Programação com Octave/Matlab
Coltec - UFMG
Capítulo 7 - Repetições

Márcio Fantini Miranda

28 de junho de 2021

Sumário

1 Repetições

Nesse capítulo veremos mais estruturas de controle de fluxo. Lembre-se que uma programa (ou mesmo um script) de computador é formado por uma sequência de instruções.

Enquanto fazíamos programas (scripts) simples, como ler um dado e plotar um gráfico, não precisávamos de **controle de fluxo** pois não havia necessidade do programa **tomar decisões** ou **efetuar repetições** nem criar laços de repetições para acessar elementos específicos em matrizes ou vetores.

Quando estudamos "decisões"vimos que o controle de fluxo pode ser alterado quando avaliamos uma condição. Usamos a estrutura if-else para apresentar um resultado, de acordo com o resultado de uma operação lógica.

Uma figura que deve sempre estar na sua mente é a do fluxograma (vamos ver mais sobre fluxogramas no nosso curso de programação na linguagem C) da figura 1 abaixo. Nela temos uma alteração (muito simples) no fluxo natural do programa. Essa alteração foi causada por uma decisão, que por sua vez foi tomada a partir de uma **expressão lógica**, que tem como resultado falso ou verdadeiro.

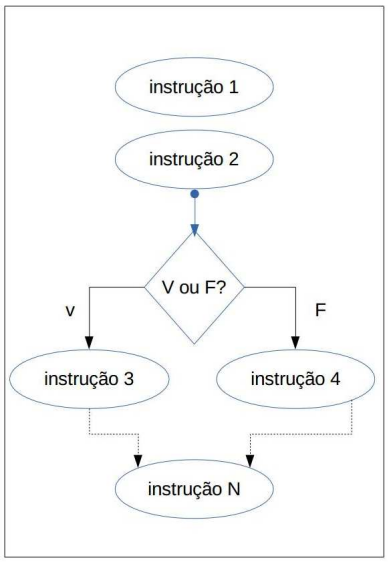

Figura 1: Estrutura de decisão

Continuando o estudo dessas estruturas que possibilitam alterar o fluxo do programa, vamos ver nas seções seguintes estruturas que possibiltam ao progama **repetir ações**. As ações são repetidas **equanto** uma condição for verdadeira (ou parar de ser executada quando uma condição for falsa, por exemplo). É importante entender a lógica que determina a repetição. Lembre-se sempre que temos duas condições para controlar o fluxo: verdadeira ou falsa.

Veremos na subseção 3.2 a estrutura **PARA**. Na seção 3.1 veremos a estrutura **ENQUANTO**. Ambas podem ser entendidas como formas de fazer o programa repetir ações.

2 Estrutura do Programa de Computador

Retomando a nossa ideia sobre a organização do programa de computador, vamos rever o diagrama simplificado, já apresentado várias vezes. Esse diagrama está representado na figura 2. Ele nos mostra como um programa de computador pode, didaticamente, ser organizado.

Figura 2: Diagrama simplificado da estrutura de um programa de computador)

Em linhas gerais, um programa de computador possui uma entrada, uma saída e um processamento, que pode ser dividido em três categorias: operações, decisões e repetições.

Nessa estrutura, uma linguagem de programação pode então ser organizada da seguinte forma:

- Estruturda de entrada e e saída
- Processamento:
  - Operadores e Operações (Lógicas, Aritméticas)
  - Estrutura de Decisões

  - Estrutura para Repetições que veremos no presente Capítulo.

- Dados:
  - Tipos de Dados
  - Estrutura de Dados

Vamos nos ater agora para as formas usadas para repetirmos ações. Preste atenção nos fluxogramas e na estrutura quando falada em português. A representação no programa é simplesmente uma forma de colocarmos na linguagem do computador uma estrutura que usamos naturalmente quando falamos ou escrevemos.

3 Estrutura de Repetição: Loop ou Laço

Em computação o **laço de repetição**, ou simplesmente **laço** (ou ainda **loop**) é uma estrutura que permite que **trechos do programa** sejam executados várias vezes.

Ou, dizendo de outra forma: **Laço ou Loop** é um sequência de instruções (ou uma única instrução) que são (ou que é) executada repetidas vezes, até que uma determinada condição a interrrompa.

O fluxograma dado na figura 3 apresenta a lógica da repetição. Nesse diagrama a ação escreva "estou aprendendno a progamar" é executada **enquanto** a resposta à pergunta "continuar escrevendo" for sim. Ou seja, temos um teste lógico cujo resultado, no caso da língua falada, é sim ou não; e na computação é falso (0) ou verdadeiro (1).

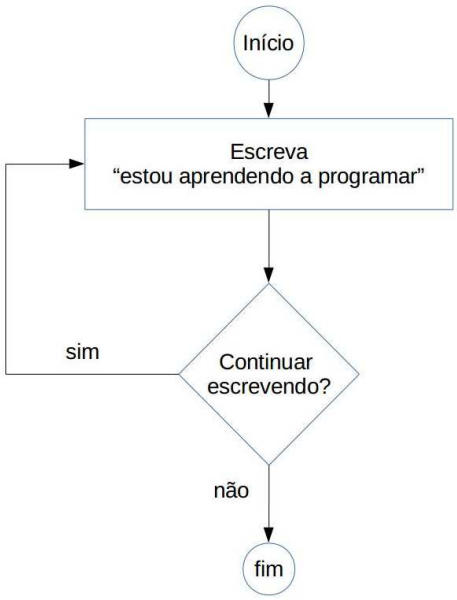

Figura 3: Fluxograma de repetição com decisão a partir de uma opção do usuário)

Ou seja:

- A estrutura de repetição faz com o que o programa fique ”fazendo uma mesma coisa” enquanto uma condição não o ”mandar parar”.

Veja o algoritmo dado a seguir, como um segundo exemplo:

```
opcao = 1
enquanto opcao for igual a 1, faça:
     escreva na tela a frase 'programar é legal'
     leia opcao (1 para não, 0 para sim)
 fim-enquanto
```

Esse trecho de algoritmo podia ser melhorado, sendo escrito mais próximo da linguagem do Octave

```
opcao = 1
enquanto opcao for igual a 1, faça:
     disp('programar é legal')
     opcao = input('continuar? 1 = Sim, 0 = N ão)
 fim-enquanto
```

Nesse caso estamos usando a palavra **enquanto** para dizer que o loop (ou laço) deva continuar enquanto algo diferente não acontença. Essa palavra **enquanto** define uma estrutura de repetição denominada **estrutura while**.

3.1 A Estrutura "while"

O algoritmo do exemplo anterior usou o **enquanto** para definer um estrutura de repetição, que em programação é chamada *estrutura while*. Veja como fica o trecho com while no Octave:

```
opcao = 1;
while(opcao == 1)
   printf("programar é legal\n")
   opcao = input('deseja continuar? (S=1,N=0)');
end
```

Repare que fazemos um teste: se a variável **opcao** for igual a 1 o loop continua. Se não for, o loop para.

Ou seja: **enquanto a variável opcao for 1, continuamos escrevendo a mensagem**.

Portanto, a partir de agora, podemos usar repetições seguindo a estrutura while no Octave, que é:

```
while (condição)
     instruções
     :
     :
end
```

3.2 Estrutura “for”

Uma outra estrutura de repetição é a **estrutura for**, que em português chamamos de estrutura “**para**”.

Ela está relacionada com uma variação que controla o número de repetições.

Considere por exemplo, que desejamos repetir a frase “programar é legal” 10 vezes.

Para usarmos a estrutura “para” definimos uma variável que vai ser usada como contador. Ou seja ela vai contar 10 vezes!

Por exemplo, defina a variável i e a inicie com o valor $i = 1$. Em português diríamos:

```
para i variando de 1 até 10
   escreva a frase "progamar é legal"
```

Ou poderíamos pensar também:

```
para x variando de 0 até 9 faça:
 escreva a mensagem "estou aprendendo a progamar"
```

O importante nessa estrutura é entender que a variável i deve ser **incrementada**.

Veja o fluxograma da figura 4. Nela a variável i é incrementada 10 vezes.

Muita atenção na atribuição:

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

A estrutura for no Octave segue a seguinte notação:

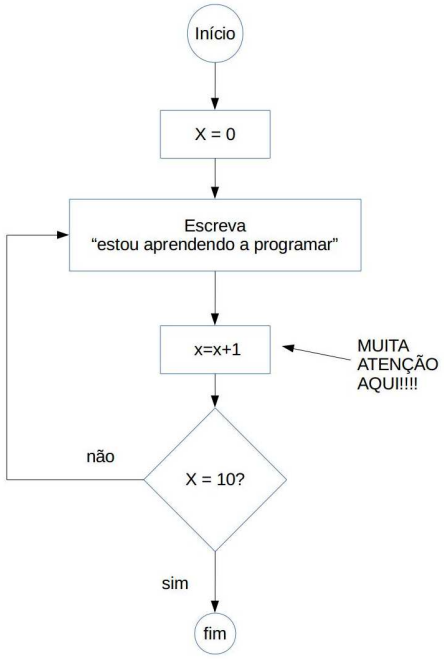

Figura 4: Fluxograma com incremento de variável. A repetição encerra quando a variável atingir 10.)

```
 %exemplo com estrutura for

for x=1:5
 printf("programar é legal\n");
endfor
disp("-----------------------------------------");
for x=1:5
 printf("(%i) programar é muito legal\n",x);
end
```

Veja a saída do programa do slide anterior:

```
octave:5> estrutura_for1
```

```
programar é legal
programar é legal
programar é legal
programar é legal
programar é legal
-----------------------------------------
(1) programar é muito legal
(2) programar é muito legal
(3) programar é muito legal
(4) programar é muito legal
(5) programar é muito legal
octave:6>
```

3.3 Comentários

- Além das estruturas "for"e "while"o Octave possui a estrutura "do-until"(ou faça até).
- Vamos exercitar muitos programas que fazem uso dessas estruturas. A lógica agora será muito importante.
- Em geral os loops são controlados com uma **decisão** (um valor lido, por exemplo) ou por uma **contagem**, ou seja, podemos definir quantas vezes o loop vai ficar repetindo.
- Com essas três estruturas podemos fazer diferentes lógicas de repetição. Algumas vezes é melhor usar **while** do que **for**. Outras ocorre o oposto.

- Atente para a forma de se usar essas estruturas. O conhecimento da linguagem é importante!!!

4 Exemplos

Exemplo 4.1 (Repetição com Contador)

../../../cursoC_2019/apostilaMatlab/exemplos/repeticao_com_contador.m

```
%repeticao para contar
clc
clear all
contador = 1;

while (contador <= 10)
     contador = contador + 1;
     printf("estou contando...essa é a rodada numero
        %i\n",contador);

end

disp("---------------------------------------")
disp('parei de contar');
printf("contador = %i\n",contador);

% existem outras formas de implementar esse contador
% repare o valor final do contador

% o que aconteceu?
```

Exemplo 4.2 (Repetição com Decisão)

../../../cursoC_2019/apostilaMatlab/exemplos/repeticao_com_decisao.m

```
% repeticao com decisao
clear all
clc

opcao = 1;
while(opcao==1)
  printf("programar se aprende programando\n");
  opcao = input('deseja continuar? (sim=1,ãno=0)');
end

%repeticao com decisao usando do-until
opcao=0;
disp(['opcao antes do loop do-until: ',num2str(opcao)])
do
  disp('dentrdo do loop do-until');
  printf('...ãno me canso de repetir...');
  opcao = input('deseja continuar? (sim=1,ãno=0)');
until (opcao==0)

%reparem bem a çdiferena (e étambm a çsemelhana!!!)
```

Exemplo 4.3 (Repetição com Acumulador)

../../../cursoC_2019/apostilaMatlab/exemplos/repeticao_com_acumuladorr.m

```
%repeticao para acumular
clc
clear all

soma = 0;

% vamos somar 1 10 vezes
% SOMA = 1 + 1 + ....

repete=1;
while (soma < 10)
     disp(['estou na rodada únmero: ',num2str(repete)])
     printf("estou somando..a soma vale %i\n",soma);
     soma = soma + 1;
     repete=repete+1;
end

disp("---------------------------------------")
disp('parei de somar...');
printf("soma = %i\n",soma);
disp(['fora do loop while, repete = ',num2str(repete)])
% existem outras formas de implementar esse somador...
```

Exemplo 4.4 (Repetição com Acumulador)

../../../cursoC_2019/apostilaMatlab/exemplos/repeticao_com_acumulador2.m

```
% soma dentro do while

% vamos somar os 10 primeiros numeros
%               naturais a partir do 1
% SOMA = 1 + 2 + 3 + ...
clc; clear all

 c = 1;soma=0;
while (c <= 10)


     soma = soma + c;
     printf("estou contando...essa é a rodada únmero
        %i\n",c);
     printf("estou somando..a soma vale %i\n",soma);
     c = c + 1;

end

disp("---------------------------------------")
disp('parei de somar...');
printf("soma = %i\n",soma);

% existem outras formas de implementar esse somador...
```

Exemplo 4.5 (Estrutura for no Octave)

../../../cursoC_2019/apostilaMatlab/exemplos/estrutura_for1.m

```
1 %exemplo com estrutura for
2
3 for x=1:5
4  printf("programar é legal\n");
5 endfor
6 disp("-------------------------------------------");
7 for x=1:5
8  printf("(%i) programar é muito legal\n",x);
9 end
```

Exemplo 4.6 (Estrutura for com intervalo diferente de 1)

../../../cursoC_2019/apostilaMatlab/exemplos/exemplofor00.m

```
% exemplo com loop for
% intervalo de variacao diferente de 1
% o primeiro varia de 0.5 em 0.5 (incremento)
% o segundo loop for varia de -0.5 em -0.5 (decremento)
i=1;
for x = -2:0.5:2
  printf("x = %2.2f\n",x);
end
disp("----------------------------------------");

for y = 2:-0.5:-2
  printf("y = %2.2f\n",y);
end
```

Exemplo 4.7 (Estrutura for usada com vetor)

../../../cursoC_2019/apostilaMatlab/exemplos/exemplofor01.m

```
1  %exemplo com loop for
2  % intervalo de variacao diferente de 1
3  i=1;
4  for x = -2:0.5:2
5    printf("x = %2.2f\n",x);
6    v(i) = x; % guarda o valor de x na posicao i do vetor v
7    i = i+1;
8  end
9
10 tamanho = length(v);
11
12 % "percorre" o vetor, tomando todas os elementos, a partir
       da variacao de i
13 disp('----------------------------')
14 for i = 1:tamanho
15   printf("v[%i] = %2.2f\n",i,v(i));
16 end
```

5 Repetição para Usar Vetores e Matrizes

Como vimos, os arrays (vetores e matrizes) são um tipo especial de variável que possui muitos valores, que são armazenados ordenadamente, sendo que a cada valor é associado um índice.

Vejamos os vetores $v_{3\times1}$ e $w_{1\times3}$, dados por:

$$v = \begin{bmatrix} 2 \\ -2 \\ 1 \end{bmatrix} \quad w = \begin{bmatrix} 2 & -2 & 1 \end{bmatrix} \tag{1}$$

Na Matemática (ou melhor dizendo, teoricamente) esses vetores são distintos. Ou seja, os elementos deles são acessados com índices diferentes (exceto o elemento (1,1)).

Mas na programação C e Octave, podemos acessar os elementos tanto de v quanto de w da mesma forma. Pois, para essas linguagens, o vetor é visto como um local de armazenamento de dados na memória.

Veja o exemplo a seguir. Nele usamos a repetição for para acessar os elementos tanto de v quanto de w. Entenda bem esse exemplo.

Exemplo 5.1 (Estrutura for usada com vetor. IMPORTANTE!)

../../../cursoC_2019/apostilaMatlab/exemplos/loop_for_com_vetor_01.m

```
1 % usando loop for para % acessar vetores.
2
3 v = [2;2;-1];
4 w = [2,2,-1];
5 disp("apenas para mostrar as õdimenses");
6 disp("dimenssao de v: ");
```

```
7  disp(size(v));
8  disp("dimenssao de w: ");
9  disp(size(w));
10
11 for i=1:3
12   printf("v[%i] = %i   e   w[%i] = %i\n",i,v(i),i,w(i));
13 end
```

É importante que você entenda bem o exemplo 5.1 (obviamente é importante que você entenda TODOS exemplos. Mas esse é especial pois contém vários conceitos juntos).

Importante também entender como trabalhar com matrizes. No caso de vetores tínhamos uma linha ou uma coluna. E quando temos mais de uma linha e mais de uma coluna, como fazer?

Acomapanhe o exemplo 5.2.

Exemplo 5.2 (Estrutura for usada com Matriz. IMPORTANTE!)

../../../cursoC_2019/apostilaMatlab/exemplos/loop_for_com_matriz_01.m

```
1  % usando loop for para % acessar matrizes
2
3
4  M = [2,-1;0,3];
5  N = int16(rand(3,3)*10+1);
6  disp(M);
7  disp("******************************");
8  disp(N);
9  disp("******************************");
10
11 for i=1:2
12   for j=1:2
```

```
13      printf("M(%i,%i) = %i\n",i,j,M(i,j));
14    end
15  end
16  disp("*********************************");
17  for i=1:3
18    for j=1:3
19      printf("N(%i,%i) = %i\n",i,j,N(i,j));
20    end
21  end
```

6 Somatórios

Em várias aplicações em Engenharia, Física, Química e áreas afins é comum precisamos somar diferentes quantidades para calcularmos médias, para sabermos valores totais de processos ou para acumularmos dados para serem processados depois. Esse processo muitas vezes está associado a somas bem determinadas, em que os valores possuem uma relação entre eles. Muitas vezes, ainda no contexto das Ciências Exatas e Engenharias é comum precisamos somar funções ou valores atribuidos a elas.

Em todos esses casos em que acumulamos valores ou funções existe uma regra na Matemática que define de forma precisa como efetuar essa soma e é o que chamamos de **somatório**.

Formalmente falando, somatório é um **operador matemático**[1] usado para efetuar somas de grande quantidade de termos.

Esse operador é representado pela letra grega Σ. Veremos em detalhes sobre somatórios mais a frente no nosso curso. Por hora vamos nos ater a entender como usar loops for ou while (ou ainda do-until (no Octave)) para efetuar somas bem definidas. Acompanhemos portanto os exemplos seguintes.

Exemplo 6.1 (Explicando em Detalhes de uma soma Simples)

Considere a soma dos termos dados abaixo:

$$S = 1 + 2 + 3 + 4 + \cdots + 29 + 30$$

Repare que essa soma está relacionada com a soma dos 30 primeiros números naturais. Repare ainda que podemos definir uma regra para

[1] Veremos a definição formal de operador mais a frente no nosso curso. Por hora, lembre-se que já falamos sobre operadores quando estudamos os conceitos básicos da linguagem de programação.

essa soma, regra essa associada aos elementos da soma, e que pode ser usada com um loop na programação.

Podemos dizer: efetue a soma dos n números, com n variando de 1 a 30. Isso pode ser representado usando a notação do operador somatório da seguinte forma:

$$S = \sum_{i=1}^{30} i$$

Essa equação deve ser lida assim: **efetue a soma da variável** i **com** i **variando de 1 até 30**.

Ou seja, fazemos primeiro $i = 1$ *e somamos (como é o primeiro termo S será igual a 1). Depois fazemos* $i = 2$ *somamos com o termo anteirior, que vale 1, o obtemos 3. Depois fazemos* $i = 3$ *e somamos com o termpo anterior e obtemos* 6 *e assim por diante. No final teremos obteido a soma:* $1 + 2 + 3 + 4 + \cdots + 28 + 29 + 30$.

É importante ver que essa soma pode ser feita com um loop for. Veja o trecho de código abaixo:

```
S=0;
for i=1:30
    S = S + i;
endfor
disp(S);
```

Repare que o o loop for é usado para variar i de 1 até 30, sempre somando o valor atual de S com o valor atual de i.

Importante entender essa lógica. Ela será muito usada de agora em diante no nosso curso!

Uma outra soma comum em computação, usada para treinarmos o raciocínio básico no trabalho com somatórios é a soma de números

ímpares ou pares. No exemplo seguinte temos um soma dos 10 primeiros números ímpares. Acomanhe a explicação.

Exemplo 6.2 (Soma com regra específica)
Efetue a soma:

$$S = 1 + 3 + 5 + \cdots + 17 + 19$$

Ess soma indica que estamos somando os números ímpares de 1 até 19. Ela pode ser representada, usando-se o operador somatório como:

$$S = \sum_{i=0}^{9} (2i + 1) \tag{2}$$

Repare bem nos limites de variação de i, explicitados no operador Σ*. Repare que i varia de i até* 9*. Vejamos:*

Se estamos somando o termo $2i + 1$*, teremos:*

$$\begin{aligned} 2i+1 &= 1 \ \textit{para}\ i = 0 \\ 2i+1 &= 3 \ \textit{para}\ i = 1 \\ 2i+1 &= 5 \ \textit{para}\ i = 2 \\ &\vdots \\ 2i+1 &= 19 \ \textit{para}\ i = 9 \end{aligned}$$

Logo, a equação (2) representa exatamente a soma dos 10 primeiros números ímpares, indo de 1 *até* 19*.*

Essa soma pode ser feita com o loop for, como mostra o trecho de programa abaixo:

```
S=0;
for i=0:9
     S = S + 2*i+1;
endfor
disp(S);
```

Exemplo 6.3 (Soma com 2 termos variando)
Um outro exemplo comum é somatórios é quando termos dois termos variando de forma diferente. Veja a seguinte soma:

$$S = \frac{1}{1} + \frac{3}{2} + \frac{5}{3} + \cdots + \frac{17}{9} + \frac{19}{10}$$

Repare que temos nessa soma o numerador variando de 2 em 2 e o denominador variando de 1 em 1. Nesse caso podemos escrever essa soma como:

$$S = \sum_{i=0}^{i=9} \frac{2i+1}{i+1} \tag{3}$$

Para verificar se de fato a equação (3) representa a soma acima, escreva-a variando i de 0 até 9.

É importante demais entender essas somas. Elas são usadas como exemplo para aplicarmos os loops for, while, do-until.

Um outro exemplo muito comum é quando temos somas com sinais variando. Estude com atenção o exemplo 6.4 dado a seguir.

Exemplo 6.4 (Soma com sinal variando)
Faça um script para calcular a soma dada na equação (4).

$$S = 0 - 1 + 2 - 3 + 4 - 5 + 6 \cdots - 19 + 20 \tag{4}$$

Solução: *Repare que nessa soma, todos os números ímpares são negativos e os pares são positivos. Podemos então escrevê-la da seguinte forma:*

$$S = \sum_{i=0}^{20} \left((-1)^i \; i \right)$$

Ou seja, usamos o termos $(-1)^i$ *para mudarmos o sinal, dependendo do valor de* i*. Para* i *par temos* $(-1)^i$ *igual a* 1*. E para* i *ímpar temos* $(-1)^i$ *igual a* -1*. Veja:*

$$\begin{aligned}
(-1)^i \; i &= 0 \;\; \textit{para} \;\; i = 0 \\
(-1)^i \; i &= -1 \;\; \textit{para} \;\; i = 1 \\
(-1)^i \; i &= 2 \;\; \textit{para} \;\; i = 2 \\
&\vdots \\
(-1)^i \; i &= -19 \;\; \textit{para} \;\; i = 19 \\
(-1)^i \; i &= 20 \;\; \textit{para} \;\; i = 20
\end{aligned}$$

Esse somatório pode ser escrito no loop for como:

```
S=0;
for i=0:20
   S = S + (-1)^i * i;
endfor
disp(S);
```

7 Exercícios de Fixação

1. Escreva scripts para executar as somas dadas nos exemplos 6.1 até 6.4. Use os trechos de códigos dados.

2. Repita os scripts do item anterior, agora usando while no lugar de for.

3. Repita o item 1, agora usando do-until no lugar de for.

4. Reescreva o código do exemplo 6.2 fazendo limite de i variar de 1 até 10 ao invés de 0 até 9.

7.1 Somatórios de Séries

Em Matemática temos a definição de um série numérica como uma sucessão de números reais. Dentre das aplicações de séries numéricas temos uma em que elas são usadas para definir números irracionais especiais, como o π.

Dentre das séries conhecidas para se obter o valor de π temos a dada na equação (5).

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \ldots \tag{5}$$

Repare que essa é uma soma **infinita**. Ela não termina nunca. Para obter o valor do π precisamente (ou melhor dizendo, o mais preciso possível), precisamos de acrescentar cada vez mais termos.

Se por exemplos usássemos apenas 3 termos teríamos:

$$\pi = 4 - \frac{4}{3} + \frac{4}{5}$$

ou seja

$$\pi = 4 - 1.3333 + 0.8 = 3.4667$$

que está longe do valor correto, que é $\pi = 3.1415$ (quando usamos 4 casas decimais). Claro que podemos ter a **precisão** que desejarmos.

Podemos por exemplo obter o π dado por:

$$\pi = 3.14159265358979;$$

mas para isso precisáriamos de somar muito mais termos.

O tipo de problema dado acima, que é o de obtermos um número especial a partir de uma soma, pode ser feito usando-se loops, entretanto o critério de parada da soma não é mais o último termo, pois só vamos parar de somar **quando encontrarmos um valor de π que acharmos adequado**. Ou seja, temos um critério de parada que pode ser pensando em portugues como:

```
EQUANTO não obter um valor de pi adequado, continue somando...
```

Portanto temos que definir primeiro o valor que queremos obter (pergunta: o que seria um valor de π adequado?...)

Uma forma de tratar esse problema é definindo uma variável que calcula o **erro entre o valor ideal** (ou desejado) e o **valor calculado** (obtido pelo somatório). Veja o fluxograma que representa esse cálculo, dado na figura 5.

O exemplo 7.1 apresenta um algoritmo para o cálculo do π usando o somatório dado na equação (5). Ele é a forma escrita usando a linguagem portugol para o fluxograma da figura 5.

Exemplo 7.1 (Algoritmo para Cálculo do π)
Para o cálculo do π aproximado podemos usar o algoritmo dado abaixo. Repare que calculamos o erro a cada iteração e ficamos dentro do loop **enquanto** *o erro for maior que 0.001.*

```
(considere pi como sendo o pi "real", isso é o dado pelo octave)
(esse algoritmo tem como objetivo calcular P com um valor próximo ao pi)
```

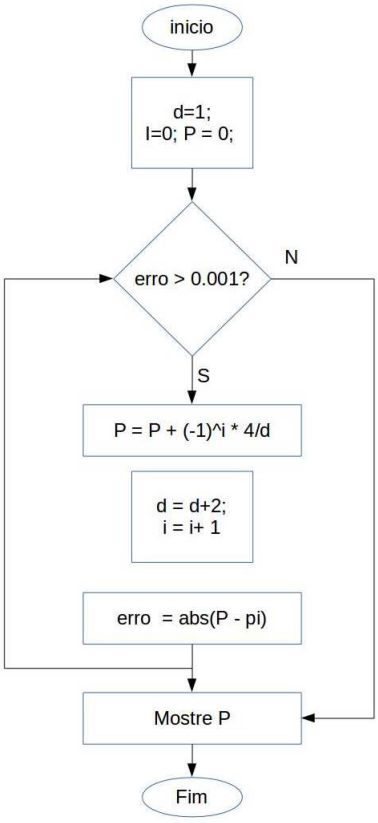

Figura 5: Fluxograma (usando parte da notação do Octave) para o cálculo do π a partir de uma iteração.

```
(a diferença entre o pi real e o pi aproximado (dado por P)
é o que chamamos de erro).

d=1;
i=0;
P=0;
faça
     P = P + (-1)^i * 4/d;
     d = d+2;
```

```
    i = i+1;
    erro =  abs(P - pi);
enquanto (erro > 0.001);
```

8 Exercícios de Fixação

1. Faça um script Octave para calcular o π aproximado usando o algoritmo do exemplo 7.1, usando, como dado no algoritmo a estrutura **do-until**. Ao final, saindo do loop, mostre o valor do π encontrado, o do π real (dado pelo Octave) e do erro obtido. Repare que a diferença (tolerância desejada) entre o π real e o calculado é de $0,001$.

2. Repita o item anterior agora usando a estrutura while.

3. Reescreva o script do item 1 agora lendo o valor desejado para a tolerância desejada.

4. Reescreva o script do item 2 agora lendo o valor desejado para a tolerância desejada.

5. Responda

   (a) Seria possível reescrever o script anterior agora usando a estruutra for? Se sim, qual seria o critério de parada do loop for? Ou seja, se o for é usado com uma faixa de valores, qual seria a faixa que você usuaria?

   (b) Pense qual estrutura de repetição seria mais adequada para esse tipo de problema. Compare esse tipo de problema com os vistos anteriormente em que usamos o loop para

efetuar somatórios. Qual a diferença entre eles? Quando seria melhor usar for, while e do-until?

6. Reescreva o script do item 1 usando o loop for, caso seja possível fazê-lo. Não sendo, justifique.

7. Reescreva o script do item 1, agora criando um vetor que guarda, para cada iteração o valor do erro. Ao final (quando sair do loop), mostre o erro final e plote o gráfico do vetor. Plote em uma figura usando o comando **plot()** e em outra figura usando o comando **semilogx()**. Interprete e entenda o resultado.

9 Loops para Tratar índices de Vetores e Matrizes

Na seçao 5 vimos que podemos usar os loops para acessar os valores dos vetores e das matrizes. Nessa seção vamos estudar um pouco mais detalhadamente essa questão.

Considere um vetor v, de dimensão $1 \times N$, cujos elementos são acessados por índices e são denominados de v_i. Dizemos, usando a notação Matemática que v_i é o i-**ésimo** elemento de v. Por exemplo, v_1 é o primeiro elemento de v, v_2 o segundo e assim por diante, até o último elemento que seria o de índice N, ou seja v_N. Esse vetor seria, genericamente representado como:

$$v = \begin{bmatrix} v_1 & v_2 & v_3 & \dots & v_{N-1} & v_N \end{bmatrix}$$

Se, por exemplo, tivermos $N = 4$, o seguinte vertor

$$v = \begin{bmatrix} -1 & 5 & 9 & 0 \end{bmatrix} \tag{6}$$

teria os elementos:

```
v[1]:=-1; v[2]:=5;v[3]=9;v[4]=N
```

Dizemos, numa notação simplificada que os elementos de v são dados por v_i, $i = 1, 2..N$, que no caso é $N = 4$.

Para trabalharmos com vetores e matrizes precisamos acessar os elementos a partir dos seus índices. No trecho de código abaixo usamos o loop for para acessar os elementos do vetor dado na equação (6).

```
v=[-1,5,9,0];
for i=1:4
   printf("v[%d] = %d\n",i,v(i)");
endfor
```

Quando queremos que o usuário informe os dados de um vetor, podemos perguntar inicialmente qual o tamanho do vetor. Na linguagem C veremos que temos que saber, de antemão o tamanho do vetor. No caso do Octave o vetor vai aumentando na medida que vamos colocando dados nele.

O trecho de código é um exemplo de preenchimento de dados de um vetor com números de matrícula de uma turma.

```
N = input("quantos alunos a turma possui?");
for i=1:N
  n_mat(i) = input(["Entre com o número de matrícula do aluno ",num2str(i
end
```

Nesse trecho o loop for faz com que a entrada seja realizada N vezes (ou seja, são feitas N leituras). Cada leitura é guardada na posição i. A variável i é incrementada a cada passo do loop for.

Observação:

Repare na forma que usamos o input: criamos um vetor com dois valores. O primeiro valor é uma string *Entre com o número de matrícula do aluno número* e o segundo valor é a string gerada pelo conversão do número i para string (no caso, para caractere) a partir da função num2str().

9.1 Loops para Matrizes

Quando trabalhamos com vetores, seja para ler dados e salvar neles seja para ler os dados e apresentar na tela, usamos um loop for, que tem por objetivo percorrer o vetor, variando o índice.

Por outro lado, quando trabalhamos com matrizes bidimensionais temos que percorrer as linhas e as colunas, portanto precisamos de dois loops: um para as linhas e outro para as colunas. Em geral, fazemos um loop externo para percorrer as linhas e um interno para as colunas. Acompanhe o exemplo a seguir.

Exemplo 9.1 *Faça um script para preencher uma matriz* 3 *times*4 *com números aleatórios entre 1 e 20.*

O trecho a seguir resolve essa questão:

```
  for i=1:3       % varia 3 linhas
    for j=1:4   % varia 4 colunas
        M(i,j) = int32(1 + 20*rand());
     end
end
```

Ao terminar esse trecho o programa pode ter um outro trecho para mostrar na tela o resultado. Podemos mostrar elemento por elemento:

```
  for i=1:3       % varia 3 linhas
    for j=1:4   % varia 4 colunas
         printf("M[%i,%i] = %i\n",i,j,M(i,j));
     end
end
```

Ou podemos mostrar no formato matricial (linhas e colunas):

```
   for i=1:3       % varia 3 linhas
    for j=1:4   % varia 4 colunas
        printf("\t %i",M(i,j));
     end
     printf("\n");
end
```

Repare que só damos o comando de nova linha após terminarmos a impressão de uma linha inteiro. O \ t é usado para tabular as saídas.

10 Exercícios de Fixação

1. Faça scripts completos com os trechos de programas apresentados na seção 9.

2. Faça um script para criar duas matrizes $4\times$ de inteiros entre 1 e 100 e apresentar: as matrizes, a soma, os produtos, as divisões entre as duas matrizes criadas.

3. Faça um script para criar um vetor v de tamanho N, informado pelo usuário, v sendo preenchido segundo a regra $v(i) = i + 1$. O programa deve mostrar o v.