUTINE
```

EXEMPLO 8
Um programa para estimar as constantes da equação da lei de Arrhenius, apresentada no Exemplo 7 teria a forma:

```
MODULE GLOBAL
    REAL*8 CTEGAS, Y(10,1000)
    INTEGER NOBS
END MODULE
```

```
! PROGRAMA PARA CALCULO DA ENERGIA DE ATIVAÇÃO E
! FATOR PRÉ-EXPONENCIAL DE UMA REAÇÃO QUÍMICA
PROGRAM CINET01
    USE GLOBAL
    USE IMSL
    IMPLICIT REAL*8 (A-H,O-Z)
    PARAMETER (NPRM = 2)              ! NPRM = NÚMERO DE PARÂMETROS
    DIMENSION THETA(NPRM), R(NPRM,NPRM)
    EXTERNAL FCNMOD

!   ABERTURA DE ARQUIVO DE DADOS
    OPEN(2,FILE='CINET.DAT')

!   INICIALIZAÇAO DAS VARIÁVEIS
    CTEGAS = 8.314D0                  ! CONSTANTE DOS GASES

!   LEITURA DE DADOS DO ARQUIVO
    NOBS = 0
    DO WHILE (.NOT.(EOF(2)))
        NOBS = NOBS + 1
        READ(2,*) Y(1,NOBS), Y(2,NOBS)
!       Y(1,i) = K - TAXA DE REAÇÃO
!       Y(2,i) = T - TEMPERATURA
    ENDDO

!   "CHUTE" INICIAL PARA OS PARÂMETROS
    THETA(1) = 4.6D16                 ! FATOR PRÉ-EXPONENCIAL
    THETA(2) = 100000.0D0             ! ENERGIA DE ATIVAÇÃO

!   INICIALIZAÇÃO DAS CONDIÇÕES DA REGRESSÃO LINEAR
    IOPT = 0                          ! OPÇÃO DE AVALIAÇÃO DA FUNÇÃO

    CALL DRNLIN(FCNMOD, NPRM, IOPT, THETA, R, NPRM, IRANK, DFE, SSE)

!   IMPRESSÃO DOS RESULTADOS
    WRITE(*,*) 'K  = ', THETA(1)
    WRITE(*,*) 'EA = ', THETA(2)
    WRITE(*,*)
    WRITE(*,*) 'SSE = ', SSE
END

! SUBROTINA COM A FUNÇÃO DA TAXA DE REAÇÃO
SUBROUTINE FCNMOD(NPRM, THETA, IOPT, IOBS, FRQ, WT, E, DE, IEND)
    USE GLOBAL
    IMPLICIT REAL*8 (A-H,O-Z)
    DIMENSION THETA(NPRM), R(NPRM,NPRM)

    IEND = 0
    IF (IOBS <= NOBS) THEN
        WT   = 1.0D0          ! PESO DA OBSERVAÇÃO
        FRQ  = 1.0D0
```



```
        E    = DLOG(Y(1,IOBS)) - (DLOG(THETA(1)) - THETA(2)/(CTEGAS*Y(2,IOBS)))
    ELSE
!       ACABARAM-SE AS OBSERVAÇÕES
        IEND = 1
    END IF

END SUBROUTINE
```

12.5. Estimativa de Parâmetros

As técnicas de regressão são úteis para estimar parâmetros de uma única equação, mas quando se deseja estimar parâmetros de um sistema de equações, deve-se utilizar de técnicas mais avançadas de estimativa de parâmetros, sendo que as mais comuns são baseadas na técnica de Levenberg-Marquardt que estima parâmetros usando o método de mínimos quadrados não linear.

12.5.1. Usando IMSL

A subrotina mais comum para estimar parâmetros no IMSL é a DBCLSF, baseada na técnica de Levenberg-Marquardt.

A chamada desta subrotina tem a seguinte estrutura:

```
DBCLSF(<modelo>, NOBS*NEQ, NPRM, THETA0, IBTYPE, XLB, XUB,
       XSCALE, FSCALE, IPARAM, RPARAM, THETA, FVEC,
       FJAC, LDFJAC)
```

onde	<modelo>	nome da subrotina que contém a função a ser minimizada
	NOBS	número de observações
	NEQ	número de equações no modelo matemático
	NPRM	número de parâmetros a ser estimados
	THETA0	"chute" inicial para os parâmetros
	IBTYPE	tipo de restrição aplicado aos parâmetros: **0** – limites são especificados pelo usuário via XLB e XUB; **1** – os parâmetros são todos positivos; **2** – os parâmetros são todos negativos.
	XLB	vetor com os limites inferiores para os parâmetros
	XUB	vetor com os limites superiores para os parâmetros
	XSCALE	vetor com o valor de escalonamento dos parâmetros
	FSCALE	vetor com o valor de escalonamento para as funções
	IPARAM	vetor com as opções de configuração da subrotina

RPARAM	vetor com as opções de configuração da subrotina
THETA	vetor com os parâmetros que foram estimados pela subrotina
FVEC	vetor que contém os resíduos da solução
FJAC	matriz que contém o Jacobiano da solução
LDFJAC	dimensão de FJAC

Subrotina do Modelo Matemático

A subrotina que contém o modelo matemático a ser integrado tem a seguinte estrutura:

```
SUBROUTINE <modelo> (M, NPRM, THETA, ERRO)
```

onde	M	número de observações * número de equações
	NPRM	número de parâmetros
	THETA	vetor com o valor dos parâmetros
	ERRO	vetor que retorna o valor do erro entre a variável dependente observada e a variável dependente calculada com os valores atuais dos parâmetros sendo estimados.

Esta subrotina deve ser programa de forma a retornar o erro da observação sendo analisada (**E**), ou seja a diferença entre o valor observado e o valor estimado pelo modelo. Em geral, para estimativa de parâmetros se utiliza a diferença ao quadrado.

```
ERRO(<obs>) = (YOBS(I,<obs>) - YSIM(J,<obs>)**2.0D0
```

onde	YOBS	valor observado
	YSIM	valor calculado com o valor atual dos parâmetros sendo estimados pela subrotina.

Estrutura Geral do Programa

A estrutura geral de um programa de integração usando a **DBCLSF** tem a forma:

```
MODULE GLOBAL
    INTEGER NRUN,NEQT,NOBT,IMOD
    REAL*8 XOBS(1000), YOBS(10,1000), YSIM(10,1000)
```



```
!       SE TIVER MAIS QUE 10 VARIÁVEIS DEPENDENTES,
!       MUDAR O NÚMERO DE Y(<campo>,1000)
!       SE TIVER MAIS QUE 1000 OBSERVAÇÕES, MUDAR O NÚMERO DE
!       Y(10,<observações>) E X(<observações>)
    REAL*8 TTA(50)
END MODULE


! PROGRAMA PARA ESTIMATIVA DE PARÂMETROS
PROGRAM DEXPRM
    USE IMSL
    USE GLOBAL
    IMPLICIT REAL*8 (A-H,O-Z)

    ! NPRM = NÚMERO DE PARÂMETROS
    ! NOBS = NÚMERO DE OBSERVAÇÕES
    ! NEQ  = NÚMERO DE EQUAÇÕES DO MODELO
    PARAMETER (NPRM = <valor>, NOBS = <valor>, NEQ = <valor>)

    DIMENSION THETA(NPRM), THETA0(NPRM), R(NPRM,NPRM)
    DIMENSION XLB(NPRM), XUB(NPRM), XSCALE(NPRM)
    DIMENSION IPARAM(6), RPARAM(7)
    DIMENSION FSCALE(NOBS*NEQ),FVEC(NOBS*NEQ),FJAC(NOBS*NEQ,NPRM)

    EXTERNAL FCNPRM         ! SUBROTINA QUE IRÁ CONTROLAR
                            ! QUAL OBSERVAÇÃO SERÁ USADA NA
                            ! ESTIMATIVA DE PARÂMETROS
!   TRANSFERE OS VALORES DO NÚMERO DE EQUAÇÕES DO MODELO E DO
!   NÚMERO DE OBSERVAÇÕES QUE SERÁ USADO NA ESTIMATIVA DOS
!   PARÂMETROS. ISTO É FEITO POIS AS VARIÁVEIS NEQ E NOBS
!   NÃO PODEM SER PASSADAS DIRETAMENTE PARA A SUBROTINA FCNPRM E
!   PARA A SUBROTINA <MODELO>
    NEQT = NEQ
    NOBT = NOBS

!   ABRE O ARQUIVO QUE CONTÉM OS DADOS
    OPEN(2, FILE = '<arquivo>')

!   CHUTE INICIAL DOS PARÂMETROS
    THETA0(<param>) = <valor>

!   LEITURA DOS PONTOS EXPERIMENTAIS
    DO I = 1,NOBS
!       INSIRA O NÚMERO DE VARIÁVEIS QUE FOR NECESSÁRIO
!       XOBS - VARIÁVEL INDEPENDENTE
!       YOBS - VARIÁVEL DEPENDENTE
        READ(3,*) XOBS(I),YOBS(<campo>,I),YOBS(<campo>,I)
    ENDDO

!   DEFINE O TIPO DE MODELO MATEMÁTICO
    IMOD = 1                ! IMOD = 1 SE MODELO DIFERENCIAL
                            !           NECESSITA SER INTEGRADO
```

```
                                ! IMOD = 2 SE MODELO ALGÉBRICO

!   CHAMA SUBROTINA QUE INICIALIZA AS CONDIÇÕES PARA O
!   MÉTODO DE ESTIMATIVA DE PARÂMETROS
    FSCALE = 1.0D0
    XSCALE = 1.0D0
    NRUN = 0
    LDFJAC = NOBS*NEQ
    CALL DU4LSF(IPARAM,RPARAM)      ! INICIALIZA CONFIGURAÇÃO DA DBCLSF

    IPARAM(1) = 1
    IPARAM(3) = 10000
    IPARAM(4) = 1000
    IPARAM(5) = 1000

!   CHAMA SUBROTINA PARA ESTIMATIVA DOS PARÂMETROS DO MODELO
    CALL DBCLSF(FCNPRM,NOBS*NEQ,NPRM,THETA0,1,XLB,XUB,XSCALE, &
                        FSCALE,IPARAM,RPARAM,THETA,FVEC,FJAC,LDFJAC)

!   IMPRIME OS PARÂMETROS QUE FORAM ESTIMADOS
    DO I = 1,NPRM
        WRITE(*,*) THETA(I)
    ENDDO
END


!   SUBROTINA DE MINIMIZAÇÃO
SUBROUTINE FCNPRM(NPTS, NPRM, THETA, ERRO)
    USE GLOBAL
    IMPLICIT REAL*8 (A-H,O-Z)
    DIMENSION THETA(NPRM), R(NPRM,NPRM)
    DIMENSION PARAM(50)                             ! USADO PELA INTEGRAÇÃO
    DIMENSION C(NPTS), ERRO(NPTS)

    EXTERNAL FCNMOD

!   TRANSFERE OS VALORES DOS PARÂMETROS SENDO ESTIMADOS PARA
!   A VARIÁVEL GLOBAL QUE SERÁ PASSADA PARA A SUBROTINA
!   QUE CONTÉM O MODELO. ISTO É FEITO POIS A VARIÁVEL THETA
!   NÃO PODE SER PASSADA DIRETAMENTE PARA A SUBROTINA <MODELO>
    DO I = 1,NPRM
        TTA(I) = THETA(I)
    ENDDO

    IF (IMOD == 1) THEN
!   MODELO DIFERENCIAL
!   FAZ INTEGRAÇÃO DO MODELO
        DO I = 1,NOBT
            IF (XOBS(I) == 0.0D0) THEN
!               PEGA O VALOR INICIAL PARA A INTEGRAÇÃO NUMÉRICA
                DO J = 1,NEQT
                    C(J) = YOBS(J,I)
```



```
                    YSIM(J,I) = YOBS(J,I)
                ENDDO

!               INICIALIZA PARÂMETROS PARA A INTEGRAÇÃO NUMÉRICA
                ATOL = 1.0D0
                IDO = 1
                PARAM = 0.0D0
                PARAM(4) = 1000000
                T = 0.0D0
            ENDIF

!           CHAMA SUBROTINA DE INTEGRAÇÃO NUMÉRICA
            IF (XOBS(I+1) /= 0.0D0) THEN
                TOUT = XOBS(I+1)
                CALL DIVPRK (IDO, NEQT, FCNMOD, T, TOUT, ATOL, PARAM, C)
                DO J = 1,NEQT
                    YSIM(J,I+1) = C(J)
                ENDDO
            ELSE
                CALL DIVPRK (3, NEQT, FCNMOD, T, TOUT, ATOL, PARAM, C)
            ENDIF
        ENDDO
    ELSE

!   MODELO ALGÉBRICO
!       ESCREVA AQUI AS EQUAÇÕES ALGÉBRICAS DO MODELO
!       YSIM(<campo>,I) É A VARIÁVEL INDEPENDENTE SENDO CALCULADA
        DO I = 1,NOBT
            YSIM(<campo>,I) = <equação>
        ENDDO
    ENDIF

!   CALCULO O ERRO DO MODELO
    PESO = 1.0D0
    I = 0
    DO K = 1,NOBT
        DO J = 1,NEQT
            I = I + 1
            ERRO(I) = (YOBS(J,K) - YSIM(J,K))**2.0D0
        ENDDO
    ENDDO
END SUBROUTINE


! SUBROTINA QUE CONTÉM AS EQUAÇÕES DIFERENCIAIS
SUBROUTINE FCNMOD(NEQ,T,Y,YPRIME)
    USE GLOBAL
  IMPLICIT REAL*8 (A-H,O-Z)
  DIMENSION Y(NEQ), YPRIME(NEQ)

!   INICIALIZAÇÃO DAS VARIÁVEIS
    <variaveis> = <valor>
```

```
!   MODELO DIFERENCIAL
!       ESCREVA AQUI AS EQUAÇÕES DIFERENCIAIS ONDE
!       YPRIME(<campo>) = DIFERENCIAL DA VARIAVEL Y(<campo>)
    YPRIME(<campo>) = <equação>

END SUBROUTINE
```

A subrotina **DBCLSF** estima parâmetros com base no erro de cada observação, independente do número de equações do modelo. É por isso que deve-se prestar atenção na diferença entre as variáveis **NPTS** e **NOBT** na subrotina **FCNPRM**. Nesta subrotina, a variável **NOBT** controla o número de conjuntos de observações, enquanto que **NPTS** controla o total de observações, ou seja **NOBT*NEQT** (número de conjunto de observações * número de equações do modelo).

Se tivermos dados de uma variável independente **X** e duas variáveis dependentes **Y1** e **Y2** (portanto duas equações). Então um conjunto de observação é composto dos valores de **X**, **Y1**, **Y2**. Já uma observação é o valor individual de **Y1** ou **Y2**.

A variável **IMOD** controla o tipo de modelo matemático: **1** para modelo diferencial e **2** para modelo algébrico.

Se o modelo for diferencial (**IMOD** = 1), o programa irá integrar o modelo diferencial de forma a obter os resultados dos modelo para as variáveis dependente. O arquivo de dados deve conter em cada linha os valores para as observações feitas para a variável independente e variáveis dependentes na sequência em que foram obtidos. Pode-se ter várias corridas experimentais para a integração, deste que a condição inicial seja marcada com **X** = 0. O quadro 12.1. mostra um exemplo de arquivo de dados para modelo diferencial, onde a primeira coluna se refere à variável independente **X**, e a segunda e terceira colunas às variáveis dependentes **Y(1)** e **Y(2)**.

Quadro 12.1. Exemplo de arquivo de dados para modelo diferencial.

0.0	12.1	0.0	
1.0	11.9	2.1	
2.0	10.5	3.5	
3.0	9.4	4.4	
0.0	13.9	0.0	⇐ X = 0.0 marca o início de um novo experimento e portanto de o início de uma nova integração.
1.5	12.3	2.5	
1.9	12.1	2.9	
3.0	10.3	4.0	
4.0	9.0	5.4	

Se o modelo for algébrico (**IMOD** = 2), o arquivo de dados não necessita ter as condições iniciais do sistema. O quadro 12.2. mostra um exemplo de arquivo de dados para modelo algébrico, onde a primeira coluna se refere à variável independente **X**, e a segunda e terceira colunas às variáveis dependentes **Y(1)** e **Y(2)**.

Quadro 12.2. Exemplo de arquivo de dados para modelo algébrico.

1.0	11.9	2.1
2.0	10.5	3.5
3.0	9.4	4.4
1.5	12.3	2.5
1.9	12.1	2.9
3.4	10.3	4.0
4.2	9.0	5.4

Dependendo do modelo utilizado, pode-se modificar o programa acima, acrescentando mais variáveis independentes e dependentes. O único cuidado que se deve ter é saber qual variável independente irá controlar a integração no caso de modelo diferencial.

EXERCÍCIOS

EXERCÍCIO 1
Um experimento para obter a pressão parcial de tolueno obteve os seguinte dados:

Pressão de Vapor	Temperatura
1	-26.7
5	-4.4
10	6.4
20	18.4
40	31.8
60	40.3
100	51.9
200	69.5
400	89.5
760	110.6

Desenvolva um programa para calcular os parâmetros da equação de Antoine para os dados acima.

$$P_{vap} = \exp\left(A + \frac{B}{T + 273} \right)$$

A e B	parâmetros
P_{vap}	pressão de vapor (mmHg)
T	temperatura em ºC

EXERCÍCIO 2

Uma reação de hidrogenação do benzeno é realizada em um reator tubular operando de forma adiabática.

O balanço de massa (adimensionalizado) é dado por:

$$\frac{dy}{dx} = -0{,}1744 \cdot \exp\left(\frac{3{,}21}{T^*} \right) \cdot y$$

O balanço de energia (adimensionalizado) é dado por:

$$\frac{dT^*}{dx} = -0{,}06984 \cdot \exp\left(\frac{3{,}21}{T^*} \right) \cdot y$$

$$T^* = \frac{T}{T_0}$$

T	temperatura em K
T_0	temperatura inicial = 423 K
T*	temperatura adimensional
y	concentração de benzeno adimensional
x	comprimento do reator adimensional

Condições iniciais:

em x = 0 → y = 1 e T* = 1

Calcular a temperatura real e a concentração adimensional em função do comprimento do reator (x entre 0 e 1, com intervalo de impressão de 0,1).

13. ERROS DE COMPILAÇÃO

Muitos erros podem ocorrer durante a compilação do programa (geração do programa executável), sendo que a maioria se deve a falta de algum comando ou erro de digitação.

Durante a compilação do programa, os erros aparecem em uma janela separada do código do programa (Figura 13.1).

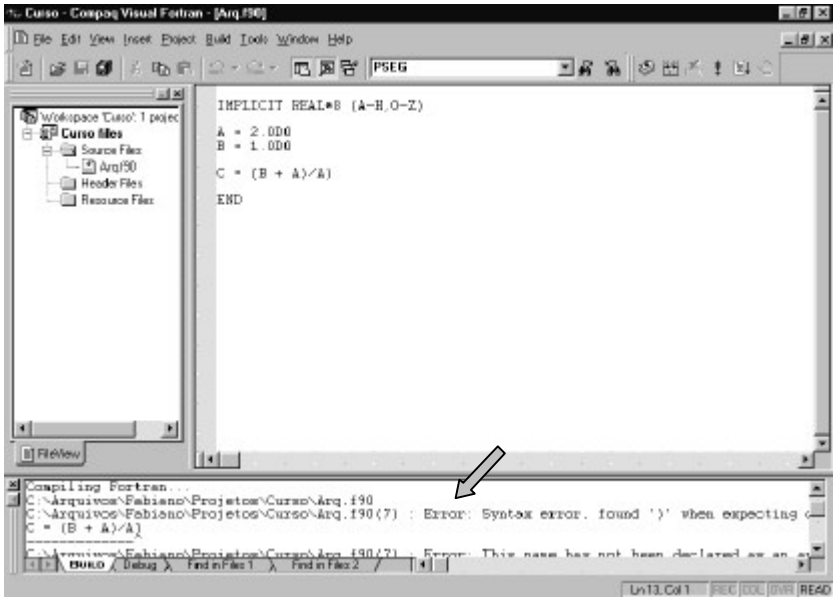


Figura 13.1. Tela do programa e local onde as mensagens de erro são listadas.

As mensagens que aparecem tem a forma:

```
C:\Arquivos\Arq.f90(6) : Warning: Variable A is used before its
value has been defined
C = B/A
------^
```

onde:	`C:\Arquivos\Arq.f90`	é o diretório e arquivo onde ocorreu o erro
	`(6)`	linha do programa onde ocorreu o erro
	`Warning`	tipo de erro. Pode ser *Warning* (o compilador cria o programa executável, mas poderá ocorrer algum erro durante sua execução) ou *Error* (erro grave – compilador não cria o programa executável)
	`Variable A is …`	Descrição do erro
	`C = B/A` `------^`	Cópia da linha do erro e indicação onde o erro foi detectado

Abaixo listamos as principais mensagens de erro de compilação, a causa provável do erro e como consertar o problema.

```
Error: A logical data type is required in this context.
IF (C = 0) THEN
------^
```

Operador lógico está incorreto (falta um =). O certo é ==.

```
Error: A logical data type is required in this context.
IF ((C == 0) OR (A == 0)) THEN
-------------^
```

Operador lógico está incorreto (OR). O certo é **.OR.**
Pode ocorrer com **.AND.** também.

```
Error: An ENDIF occurred without a corresponding IF THEN or ELSE
statement.
ENDIF
^
```

Falta o comando **THEN** na estrutura **IF..THEN..ELSE**

```
Error: An unterminated block exists.
IF (C == 0.0D0) THEN
^
```

Falta um **ENDIF** no bloco **IF..THEN..ELSE**.
Correção:
Procure o final do **IF..THEN..ELSE** e insira o comando **ENDIF**.

```
Error: An unterminated block exists.
DO I = 1,100
^
```

Falta um **ENDDO** no bloco **DO..ENDDO**

Correção:

Procure o final do **DO..ENDDO** e insira o comando **ENDDO**.

```
Error: Syntax error, found '=' when expecting one of: ( * :: ,
<END-OF-STATEMENT> ; : ) + . - % (/ [ ] /) . ** / > ...
IF (C = 0) THEN
------^
```

Operador lógico está incorreto (falta um =). O certo é **==**.

```
Error: Syntax error, found '.' when expecting one of: <LABEL>
<END-OF-STATEMENT> ; BLOCK BLOCKDATA PROGRAM TYPE COMPLEX BYTE
CHARACTER ...
.
^
```

Tem um ponto final “perdido”em alguma linha do programa.

Correção:

Verifique a linha do problema e remova o ponto final.

```
Error: Syntax error, found ',' when expecting one of: <END-OF-
STATEMENT> ;
A = 2,0D0
-----^
```

Número foi digitado errado (2,0D0). O certo é 2.0D0 (com ponto ao invés de vírgula).

```
Error: Syntax error, found END-OF-FILE when expecting one of:
<LABEL> <END-OF-STATEMENT> ; BLOCK BLOCKDATA PROGRAM TYPE COMPLEX
BYTE CHARACTER ...
```

Falta um **END** no final do programa principal.

```
Error: Syntax error, found END-OF-STATEMENT when expecting one
of: , )
C = ((B + A)/A
-------------^
```

Falta um parênteses na equação.

Correção:

Verifique em que ponto da equação está faltando um parênteses.


```
Error: The number of subscripts is incorrect.   [A]
A(10) = 2.0D0
^
```

A variável **A** foi definida como uma matriz A(x,y) e foi usada como um vetor A(x).

```
Error: The statement following the Logical IF statement is
invalid.
IF (C == 0.0D0)
^
```

Falta o comando **THEN** na estrutura **IF..THEN..ELSE**

```
Error: This DO variable has already been used as an outer DO
variable in the same nesting structure.   [I]
 DO I = 1,50
-----------^
```

A variável de controle (**I**) do **DO..ENDDO** já está sendo usada por outro **DO..ENDDO**.

Correção:

Dê outro nome para a variável de controle deste **DO..ENDDO**.

```
Warning: In the call to SOMA, actual argument #1 does not match
the type and kind of the corresponding dummy argument.
CALL SOMA(I,A,B,C)
^
```

A subrotina **SOMA** foi definida como:

SUBROUTINE SOMA(I,A,B,C)

A variável **I** (argumento #1) por sua vez foi declarada como inteiro no programa principal e como real na subrotina.

Correção:

Modifique o tipo da variável **I** no programa principal ou na subrotina, pois as variáveis passadas para a subrotina devem ser de mesmo tipo no programa principal e na subrotina.

```
Warning: In the call to SOMA, there is no actual argument
corresponding to the dummy argument C.
CALL SOMA(A,B)
^
```

A subrotina **SOMA** foi definida como:

SUBROUTINE SOMA(A,B,C)

porém a subrotina foi chamada somente com os parâmetros A e B, faltando o parâmetro C.

Correção:

Procure o parâmetro que está faltando e insira-o na chamada da subrotina.

```
Warning: This statement function has not been used.   [A]
A(1) = 2.0D0
^
```

A variável **A** não foi declarada como um vetor ou matriz.

Correção:

Declare a variável **A** como um vetor usando o comando **DIMENSION**.

```
Warning: Variable A is used before its value has been defined
C = B/A
------^
```

A variável **A**, usada no cálculo da variável C não foi inicializada antes de ser usada para calcular **C**, e é a primeira vez que **A** aparece no programa. A variável **A** portanto contém o valor zero, podendo ser um fator que causará erro no cálculo da variável **C**.

Correção:

Verifique se o nome da variável foi digitado corretamente.

Inicialize a variável com o valor apropriado.

13.1. Erros de Execução

A maioria dos erros de execução dependem de uma análise mais profunda de sua causa, e serão explicados no Capítulo 14.

Quando um erro de execução ocorre, a tela apresentada geralmente é parecida com a mostrada na Figura 13.2.

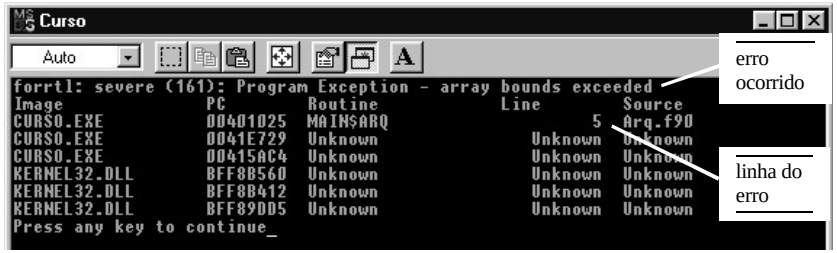


Figura 3.2. Tela do programa quando ocorre erro de execução

No caso acima, o erro se deve a um erro de programação.

```
Severe(161): Program Exception – array bounds exceeded
```

Ocorre quando tenta-se usar um campo inexistente do vetor ou matriz. Por exemplo: um vetor dimensionado é como A(5), mas em algum lugar do programa tenta-se usar o valor de A(6), sendo que o campo 6 não existe.

14. DEBUG

Debugar significa remover erros de programação que ocorrem durante a execução de um programa.

14.1. Quando Debugar

- quando o programa parece não sair do lugar.
- quando ocorre divisão por zero ou outro erro matemático grave.
- quando o resultado numérico é NAN (*Not a Number*)
- quando o resultado retornado é estranho.

14.2. Antes de Debugar

Debugar toma tempo, principalmente quando não se sabe o que se está procurando. A primeira coisa antes de debugar, é parar, pensar e refletir em qual a causa mais provável do erro.

14.3. Problemas que Causam Problemas

14.3.1. Programa Parece Não Sair do Lugar

É geralmente causado por *loop* infinito.

Procure todos os **DO..ENDDO** e **DO WHILE** e reveja a condição de saída do *loop*. Verifique se ela esta correta. Caso esteja, procure a variável usada na comparação e veja porque seu valor não muda.

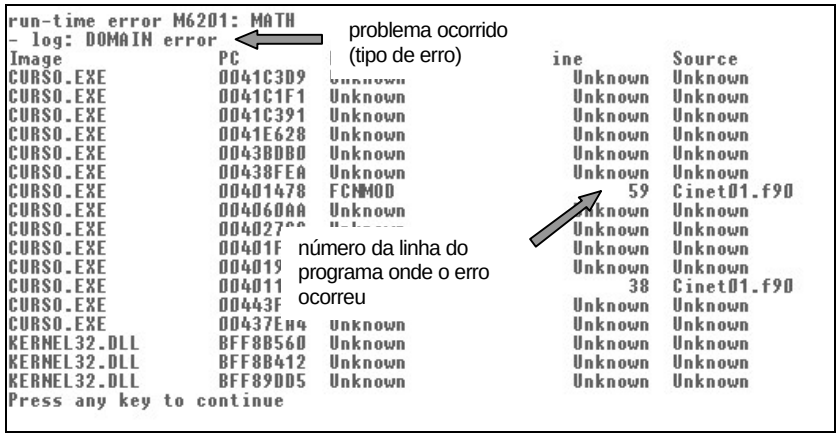
As vezes é necessário repensar a condição de saída. Ela pode ser muito "radical" sendo que o processo pode não gerar tal valor esperado para a variável. Ocorre muito quando se estabelece uma tolerância muito rígida ou pequena demais.

14.3.2. Ocorre Divisão por Zero / Erro em Logaritmo

Quando ocorre divisão por zero, erro em logaritmo ou exponencial geralmente o programa exibe uma mensagem informando a linha onde o problema ocorreu (Figura 14.1).

Vá até esta linha de programa e procure qual variável pode ter valor zero (divisor). Procure no programa o porque ela tem valor zero. Em geral é devido ao uso de uma variável não inicializada (quer portanto tem valor zero). Ou uma falha na sua inicialização ou cálculo.

Quando ocorre erro em logaritmo, o procedimento é o mesmo.




```
run-time error M6201: MATH
- log: DOMAIN error
Image              PC                Routine            Line        Source
CURSO.EXE          00041C3D9         Unknown            Unknown     Unknown
CURSO.EXE          00041C1F1         Unknown            Unknown     Unknown
CURSO.EXE          00041C391         Unknown            Unknown     Unknown
CURSO.EXE          00041E628         Unknown            Unknown     Unknown
CURSO.EXE          00043BDB0         Unknown            Unknown     Unknown
CURSO.EXE          00438FEA          Unknown            Unknown     Unknown
CURSO.EXE          00401478          FCNMOD                   59    Cinet01.f90
CURSO.EXE          00406DAA          Unknown            Unknown     Unknown
CURSO.EXE          004027C0          Unknown            Unknown     Unknown
CURSO.EXE          00401F            Unknown            Unknown     Unknown
CURSO.EXE          004019            Unknown            Unknown     Unknown
CURSO.EXE          004011                                    38    Cinet01.f90
CURSO.EXE          00443F            Unknown            Unknown     Unknown
CURSO.EXE          00437EH4          Unknown            Unknown     Unknown
KERNEL32.DLL       BFF8B560          Unknown            Unknown     Unknown
KERNEL32.DLL       BFF8B412          Unknown            Unknown     Unknown
KERNEL32.DLL       BFF89DD5          Unknown            Unknown     Unknown
Press any key to continue
```

Figura 14.1. Exemplo da tela com a mensagem de erro devido a erro no logaritmo.

14.3.3. Overflow ou Número Infinito

Ocorre quando uma variável ou cálculo retorna um valor muito maior do que a variável consegue armazenar (*overflow*). Alguns compiladores param a execução quando há *overflow*, e outros atribuem o nome "*Infinity*" para a variável, sendo que a partir deste momento nenhum cálculo pode ser realizado com esta variável.

Geralmente, o *overflow* ocorre com cálculo com exponenciais (a exponencial de um número grande é um número maior ainda) ou em cálculos com somatórios que não são inicializados corretamente.

Quando o problema ocorre com a exponencial. Procure pela variável que causa o problema e veja porque esta variável está com um valor tão grande.

Quando o problema é com o somatório, verifique se o cálculo do somatório foi inicializado.

Certo	Errado
SUM = 0.0D0 DO I = 1,100 SUM = SUM + X(I) ENDDO	DO I = 1,100 SUM = SUM + X(I) ENDDO

Se um somatório deste tipo existe num programa, no caso *Certo*, a variável **SUM** começa com zero e então é realizado o somatório. Se o programa reutiliza este código, no caso *Certo*, **SUM** começa com o valor zero; e no caso *Errado*, **SUM** começa com um número grande (resultado do último somatório) podendo resultar num futuro *overflow*.

14.3.4. Resultado NAN

Quando subrotinas numéricas do IMSL ou outras são usadas, elas podem conter internamente um sistema de detecção de erro que não deixa que divisões por zero ou erros simples de cálculo causem a interrupção do programa.

Neste caso, quando existe a divisão por zero ou outro erro, esta subrotina intercepta o erro e atribui o código NAN (*Not A Number*) para a variável. Após esta variável receber o código NAN, qualquer outra variável que se utilize do valor NAN em seu cálculo, passa automaticamente a ter o valor NAN.

Para saber onde está a causa do erro, deve-se debugar o modelo matemático utilizado linha por linha. Primeiro, ao entrar na subrotina do modelo, verifique se todas as variáveis estão com o valor correto (as vezes pode haver problema na passagem dos valores do programa principal para a subrotina do modelo – pouco provável se o sistema de módulo de variáveis globais é usado). Depois procure em todas as equações qual gera o primeiro NAN. Pode estar em alguma divisão por zero, exponencial, seno, co-seno ou logaritmo. No primeiro NAN, veja na equação qual a variável que tem um valor que possa gerar o erro matemático.

Finalmente procure o que ocorre com esta variável (cálculo errado da variável, falta de inicialização, erro na leitura, etc.).

14.3.5. Resultado Retornado é Estranho

Pior problema a ser resolvido, pois a fonte do problema é desconhecido. Primeiro revise suas equações matemáticas (se ela foi digitada corretamente, problemas de sinal, etc.). Esta é a fonte de grande parte dos erros de cálculo.

Se as equações estão corretas, divida o programa em seções debugando uma seção de cada vez. Execute o programa até o final da primeira seção e veja se os valores calculados até então estão corretos. Caso estejam, execute o programa até o final da segunda seção e assim por diante. Quando achar um valor estranho, o problema pode estar dentro daquela região do programa.

Verifique se os valores passados para e da subrotina estão corretos. Depois verifique se existe algum **IF..THEN..ELSE** ou **DO..ENDDO** ou **DO WHILE** que está sendo ignorado (condição pode estar falhando).

14.4. Usando o Debug do Compaq Fortran

Antes de iniciar começar o debug de um programa, é necessário definir uma linha na qual a execução do programa irá parar. Para selecionar uma linha, posicione o cursor na linha desejada e pressione no botão *Stop* (botão em forma de mão) (Figura 14.2).

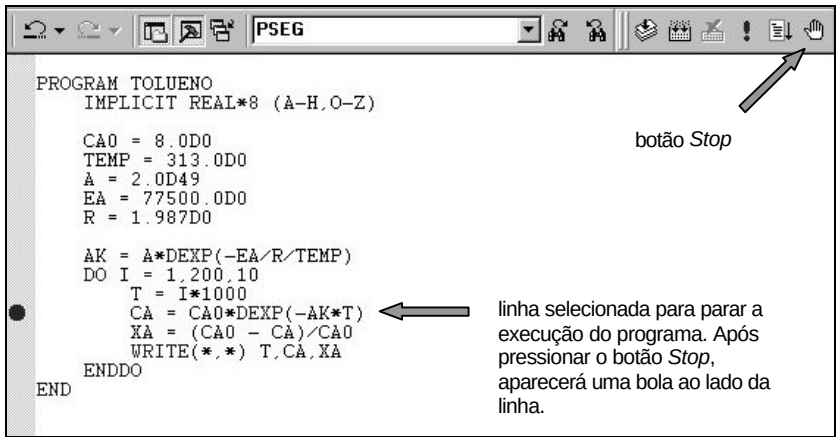


Figura 14.2. Selecionando a linha de parada.

Pode-se definir quanto pontos de parada se desejar.

Para iniciar a sessão de debug, selecione a opção *Build* no menu principal, e depois selecione as opções *Start Debug* e *Go* (Figura 14.3). O programa irá iniciar sua execução e irá parar no ponto escolhido anteriormente.

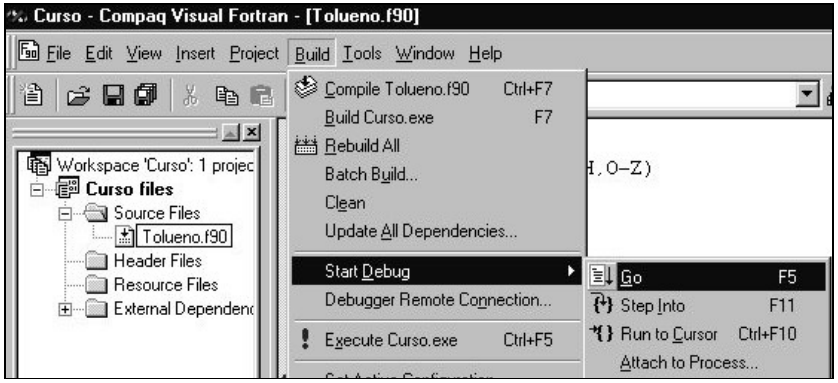


Figura 14.3. Iniciado a seção de debug.

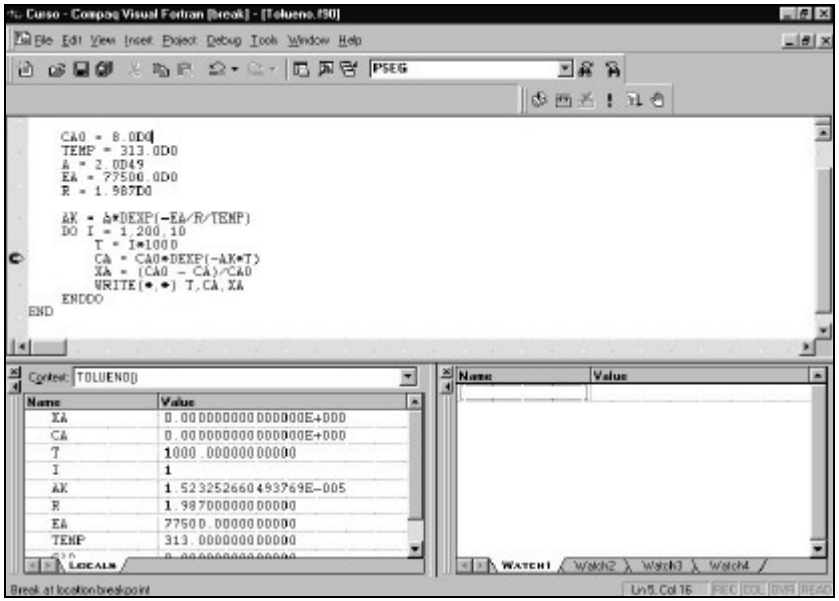


Figura 14.4. Tela de um programa sendo debugado.

Quando o programa para no ponto escolhido para ser debugado, a tela apresentada pelo compilador será semelhante à apresentada na Figura 14.4. Na parte superior da tela será apresentado o código do programa. Na parte inferior serão apresentados, uma relação com todas as variáveis usadas no programa e seus valores (do lado esquerdo), uma lista com variáveis especificadas pelo usuário (do lado direito). No lado direito pode-se escrever qual variável se deseja saber o valor. Passando o cursor em alguma variável no código do programa irá mostrar um pequeno quadrado com o valor desta variável.

Para passar a execução do programa para a próxima linha, tecle **F10**. Para continuar a execução do programa até o próximo ponto de parada, tecle **F5**.