3 A Iteratividade

3.1 Automatizando as Células do Excel

Até o momento a única forma de mostrar ao usuário o resultado da programação era o uso do *MsgBox*. Essa caixa de saída é uma forma muito próxima do que faziam os programas escritos em linguagens antes das orientadas à objeto. No VBA em questão, o objetivo principal é trabalhar direto nas células que estão armazenando dados. Empresas nos dias atuais possuem dados arquivados em planilhas Excel e não são poucos. Com a internet, aquisição em "*real time*" se tornou muito fácil e assim as planilhas lotam facilmente durante algumas horas de serviço. A grande novidade do Excel foi seu poder de não somente trabalhar com as células como um banco de dados e com operações simples pré-definidas, mas possibilitar que programas que antes precisavam de linguagens e entradas e saídas de dados específicas pudessem ser automatizadas com os conhecimentos já existentes em termos de programação. Assim, usando o VBA é possível "conversar" com as células e ordenar tarefas tais como fazer aquisição de dados, fazer gráfico de maneira automática, copiar, colar, e programas mais complexos tais como testes estatísticos, simulações, jogos, etc.

Para programar uma planilha e interagir com as células a função chave dentro do VBA é *Cells*. O leitor deve enxergar na planilha como se fosse uma grande matriz, onde as colunas A, B, C, etc. estarão numeradas no VBA como 1, 2, 3, etc. e as linhas da mesma forma. Assim, para copiar o valor de uma célula do Excel para uma variável do VBA o comando é:

Variável = Cells(linha, coluna)

Onde linha e coluna são os números associados a linha e coluna do Excel. Por exemplo a célula A1 dentro do VBA será *cells(1,1)*, a célula B4 será *cells(4,2)* e assim por diante. Suponha que se deseja alimentar uma variável x do VBA com o valor que está na célula A1. Para isso basta dizer que a variável recebe o valor de *cells(1,1)* conforme a programação a seguir.

	A	B
1	2,5	4,1
2	1,7	1,1

```
Sub programa_celulas()
Dim x As Single

x = Cells(1, 1)

End Sub
```

Se for desejado o contrário, ou seja, colocar os cálculos realizados no VBA dentro das células do excel o processo é inverso e para isso basta dizer que *cells(2,3)* recebe x, o que indicará ao Excel que a célula C2 deverá receber tal valor.

Exemplo 3.1

Fazer um programa que calcule a média dos valores armazenados nas células A1, A2, A3 e A4 do Excel a seguir e coloque o valor na célula A5.

	A
1	2,5
2	1,7
3	2,2
4	4

```
Sub media_celulas()
Dim media As Single

 media = (Cells(1, 1) + Cells(2, 1) + Cells(3, 1) + Cells(4, 1)) / 4
 Cells(5, 1) = media

End Sub
```

O leitor deve perceber duas noções importantes nesse exemplo. A primeira é que o nome do programa não deve ser o mesmo da variável. Se criar um programa com o mesmo nome da variável o compilador apresentará mensagem de erro em alguns casos futuros. Nesse exemplo simples o compilador não apresenta erro e deixar passar, mas em exemplos mais complexos ele proíbe o uso de nome da variável. Deve ser observado que o nome do programa foi “media_células” (sem acento) enquanto o nome da variável foi “media” (sem acento). A segunda noção importante é quanto ao parêntesis. O parêntesis da programação é fundamental, uma vez que todos os termos do divisor devem estar sob o mesmo parêntesis. Isso indica ao computador que todos os termos devem ser computados e o resultado dividido por quatro. Se o parêntesis mais externo for retirado, o computador vai entender que se deseja dividir apenas a célula A4 por quatro e não todas, o que deixa o programa errado. Esse erro seria um erro de lógica e não um erro de linguagem, o que não seria detectado pelo compilador.

Exemplo 3.2

Dispor os dados da Acesita-SA da forma como está representada abaixo em sua planilha e fazer um programa em VBA para dar o somatório dos cinco valores da ação.

	A
1	
2	Acesita-Preço
3	0,59
4	0,61
5	0,6
6	0,61
7	0,58

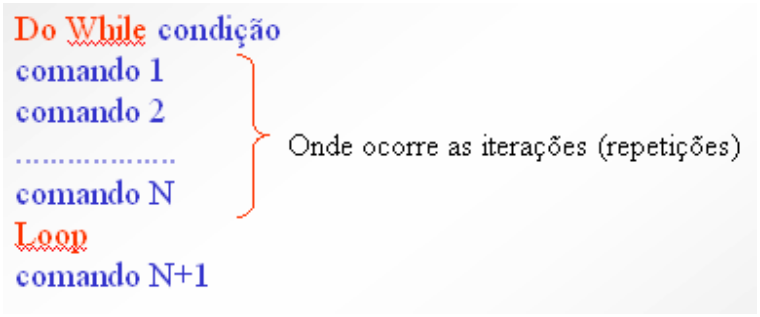
A solução é a programação a seguir onde o resultado será apresentado na célula A8 ou *cells(8,1)* no VBA.

```
Sub programa_soma()
Dim soma As Single

 soma = Cells(3, 1) + Cells(4, 1) + Cells(5, 1) + Cells(6, 1) + Cells(7, 1)
 Cells(8, 1) = soma

End Sub
```

O leitor deve estar perguntando "mas se for muitas linhas?". Claro, para um número acima de três linhas não tem sentido esse tipo de programação. Mas uma vez entendida a relação entre as células e o VBA o leitor está preparado para a noção de iteratividade.

3.2 O comando Do While

A palavra é iteração e não interação. A iteratividade significa repetição e a interatividade o modo como se estabelece uma relação entre pessoas, equipamentos, softwares, etc. A iteração é a peça principal da programação, é o que faz a automação funcionar e dar a falsa impressão as pessoas que o computador "pensa" e rápido para realizar suas tarefas. É na iteração que os cálculos mais complexos se realizam e as atividades mais complexas executadas de maneira rápida e automática. Em todas as linguagens de programação existem tipos de comandos ou funções para exercer essa atividade de iteração. O primeiro comando de repetição no VBA é o comando ***Do While*** cuja estrutura é a seguinte.



Entre o comando *Do While* e o comando *Loop*, tudo o que estiver dentro deverá ser executado repetidamente quantas vezes for necessária até o atendimento da condição lógica estabelecida na frente de *Do While*. Essa condição lógica ou condição de parada da repetição deve ser colocada de forma adequada. O uso do comando *Do While* de forma errada pode ocasionar um *loop* infinito no programa que precisará ser abortado para poder ser interrompido. As condições de parada são as mesmas condições lógicas para a função *if* ou seja,

<	menor que
<=	menor ou igual a
>	maior que
>=	maior ou igual a
<>	diferente

Exemplo 3.3

Fazer um programa para escrever 10 vezes a palavra "teste".

Para resolver esse problema o programador deverá introduzir a figura do "contador". É ele que dirá ao *Do While* quantas vezes o cursor deverá ir até o comando *loop* e voltar. O algoritmo básico desse programa será:

(1) Ler o número de repetições desejado pelo usuário (no caso 10).
(2) O contador começa com 1.
(3) Enquanto o contador não for igual ou superior a 10, continue imprimindo a palavra "teste".
(4) Acrescente o contador do valor anterior mais um.
(5) Se contador ultrapassou a 10 o programa pára.

O passo (4) do algoritmo é muito importante. Isso porque em programação como já mencionado antes não existe igualdade de valores, mas sim posições ocupadas (bit ocupado). Então para aumentar um contador de uma unidade, o programador deve informar ao computador que o valor que ele possui no passado em sua memória deve ser apagado e substituído pelo que ele tinha somado de um, ou seja,

$$cont \leftarrow cont + 1$$

O programa nesse caso será:

```
Sub frase_repetida()
Dim i As Integer
Dim n As Integer

n = 10
i = 1
Do While i <= n
  MsgBox ("teste")
  i = i + 1
Loop

End Sub
```

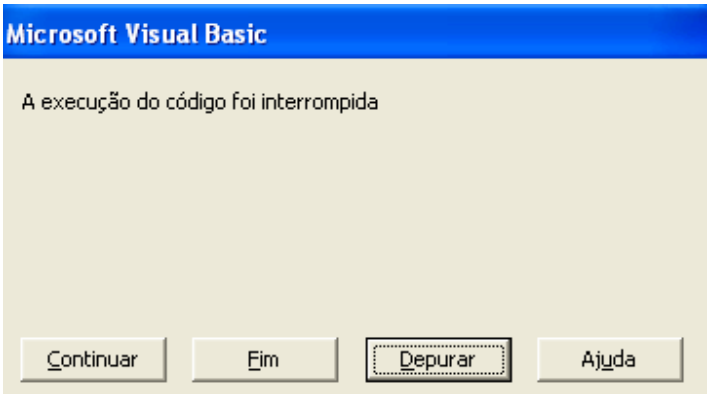
Exemplo 3.3

Fazer um programa para escrever 10 vezes a palavra "teste" nas células do Excel onde a coluna A deve mostrar o número da repetição e a coluna B a palavra repetida.

	A	B
1	1	teste
2	2	teste
3	3	teste
4	4	teste
5	5	teste
6	6	teste
7	7	teste
8	8	teste
9	9	teste
10	10	teste

```
Sub frase_repetida()
Dim i As Integer
Dim n As Integer

n = 10
i = 1
Do While i <= n
  Cells(i, 1) = i
  Cells(i, 2) = "teste"
  i = i + 1
Loop

End Sub
```

O que acontece se tanto no exemplo com MsgBox ou nas células o programador esquecer da linha do contador i = i +1? O programa entra em loop infinito e apenas um *ctrl+break* consegue interrompê-lo.



Esse fato ocorre porque como não foi alimentado o contador "i", o valor dele será sempre 1 e quando o cursor alcança a palavra *loop* é obrigado a voltar até onde está o *Do While*. Na verificação da pergunta "se i < 10" como i será sempre 1, será sempre menor que 10 o que obriga o computador infinitamente a repetição a menos que seja interrompido pelo comando *break*.

Outro tipo de *loop* infinito é quando o programador esquece e acaba invertendo os papéis do contador e do valor de parada "n". Observe o programa a seguir.

```
Sub frase_repetida()
Dim i As Integer
Dim n As Integer

n = 1
i = 10
Do While i >= n
Cells(i, 1) = i
Cells(i, 2) = "teste"
i = i + 1
Loop

End Sub
```

Nesse erro o programador trocou *n* por *i* e assim infinitamente o computador imprime a palavra “teste” nas células. Por sorte como *i* foi definido como inteiro seu valor tem um limite pequeno de memória e a última célula que recebe a palavra está na linha 32767. Mas se for definida a variável *i* como número *single* ou *double* o programa só pára na última linha do Excel (65.536).

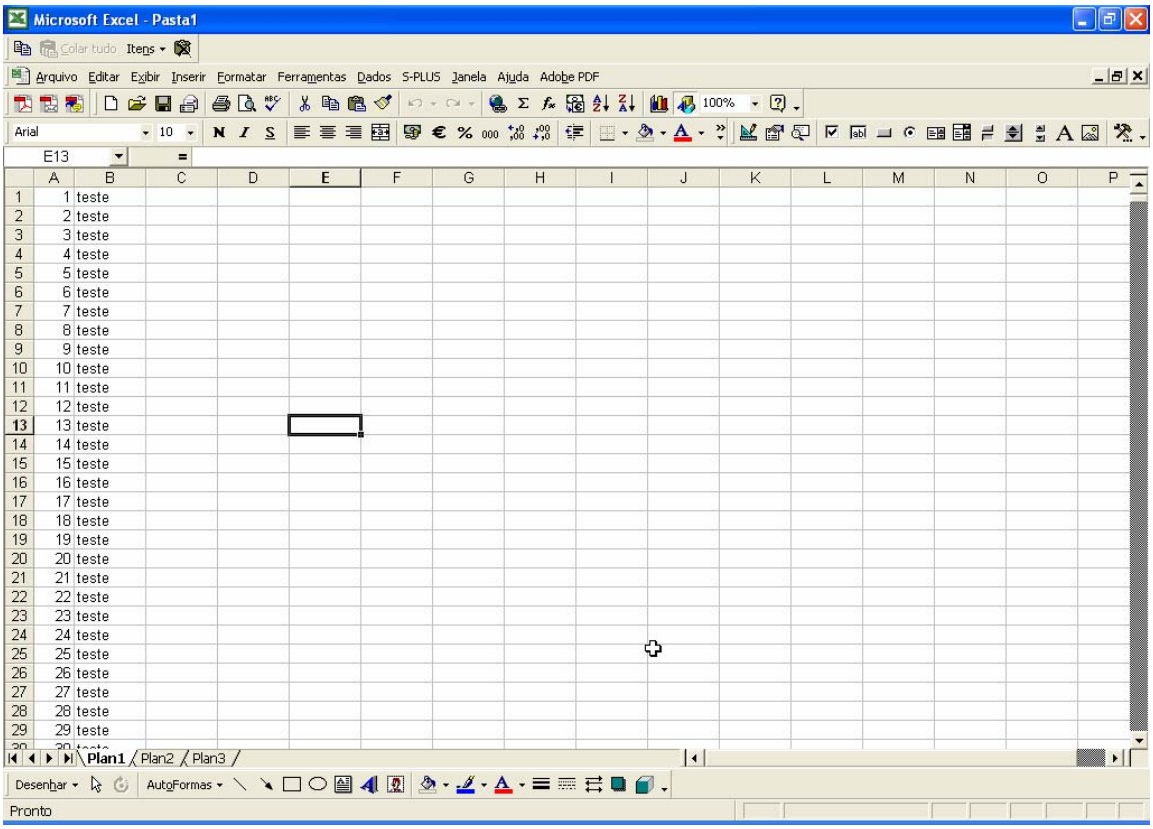


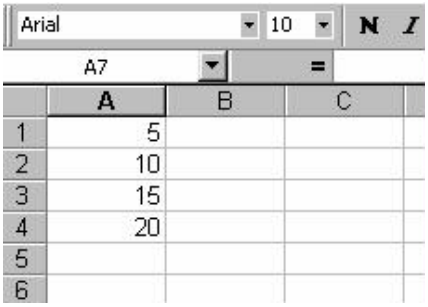
Figura 3.1 – *Loop* infinito por erro de programação do comando *Do While*

O cálculo de algumas leis matemáticas é um excelente exercício de raciocínio lógico para a programação de computadores. Por exemplo, regras matemáticas, geração de números, sempre preparam melhor um programador quando estiver diante de problemas mais complexos. O exemplo 3.4 sugere uma forma de geração de múltiplos do número 5.

Exemplo 3.4

Fazer um programa que calcule e imprima nas células do Excel todos os múltiplos de 5 até 20 inclusive.

```
Sub mult5()
Dim i As Integer
Dim mult As Double
i = 1
Do While mult < 20
mult = 5 * i
Cells(i, 1) = mult
i = i + 1
Loop

End Sub
```



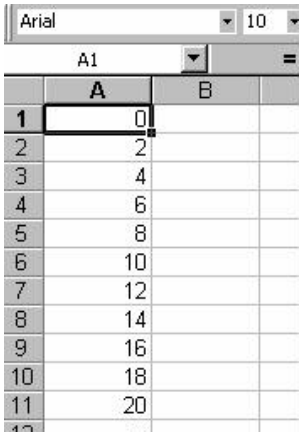
Nesse caso os múltiplos são sempre gerados pelo próprio contador multiplicado pelo número 5. Observando a tabela pode-se notar como a regra do contador gera os números.

Contador	*Número gerado*
1	5***1**
2	5***2**
3	5***3**
4	5***4**

Exemplo 3.4

Fazer um algoritmo que gere e imprima na forma de linhas da planilha os primeiros 11 números pares começando pelo zero.

```
Sub pares()
Dim i As Integer
Dim soma As Double
soma = 0
i = 1
Do While soma <= 20
Cells(i, 1) = soma
soma = soma + 2
i = i + 1
Loop

End Sub
```



A forma de geração dos pares do exemplo 3.4 poderia ser usada para a geração dos múltiplos de 5 do exemplo 3.3. A idéia básica ainda é a mesma, ou seja, uma condição de parada que no caso foi o número par 20 pois de antemão o programador sabia que ele era o

décimo primeiro par depois do zero. Mas uma maneira melhor de programar é evitando números predefinidos ou conhecidos pelo programador, pois se o usuário mudar de idéia o programa ainda deve continuar funcionando. Assim uma melhor programação seria:

```
Sub numero_par()
Dim i As Integer
Dim n As Integer
Dim soma As Integer

n = 11
i = 1
soma = 0
Do While i <= n
Cells(i, 1) = i
Cells(i, 2) = soma
soma = soma + 2
i = i + 1
Loop

End Sub
```

	A	B
1	1	0
2	2	2
3	3	4
4	4	6
5	5	8
6	6	10
7	7	12
8	8	14
9	9	16
10	10	18
11	11	20

Essa forma de programação é melhor que a anterior pois se o usuário não quiser mais apenas 11 números pares mas 50 números, por exemplo, então o programador não precisaria alterar nada. Na verdade, para ficar o mais genérico possível seria ainda melhor que o programa lesse o valor da quantidade de números desejada para ser gerada. Assim, ao invés de n = 11 o mais correto seria

$$\mathbf{n} = \textit{Cint}\ (\ \textit{InputBox}\ (\ \text{“ n = “}\ )\ )$$

Outro ponto a ressaltar é que nesse algoritmo existem na verdade dois contadores que agem de forma diferente. O contador da variável *i* marca a quantidade de números gerados e a variável *soma* gera os pares subseqüentes.

i	Soma(passada)	Soma(futura)
1	**0**	2
2	**2**	4
3	**4**	6
4	**6**	8
…	…	…

Quando o valor de *i* é 1 a soma que começa é zero e depois de impressa nas células ela recebe o valor somado ao seu antigo valor, por isso $soma \leftarrow soma + 2$. Quando a variável *i* aumenta para 2 está indicando que esse novo valor 2 recebido por soma é um valor futuro que deve ser computado quando o cursor retornar na linha do *Do While*.

Exemplo 3.5

Faça um programa para imprimir nas células do Excel a soma de n números.

Esse tipo de programa é muito parecido com a programação existente dentro da função *autosoma* do excel representada pelo botão de somatório Σ.

Novamente o programador precisa de dois contadores, um para marcar quantos pontos estão sendo contados dentro do *loop* e o contador que fará a soma dos números. A diferença em relação ao exemplo 3.4 é que agora o número não é fixo como o dois dos números pares. Aqui os *n* números estão na planilha.

	A
1	5
2	4
3	3
4	2
5	5
6	1

```
Sub numero_soma()
Dim i As Integer
Dim n As Integer
Dim soma As Integer

n = CInt(InputBox("entre com o total de números ="))
i = 1
soma = 0
Do While i <= n
soma = soma + Cells(i, 1)
i = i + 1
Loop

Cells(n + 1, 1) = soma

End Sub
```

O programa lê a quantidade de números inseridas nas células da coluna A e então começa a fazer a soma dos números. A soma inicial é zero, e quando entra no *Do While* a soma passada desaparece e em seu lugar é salvo o valor antigo mais o valor da célula. Quando o cursor voltar a esse ponto, o valor da soma desaparece e em seu lugar é somado o valor antigo do passo anterior com o valor novo da célula e assim até o contador *i* "estourar" o valor final *n.*

No último comando, como o programa já conhece a quantidade de dados *n* ele ordena que o computador coloque depois do último dado o valor da soma, por isso o comando de linha *Cells (n+1 , 1) = soma*.

O resultado final será:

	A
1	5
2	4
3	3
4	2
5	5
6	1
7	**20**

←última linha com o valor da soma

Exemplo 3.6

Fazer um programa que tome os valores da coluna A do Excel, multiplique termo a termo com os elementos da coluna B salvando os resultados na coluna C. No final o programa deverá somar os valores da coluna C e mostrar a soma após a última célula.

	A	B	C
1	5	2	
2	4	1	
3	3	3	
4	2	2	
5	5	1	
6	1	2	

Nesse exemplo o procedimento para a resolução é o seguinte:

(1) Leia a quantidade de números.
(2) Coloque no *Do While* o produto de cada célula de A por B em C.
(3) Crie uma variável soma que some cada novo valor da variável C.
(4) Imprima após a última célula de C o valor da soma total.

```
Sub numero_soma()
Dim i As Integer
Dim n As Integer
Dim soma As Integer

n = CInt(InputBox("entre com o total de números ="))
i = 1
soma = 0
Do While i <= n
Cells(i, 3) = Cells(i, 1) * Cells(i, 2)
soma = soma + Cells(i, 3)
i = i + 1
Loop

Cells(n + 1, 3) = soma

End Sub
```

O leitor deve observar que apesar de saber quantos elementos têm em cada coluna, o programa fica mais geral quando se pergunta o valor de *n*. Se a planilha alterar a quantidade de números o programa não precisa ser alterado. A coluna C está representada pelo comando de linha *Cells (i , 3) = Cells (i , 1) * Cells (i , 2)*. Para cada linha *i* alterada pelo *loop* uma nova célula na mesma linha e coluna C é gerada pelo produto de A e B.

Ao mesmo tempo o programa repete as lógicas das somas dos exemplos anteriores e vai somando esses termos novos para cada nova linha *i*. Depois que o contador *i* chegou ao fim, o valor final da soma é impresso na linha após o último dado representado pelo comando de linha *Cells (n + 1, 3) = soma*.

3.3 Seqüências Matemáticas

Como já mencionado antes, a melhor maneira de aprender a lógica de programação é se estudando as regras matemáticas e operações algébricas, tentando automatizá-las. Quando se consegue isso, os outros tipos de programação do dia a dia se tornam muito mais fáceis para os programadores iniciantes. Seqüências de números e séries numéricas sempre foram desafiadoras para o entendimento matemático sobre a noção de limite e a definição da entidade definida como infinito.

A seqüências possuem uma representação própria quando se conhece a regra que gera os números que a compõem. Assim, por exemplo, algumas representações podem ser formuladas como nos casos a seguir, tomando-se um termo genérico da seqüência como a_n:

(a) $1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4} \cdots$ tem representação $a_n = \frac{1}{n}$

(b) $\frac{1}{3}, \frac{2}{5}, \frac{3}{7}, \frac{4}{9}, \cdots$ tem representação $a_n = \frac{n}{2n+1}$

(c) $1, -\frac{1}{2}, \frac{1}{3}, -\frac{1}{4}, \cdots$ tem representação $a_n = \frac{(-1)^{n+1}}{n}$

(d) $1, \sqrt{2}, \sqrt{3}, \sqrt{4}, \cdots$ tem representação $a_n = \sqrt{n}$

Essas fascinantes representações matemáticas sempre intrigaram sobre a possibilidade da existência de um limite e se esse limite é finito ou infinito. A definição de limite tem uma formulação matemática rigorosa e encontrada nos livros de cálculo como a seguir.

Definição de Limite de Seqüências

Uma seqüência $\{a_n\}$ ***converge*** *para o número L se para todo número positivo ε existe um número inteiro N > 0 tal que para todo n*

$$n > N \Rightarrow |a_n - L| < \in$$

Se esse número L não existe, diz-se que a seqüência $\{a_n\}$ ***diverge****.*

A definição matemática anterior afirma que uma vez escolhida uma região de raio ε em torno de um número L, existirá dentro da seqüência um termo inicial a partir do qual todos os outros nunca ultrapassarão a distância ε em torno desses termos. Se esse termo cujo índice é N não puder ser encontrado, afirma-se então que a seqüência não tem limite e cresce indefinidamente.

O leitor pode estar se perguntando qual a utilidade de tais definições dentro da computação. Essas definições permitiram que diversas áreas criassem tabelas padrão para seus cálculos tais como física, química, matemática, engenharia, administração e economia. Mais especificamente em economia e finanças seqüências específicas permitiram a criação de tabelas por exemplo para depreciação de ativos usando a seqüência de Fibonacci.

Essas seqüências podem ajudar ainda a formarem séries infinitas as quais podem representar funções, integrais, derivadas, números específicos tais como π e o número de Euler *e*. A mais famosa história é a respeito de Gauss que com oito anos de idade, de castigo em sala de aula acabou criando a regra para soma de série infinita com progressão geométrica (PG). Assim como seqüências também o estudo de convergência para séries infinitas são muito importante para todas as áreas de estudo relacionadas com computação.

Uma série bastante conhecida é a série *harmônica* que é representada pela soma dos termos

$$S = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots$$

onde S pode ser escrito na representação matemática

$$\sum_{n=1}^{\infty} \frac{1}{n} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots$$

Essa série é utilizada por exemplo em estudos de harmonia musical, em engenharia, matemática e pode-se provar, ao contrário do que o leitor pode pensar, que essa série tem limite infinito. Ou seja, essa série cresce indiscriminadamente e não converge para nenhum número de termo final *n*. Se um programa computacional fosse feito para encontrar o valor dessa série dependendo no número de termos colocados pelo usuário ocorreria o chamado *overflow* ou *estouro de memória*.

Algumas séries se tornaram tão especiais que receberam nomes e hoje são usadas por milhares de pessoas todos os dias com um único aperto de tecla de uma calculadora de bolsa, tais como seno, cosseno, raiz quadrada, exponencial entre outras. As representações de algumas séries especiais são apresentadas a seguir:

(a) $\text{sen}(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots + (-1)^n \frac{x^{2n\pm1}}{(2n+1)!} \pm \cdots$

(b) $\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots + (-1)^n \frac{x^{2n}}{(2n)!} \pm \cdots$

(c) $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} + \cdots$

(d) $\ln(x) = 2\left[\frac{x-1}{x+1} + \frac{(x-1)^3}{3(x+1)^3} + \frac{(x-1)^5}{5(x+1)^5} + \cdots + \frac{(x-1)^{2n+1}}{(2n+1)(x+1)^{2n+1}} + \cdots\right]$ para valores $x > 0$

(e) $\text{senh}(x) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \cdots \frac{x^{2n+1}}{(2n+1)!} + \cdots$

O leitor deve ter percebido que algumas representações dos termos a_n dessas séries são bastante complexos. É nesse aspecto que a computação com a idéia de iteratividade veio agilizar todo a forma de raciocínio científico com rapidez e segurança. O programador não precisa conhecer o termo de convergência a_n de uma determinada série. O programador precisará apenas de observar qual a regra lógica de iteratividade existente na série e com o

uso do comando *Do While* por exemplo gerar a série até o número de termos desejado pelo usuário.

Exemplo 3.7

Fazer uma programação macro para gerar e imprimir nas células da planilha do Excel a seqüência de termos abaixo.

	A	B	C	D	E	F	G	H	I	J
1	1	2	3	4	5	6	7	8	9	10
2										

Essa seqüência tem a representação matemática $a_i = i$, ou ainda, o i-ésimo argumento da seqüência é o próprio valor da seqüência. Então o algoritmo deverá perguntar ao usuário quantos termos deseja gerar e colocá-los nas células.

```
Sub seq1()
Dim i As Integer
Dim n As Integer

n = CInt(InputBox("numero de termos desejado = "))

i = 1

Do While i <= n
     Cells(1, i) = i
     i = i + 1
Loop

End Sub
```

Exemplo 3.8

Fazer um algoritmo e programe uma macro para gerar a seqüência harmônica nas células do Excel.

$$\left\{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \cdots, \frac{1}{n}\right\}$$

Nesse caso o algoritmo é muito parecido como exemplo anterior, não devendo o leitor confundir o termo *n* da representação matemática com o número de termos que os programas normalmente pedem. Por exemplo, se o usuário desejar 4 termos, a programação deverá ser o inverso do contador, até o *loop* parar no valor 4.

Para 10 termos o resultado é

	A	B	C	D	E	F	G	H	I	J
1	1	0,5	0,333333	0,25	0,2	0,166667	0,142857	0,125	0,111111	0,1
2										

e a programação da macro é a que se segue.

```
Sub seq1()
Dim i As Integer
Dim n As Integer

n = CInt(InputBox("numero de termos desejado = "))

i = 1

Do While i <= n
     Cells(1, i) = 1 / i
     i = i + 1
Loop

End Sub
```

Exemplo 3.8

Fazer uma macro para a seqüência abaixo e imprimir o resultado nas células da planilha em Excel.

$$\left\{\sqrt{1},\sqrt{2},\sqrt{3},\cdots,\sqrt{n}\right\}$$

A programação é muito parecida com a dos outros exemplos, mas com o cuidado de usar corretamente a função para extrair a raiz quadrada dos números, que no caso serão os contadores da seqüência. A função a ser utilizada é *Sqr()* conforme solução a seguir.

```
Sub seq_raiz()
Dim i As Integer
Dim n As Integer

n = CInt(InputBox("numero de termos desejado = "))

i = 1

Do While i <= n
     Cells(1, i) = Sqr(i)
     i = i + 1
Loop

End Sub
```

Cujo resultado para n = 10 é

	A	B	C	D	E	F	G	H	I	J
1	1	1,414214	1,732051	2	2,236068	2,44949	2,645751	2,828427	3	3,162278
2										

Apesar de muito parecida a programação de séries infinitas deve ser realizada com muito cuidado, pois a regra da soma deve ser sempre lembrada na hora da programação. Deve-se sempre lembrar que a igualdade representa na verdade que uma posição na memória será sempre a mesma e irá ser apagada para receber um novo valor a cada iteração.

Para fazer a programação da série $S = \sum_{i=1}^{n} i$, por exemplo, cada novo termo calculado deve-se trocar o valor da nova soma.

Soma ← Soma + *novo_valor*

A programação da série é a seguinte:

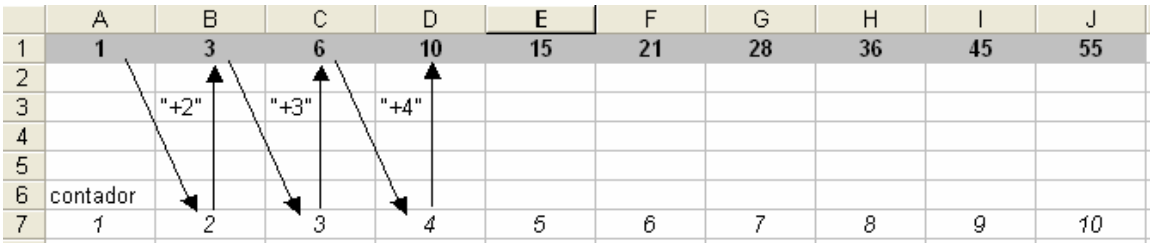
```
Sub serie_soma()
Dim i As Integer
Dim n As Integer
Dim soma As Integer
Dim novo_termo As Integer

n = CInt(InputBox("numero de termos desejado = "))

i = 1
soma = 0

Do While i <= n
     novo_termo = i
     soma = soma + novo_termo
     Cells(i, 1) = soma
     i = i + 1
Loop

End Sub
```

Usando n = 10 termos,

	A	B	C	D	E	F	G	H	I	J
1	1	3	6	10	15	21	28	36	45	55

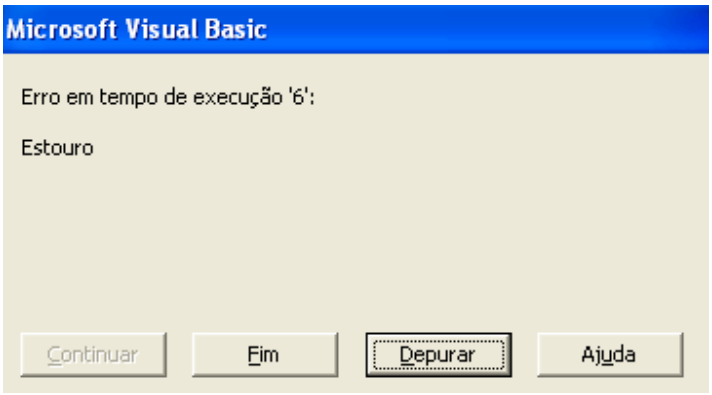
Como se chega nesses valores? O novo termo é sempre o contador que é sempre somado de um dentro do *Do While*. Então a soma anterior se apaga e no seu lugar entra o valor antigo da memória mais o valor atual do contador. Então no primeiro passo a soma é nula, mas antes do *loop* a soma recebe o novo termo que no caso é o valor 1 do contador. Como o valor antigo da soma era zero, então seu novo valor será 0 + 1. Quando o cursor chega no *loop* o contador para o próximo passo já vale 2 e então para a próxima soma, o

valor será apagado da memória e em seu lugar entrar 1 + 2, conforme os passos mostrados a seguir.

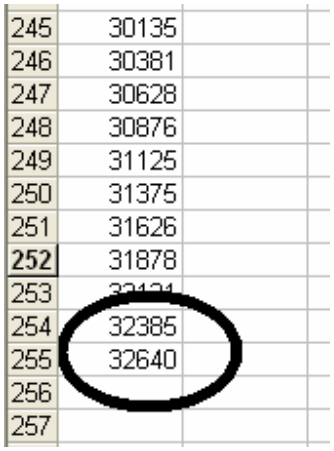


	A	B	C	D	E	F	G	H	I	J
1	**1**	**3**	**6**	**10**	**15**	**21**	**28**	**36**	**45**	**55**
2										
3										
4										
5										
6	contador									
7	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*

Mas o leitor deve ter cuidado. Imagine que se deseja somar 20.000 termos da série. Primeiro que ao invés de colunas, os resultados deverão ser salvos em linhas, trocando a formulação para *Cells (i ,1)*. Em segundo lugar, ao rodar o programa com n = 20.000 o resultado é o apresentado a seguir:



Ocorre um estouro de memória porque a definição das variáveis no *Dim* foi que a soma era inteira (*integer*), mas o valor de uma variável inteira é limitado a cerca de 33.000. Ao observar a planilha abaixo observa-se que o programa pára na linha 255, limite da variável inteira com a soma valendo 32.640.



245	30135
246	30381
247	30628
248	30876
249	31125
250	31375
251	31626
252	31878
253	
254	32385
255	32640
256	
257	

Por isso a definição correta da variável é importante e o programa por exemplo poderia ser alterado colocando-se por exemplo formato do tipo *Long*, que conforme definição do auto-ajuda do VBA tem valor variável entre –2.147.483.648 e + 2.147.483.647.

Tipo de dados Long

Consulte também Exemplo Informações específicas

As variáveis Long (inteiro longo) são armazenadas como números de 32 bits (4 bytes), sinalizados, no intervalo de -2.147.483.648 a 2.147.483.647. O caractere de declaração de tipo para **Long** é o 'E' comercial (**&**).

Outra solução poderia ser dimensionar a variável soma como *single* que tem o intervalo definido abaixo dentro do VBA.

Tipo de dados Single

Consulte também Exemplo Informações específicas

As variáveis Single (vírgula flutuante de precisão simples) são armazenadas como números IEEE de vírgula flutuante de 32 bits (4 bytes), com valor no intervalo de -3,402823E38 a -1,401298E-45 para valores negativos e de 1,401298E-45 a 3,402823E38 para valores positivos. O caractere de declaração de tipo para **Single** é o ponto de exclamação (**!**).

Então, nesse exemplo com 20 mil linhas, ou n = 20.000 termos o resultado será

$$\sum_{i=1}^{20000} \frac{1}{i} = 2 \times 10^8$$

	A
19996	2E+08
19997	2E+08
19998	2E+08
19999	2E+08
20000	2E+08

com a dimensão das variáveis adotadas como a seguir

```
Sub serie_soma()
Dim i As Integer
Dim n As Integer
Dim soma As Long
Dim novo_termo As Long
```

Exemplo 3.9

Fazer uma macro para gerar a soma de n termos da série *harmônica* conforme representação a seguir.

$$S = \sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} + \cdots$$

O programa não tem muitas diferenças em relação ao programa comentado anteriormente. Novamente a variável de soma e novos termos não podem ser inteiros, e nesse caso, nem mesmo inteiro longo(*Long*) pois tem-se frações envolvidas na série.

```
Sub serie_harmonica()
Dim i As Integer
Dim n As Integer
Dim soma As Single
Dim novo_termo As Single

n = CInt(InputBox("numero de termos desejado = "))

i = 1
soma = 0

Do While i <= n
     novo_termo = 1 / i
     soma = soma + novo_termo
     Cells(i, 1) = soma
     i = i + 1
Loop

End Sub
```

Para n = 20 termos o resultado será o apresentado a seguir. O leitor poderá perceber que intuitivamente compreende-se o comentário feito sobre convergência. Ao observar apenas os números na representação da série, permite uma falsa impressão de que os números subseqüentes estão diminuindo e portanto, a série deve convergir para algum número. Não é o que acontece como apresentado na planilha.

Exemplo 3.10

Fazer uma macro para gerar a soma de n termos da série conforme representação a seguir.

$$S = \sum_{i=1}^{n} \frac{1}{i^2} = 1 + \frac{1}{4} + \frac{1}{9} + \cdots + \frac{1}{n^2} + \cdots$$

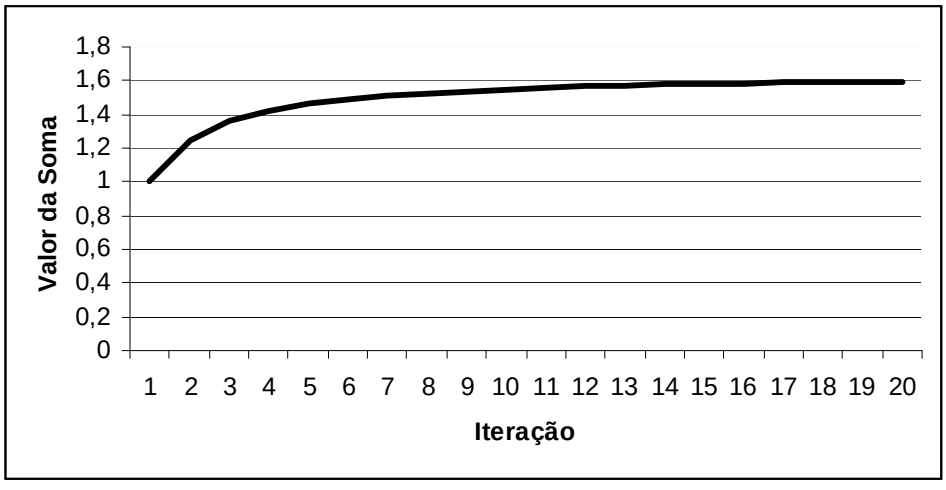
Nesse exemplo, o programa é idêntico ao anterior, mas o leitor deve tomar o cuidado de observar como fica o termo novo ao quadrado. O parêntesis é obrigatório no

divisor, caso contrário o computador dividirá 1 por *i* e o resultado da divisão multiplicar por *i* novamente.

```
Sub serie_quadrado()
Dim i As Integer
Dim n As Integer
Dim soma As Single
Dim novo_termo As Single

n = CInt(InputBox("numero de termos desejado = "))

i = 1
soma = 0

Do While i <= n
     novo_termo = 1 / (i ^ 2)
     soma = soma + novo_termo
     Cells(i, 1) = soma
     i = i + 1
Loop

End Sub
```

Para n = 20, o resultado é

	A
1	1
2	1,25
3	1,361111
4	1,423611
5	1,463611
6	1,491389
7	1,511797
8	1,527422
9	1,539768
10	1,549768
11	1,558032
12	1,564977
13	1,570894
14	1,575996
15	1,58044
16	1,584347
17	1,587807
18	1,590893
19	1,593663
20	1,596163



Esse resultado é interessante, pois empiricamente é possível comprovar a noção matemática de convergência. Essa série é demonstrada em cálculo diferencial e integral

como convergente e o programa comprova que somente com 20 termos os números estacionam em torno de 1,60.

3.4 A Seqüência de Fibonacci

Porque essa seqüência é importante e tem nome especial amplamente difundida nos meios acadêmicos? A seqüência possui a seguinte representação

0 1 1 2 3 5 8 13 21 34 55 ...

Conta-se que esta seqüência foi descrita por Leonardo de Pisa, conhecido como Fibonacci por volta do ano 1200 (DC) para descrever o crescimento de uma população de coelhos. Os números descrevem o número de casais em uma população de coelhos depois de n meses com as seguintes hipóteses:

(i) No primeiro mês nasce apenas um casal.
(ii) Casais reproduzem-se apenas o segundo mês de vida.
(iii) Todo mês cada casal fértil dá luz a um novo casal.
(iv) Os coelhos nunca morrem.

Depois que a seqüência ficou conhecida, muitas aplicações foram encontradas para sua utilização tais como a forma das galáxias, as relações na filotaxia do girassol e do abacaxi, pinturas, arquitetura grega e egípcia, o vôo dos pássaros e por fim se chegou ao *gold number* que resulta no cálculo da razão áurea.

A regra da geração da seqüência de Fibonacci é simples e observando os números percebe-se que sempre um número é gerado pela soma dos dois termos anteriores. Em termos de finanças, a depreciação de ativos é o quanto um determinado investimento perde valor anos seguidos. Essa depreciação é melhor explicada por tabelas baseadas na seqüência de Fibonacci. O detalhe é que a seqüência de Fibonacci pode também ser gerada por uma fórmula fechada, que representa cada termo. A fórmula é conhecida como fórmula de Binet(1786-1856)

$$F_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

Então o que se faz para montar uma tabela de depreciação é a criação de seqüência formada de frações onde os numeradores e denominadores são números de Fibonacci. Assim a melhor tabela de depreciação será criada para o período em anos (variável T):

$$(d_1, d_2, \cdots, d_T) = \left(\frac{F_{2T-1}}{F_{2T}}, \frac{F_{2T-3}}{F_{2T}}, \cdots, \frac{F_1}{F_{2T}}\right)$$

onde nesse caso

$$(F_0, F_1, F_2, F_3, F_4, \cdots) = (0,1,1,2,3, \cdots)$$

Se por exemplo, o período for de T = 4 anos a tabela para depreciação do valor do investimento será

$$(d_1, d_2, d_3, d_4) = \left(\frac{F_7}{F_8}, \frac{F_5}{F_8}, \frac{F_3}{F_8}, \frac{F_1}{F_8}\right) = \left(\frac{13}{21}, \frac{5}{21}, \frac{2}{21}, \frac{1}{21}\right)$$

ou seja, esses valores para anos subseqüentes deverão ser multiplicados pelo valor inicial do investimento ou do ativo.

Outra aplicação para finanças muito utilizada pelos analistas técnicos é a noção de suporte e resistência para o preço de uma ação ou opção em bolsa de valores.

Segundo análise técnica de ações, as ações seguem ciclos de compras e vendas de ações. Os preços oscilam conforme a demanda por uma determinada empresa. Se houver muitos compradores de PETR4 (Petrobrás) e poucos vendedores seu preço sobe. Se houver muitos vendedores e poucos compradores seu preço cai. Essas subidas momentâneas de preços ou quedas podem seguir um certo padrão. Assim, uma linha de suporte representa um preço segundo qual a partir de então ocorre uma reversão e uma subida. Esse é o momento de compra do ativo. Do lado oposto, a linha de resistência informa o preço máximo a partir do qual ao romper essa barreira o preço tende a voltar para valores mais baixos. Esse é o momento de venda do ativo.

Essas linhas de suporte e resistência são conhecidas como linhas de Fibonacci. Essas linhas são traçadas em torno de 38% e 62% do menor preço histórico. E porque esses valores de percentagem. Se os números de Fibonacci forem colocados na planilha do Excel, conforme mostrado a seguir, pode-se fazer a divisão de cada termo em relação ao termo anterior.

Assim, da seqüência de Fibonacci tem-se $x_1 = 1/1 = 1$; depois $x_2 = 2/1=2$ e assim sucessivamente. Na planilha a seguir pode-se verificar na linha 1 a seqüência de Fibonacci e na linha abaixo as divisões sucessivas dos termos.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	0	1	1	2	3	5	8	13	21	34	55	89	144
2			1	2	1,5	1,666667	1,6	1,625	1,615385	1,619048	1,617647	1,618182	1,617978

Observa-se que estas divisões convergem para 1,62, indicando a percentagem de 62%, ou seja a resistência no caso da ação de uma empresa negociada em bolsa de valores. Seu complemento para 100% é 38%, a linha de suporte da ação. É dessa forma portanto que muitos *traders* realizam suas operações de compra e venda observando as linhas de Fibonacci de suporte e resistência.

A geração dos números de Fibonacci, como visto tem suas diversas aplicações, e é claro o primeiro algoritmo de seqüências normalmente apresentado a iniciantes em programação. Isto porque a programação e geração automáticas envolvem a noção de variáveis recebendo valores que no próximo *loop* desaparecem.

Neste ponto é muito importante a noção do uso do chamado *teste de mesa* do programador. O programador coloca no papel, todas as variáveis que programou e acompanha, simulando na mente como se fosse o computador o que ocorre em cada passo.

A idéia é começar nomeando o primeiro valor da seqüência como *ult*, uma variável que da direita para esquerda seria o último número da seqüência. Uma segunda variável necessária seria o *penul*, como penúltimo valor da direita para a esquerda.

	A	B	C	D	E	F
1	0	1	1	2	3	5
2	*ULT*	*PENUL*				

Então deve-se seguir a seguinte ordem de repetição,

Uma vez dentro do *Do While* a repetição deverá ser de tal forma que o valor da variável *ult* recebe o valor de *penul* antes da próxima iteração e a variável *penul* recebe a *soma* anterior. Então quem tinha valor *ult* é descartado para receber *penul*, quem tinha *penul* é descartado para receber a soma e a nova *soma*? A nova soma só realizada depois da realimentação do *loop* no *Do While*

	A	B	C	D	E
1	0	1	1	2	3
2	*ULT*	*PENUL*			
3		*ULT*	*PENUL*		
4			*ULT*	*PENUL*	
5				*ULT*	*PENUL*
6					

O programa então fica,

```
Sub Fibonacci()
Dim i As Integer
Dim n As Integer
Dim soma As Single
Dim ult As Single
Dim penul As Single

n = CInt(InputBox("numero de termos desejado = "))


ult = 0                          ' inicialização de ult'
penul = 1                        ' inicialização de penul'
Cells(1, 1) = ult
Cells(1, 2) = penul
i = 3
Do While i <= n
     soma = ult + penul         ' atualiza a soma dos 2 últimos termos'
     Cells(1, i) = soma         ' impressão da soma'
     ult = penul                ' troca o valor de ult'
     penul = soma               ' troca o valor de penul'
     i = i + 1
Loop

End Sub
```

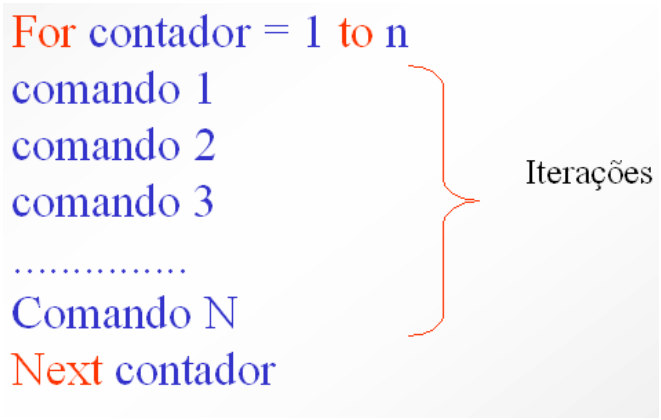
O leitor deve observar que como os dois primeiros termos da série já foram impressos nas células A1 e B1, então a contagem inicial do contador começa em 3 e não em um como os outros programas começavam.

O teste de mesa então fornece como simulação o resultado esperado do computador.

i	***ult***	***penul***	***soma***	***impressão***
3	0	1	1	1
4	1	1	2	2
5	1	2	3	3
6	2	3	5	5

3.5 O comando "for"

Assim como outras linguagens, o VBA- Excel possui uma outra forma de fazer as iterações de programas. O primeiro comando visto nas seções anteriores foi o comando *Do While*. Esse comando, como também já mencionado, faz o programa correr risco de entrar em *loop* infinito se ocorrer o esquecimento da atualização do contador. Sob o comando *for* esse risco não ocorre pois o contador é atualizado automaticamente.



A palavra *for* indica ao computador que para o início apontado logo em seguida até o final da contagem indicada por *to* o computador está preso e deverá repetir as iterações para cada *next*. A comparação se o contador já está na última iteração também é automática e está embutida na primeira linha do comando.

Exemplo 3.11

Fazer um programa para dar a soma de n termos da série

$$S = \sum_{i=1}^{n} \frac{\sqrt{2}}{i^2} = \sqrt{2} + \frac{\sqrt{2}}{4} + \frac{\sqrt{2}}{9} + \cdots \frac{\sqrt{2}}{n^2} + \cdots$$

Outra diferença importante é que o contador não precisa ser iniciado no caso do uso do *for* pois ele é automático. A programação fica muito parecida com as outras séries, trocando-se apenas os termos referentes ao *Do While* por *for*.

	A	B	C	D	E	F	G	H	I	J
1	1,414214	1,767767	1,924902	2,01329	2,069859	2,109143	2,138004	2,160101	2,177561	2,191703

```
Sub serie_for()
Dim i As Integer
Dim n As Integer
Dim soma As Single

n = CInt(InputBox("numero de termos desejado = "))

soma = 0

For i = 1 To n
    soma = soma + Sqr(2) / (i ^ 2)
    Cells(1, i) = soma
Next i


End Sub
```

Pode-se observar que a função do *next i* é dizer ao computador "próximo valor de i" o que será subentendido que i será somado de 1. Mas será possível fazer o contador percorrer valores que não sejam inteiros? Sim é possível usando o comando *step* e indicar de quanto deve ser adicionado o contador em cada passo.

```
Sub teste()
Dim i As Single
Dim n As Integer

n = CInt(InputBox("n="))
soma = 0
t = 1
For i = 1 To n Step 0.1
  soma = soma + Sqr(2) / (i ^ 2)
  Cells(1, t) = soma
  t = t + 1
  Next i
End Sub
```

O *step 0.1* diz ao computador que ele deverá repetir todos os passos dentro do *loop* até o *next i* até n, mas andando de 0,1 em 0,1. Se n = 1 tem-se 10 passos a serem repetidos. Mas de qualquer forma será necessário um contador extra para indicar em qual célula se dará a impressão. Isso porque como o contador do *for* não é um número inteiro, se o programador colocar o próprio *i* entrará em *loop* infinito pois não existe cells(1,0.1), tanto os números de linhas e colunas devem ser inteiros. Por isso foi criado um contador chamado *t* dentro do programa. Pode-se perceber que esse novo contador apenas é um auxílio como marcador das colunas.

Exemplo 3.12

Um *trader* baixou em 5 minutos 2 dados a cada 15 segundos sobre as ações da Petrobrás (Petr4) e da Usiminas (Usim5) conforme tabela a seguir. Fazer um programa para calcular o preço médio nesses 5 minutos e colocar o resultado depois dos últimos dados como mostrado na tabela.

	A	B	C
1		***Petr4***	***Usim5***
2	1	45,12	103
3	2	46	108,59
4	3	46,05	109,7
5	4	46,5	110,75
6	5	46,95	111,5
7	6	46,9	107,49
8	7	47,2	104,49
9	8	46,11	102,5
10	9	46,7	105,2
11	10	46,85	106,89
12	11	46,05	105,6
13	12	46,7	108,74
14	13	45,94	107
15	14	47,1	109,5
16	15	47,68	110,25
17	16	47,99	109,3
18	17	46,91	106,6
19	18	47,4	107
20	19	48	108
21	20	48,12	106
22	**média**		

A média é a soma dos valores das ações e depois a divisão dessa soma pelo número de termos. Então o programa deve pedir ao usuário quantos dados estão na planilha e depois da última linha de dado colocar o valor da média.

$$Média = \frac{\sum_{i=1}^{n} x_i}{n}$$

O leitor deve perceber que nesse caso os valores das células com os dados não poderão ter o mesmo contador como indicador de linha, pois na primeira linha tem-se texto indicando o nome das ações. As células com valores devem começar a partir de i + 1. No final do programa também os valores deverão ser colocados com na linha n + 2. Se os dados começassem na primeira linha a média estaria na linha n + 1, mas como a primeira linha não pode ser computada, então deverá ser mais uma linha para baixo, ou n + 2.

```
Sub media_acao()
Dim i As Integer
Dim n As Integer
Dim soma1 As Single
Dim soma2 As Single

n = CInt(InputBox("n="))
soma1 = 0
soma2 = 0

For i = 1 To n
  soma1 = soma1 + Cells(i + 1, 2)
  soma2 = soma2 + Cells(i + 1, 3)
  Next i
media1 = soma1 / n
media2 = soma2 / n
Cells(n + 2, 2) = media1
Cells(n + 2, 3) = media2

End Sub
```

Duas variáveis *soma* foram criadas para cada uma das ações e nas colunas 2 e 3 pois a primeira coluna apenas marca o número do dado adquirido. O resultado nesse caso será

21	20	48,12	106
22	**média**	46,8135	107,405
23			
24			

Exemplo 3.12

A noção de limite e convergência é muito utilizada em cursos de cálculo diferencial e integral. Pode-se provar que a série abaixo "converge" para o número 0,5. Ou seja,

$$\frac{1}{1*3}+\frac{1}{3*5}+\frac{1}{5*7}+\cdots=\frac{1}{2}$$

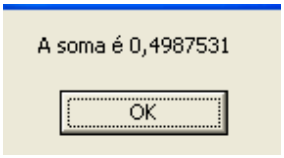
Assim, pede-se fazer um programa em VBA-Excel, onde o usuário forneça o número N de termos desejados e o programa imprime via ***MsgBox*** a soma desses N termos, respeitando a lógica da série acima.

Essa lógica é muito parecida com a geração de múltiplos apresentada anteriormente. Pode-se também utilizar a idéia do algoritmo de Fibonacci,mas gerando os números ímpares subseqüentes. Por exemplo se começar *ult =1* e *penul = 3* e ir adicionado o valor 2 para ambas as variáveis a parte de baixo das frações está pronta. O leitor não pode esquecer de declarar *ult* e *penul* como *Long* ou *Single* pois pode ocorrer o mesmo problema já

apresentado em exemplo anterior de estouro ou *overflow* por atingir o limite de número inteiro.

```
Sub serie_fracao_impar()
Dim i As Integer
Dim ult As Single
Dim penul As Single
Dim soma As Single
Dim n As Integer

n = CInt(InputBox("n="))

ult = 1
penul = 3
soma = 0
For i = 1 To n
     soma = soma + 1 / (ult * penul)
     ult = ult + 2
     penul = penul + 2
Next i

MsgBox (" A soma é " & soma)

End Sub
```

Para o caso n = 200 o resultado é



A flecha indica onde os ímpares são gerados iterativamente para as duas variáveis *ult* e *penul*.

3.6 Séries com sinais alternados

As programações das seções anteriores apenas mencionavam séries com soma dos termos de mesmo sinal. E quando os sinais forem alternados, ora positivo para um termo e ora negativo para outro. Pode-se nesse caso usar a noção de $(-1)^i$ e dessa forma quando i for par o sinal é positivo, ou ainda quando i for ímpar o sinal é negativo. Observe o exemplo a seguir.

Exemplo 3.13

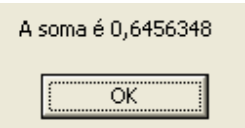
Fazer um algoritmo e implemente um programa em VBA Excel onde o usuário entra apenas com o número de termos desejado e o programa mostra em ***MsgBox*** o valor da soma.

$$S = \frac{1}{1} - \frac{2}{4} + \frac{3}{9} - \frac{4}{16} + \frac{5}{25} - \frac{6}{36} \cdots$$

A solução nesse caso é fácil de perceber pois os termos das frações em cima são valores do contador e embaixo o contador ao quadrado. Para alternar o sinal da soma deve-se colocar $(-1)^{i-1}$ multiplicando as frações. Coloca-se o sinal negativo elevado a (i-1) pois no primeiro passo quando i = 1 a soma deverá ser positiva e igual a 1.

```
Sub serie_alternada()
Dim i As Integer
Dim soma As Single
Dim n As Integer

n = CInt(InputBox("n="))

soma = 0

For i = 1 To n
     soma = soma + (-1) ^ (i - 1) * i / (i ^ 2)
Next i

MsgBox (" A soma é " & soma)

End Sub
```

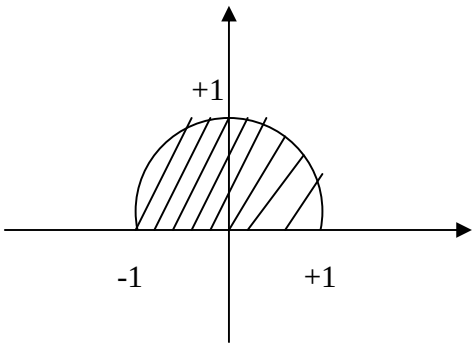
Para n = 10 o resultado será



3.7 Séries com funções matemáticas

Exemplo 3.14

Fazer uma subrotina que leia um número N de pontos com coordenadas x e y fornecidas pelo usuário (o par (x,y) é um ponto no plano) e diga se esse ponto pertence à figura abaixo com os limites $-1 \leq x \leq 1$, $y \geq 0$ e $x^2 + y^2 \leq 1$.



Para cada ponto lido deverá aparecer uma mensagem MsgBox dizendo se o ponto pertence ou não à figura.

Exemplo: (0,0) -------- PERTENCE À FIGURA
(0,2) -------- NÃO PERTENCE À FIGURA

Existem duas maneiras para resolver esse problema. As duas soluções são as seguintes:

Primeira Solução

```
Sub quest1()
Dim i, N As Integer
Dim x, y As Single
N = CDbl(InputBox("N="))
For i = 1 To N
  x = CDbl(InputBox("x="))
  y = CDbl(InputBox("y="))
  If (x >= -1) And (x <= 1) And (y >= 0) And (x ^ 2 + y ^ 2 <= 1) Then
    MsgBox ("O par (x,y) pertence à figura")
  Else
    MsgBox ("O par (x,y) NÃO pertence à figura")
    End If
Next i
End Sub
```

Segunda Solução

```
Sub quest1_2()
'segunda maneira de resolver a questão 1
Dim i, N As Integer
Dim x, y As Single
N = CDbl(InputBox("N="))
For i = 1 To N
  x = CDbl(InputBox("x="))
  y = CDbl(InputBox("y="))
  If (x >= -1) Then
    If (x <= 1) Then
      If (y >= 0) Then
        If (x ^ 2 + y ^ 2 <= 1) Then
    MsgBox ("O par (x,y) pertence à figura")
  Else
    MsgBox ("O par (x,y) NÃO pertence à figura")
      End If
     End If
    End If
  End If
Next i
End Sub
```

Exemplo 3.15

Fazer um programa para calcular o fatorial de um número inteiro N. Sabe-se que a formulação do fatorial é

$$N! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$$

Exemplo 3.16

Fazer um programa para calcular o valor do exponencial de x, onde o usuário fornece o número termos para a precisão deseja e o valor de x e o programa imprime o valor em *MsgBox* usando a série abaixo:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} + \cdots$$

3.8 Séries indiretas

Exemplo 3.17
Escrever abaixo um algoritmo para uma macro em *Excel-VBA*, onde o usuário fornece o número "n" de termos desejados e o programa informa via *MsgBox* a soma da série abaixo:

$$S = \frac{1}{2} + \frac{1}{14} + \frac{1}{35} + \frac{1}{65} + \cdots$$

```
Sub Serie()

Dim ult As Single
Dim soma As Single
Dim x As Single
Dim n As Integer

n = CDbl(InputBox("n="))
ult = 2
soma = 12
x = 1 / ult
For i = 1 To n - 1
  ult = ult + soma
  soma = soma + 9
  x = x + 1 / ult
  Next i
MsgBox (" S =" & x)

End Sub
```

3.9 Exercícios

(1) Fazer um algoritmo e uma macro no Excel, onde deve se ler o número N de uma seqüência de números naturais e posteriormente ler a própria seqüência. Uma vez armazenados os números da seqüência nas células do Excel, o programa deve mostrar em uma determinada célula o valor da soma dos números pares (SP) e em outra o valor da soma dos números ímpares (SI).

(2) Elaborar um algoritmo e uma macro no Excel onde, uma vez conhecido o tamanho N da seqüência e a própria seqüência, armazenar os números nas células do Excel. No final o programa deve mostrar em uma célula a soma dos números negativos e em outra a soma dos números positivos.

(3) Deseja-se ter uma macro onde, fornecidos os valores inteiros A, B e C, verificar se eles formam os lados de um triângulo retângulo. Ao final, a macro deve imprimir uma mensagem dizendo se eles formam ou não lado de um triângulo retângulo.

(4) Gerar e salvar nas células do Excel usando macro, a seqüência: 1, 3, 4, 7, 11, 18, 29, ... até seu vigésimo termo.

(5) Gerar e salvar nas células do Excel usando macro a seqüência: 1, 4, 9, 16, 25, 36, 49,...

(6) Observe a tabela dos rendimentos(%) de um fundo de investimento:

Mês	1	2	3	4	5	6	7	8	9	10	11	12
Ret	7	7,5	6	6,5	8	7,1	6	6	10	9,5	9	9

Fazer um algoritmo e um programa em macro onde:

(i) O usuário entra com os valores via *inputBox* e o programa salva esses valores nas células do Excel.

(ii) O programa descobre o maior rendimento.

(iii) O programa mostra o valor do maior rendimento via MsgBox.

(7) Fazer um algoritmo e um programa em macro onde o usuário entra com N valores de ações de uma empresa via Cdbl e o programa armazena o resultado nas células do Excel. Após isso, o programa deverá fornecer a média desses valores e o desvio padrão (risco da ação).

(8) Fazer um algoritmo para passar n elementos de uma coluna do Excel para uma linha.

(9) Criar um algoritmo para calcular a soma abaixo até o termo N que o usuário desejar.

$$S = \frac{70}{7} + \frac{69}{14} + \frac{68}{21} + \frac{67}{28} + \ldots$$

(10) Fazer um programa que leia um valor X e depois calcule e escreva o resultado do seguinte somatório até o termo N desejado pelo usuário:

$$\frac{X^{25}}{1} - \frac{X^{24}}{2} + \frac{X^{23}}{3} - \frac{X^{22}}{4} + \ldots$$

(11) Fazer um algoritmo e programa em VBA-Excel que leia uma quantidade N de valores numéricos e conte e diga ao usuário quantos pares e quantos ímpares existem.

(12) O método de Newton para encontrar solução numérica para raiz de uma função é muito simples e útil em diversas situações. A fórmula é:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Onde f é a função e f ’ é a derivada da função. Fazer um algoritmo e rode o programa em VBA-Excel para encontrar uma das raízes da função $f(x) = x^2 - 5x + 6$.

(13) Fazer um algoritmo e implemente um programa em VBA Excel onde o usuário entra apenas com o número de termos desejado e o programa mostra em ***MsgBox*** o valor da soma.

$$\mathrm{S} = 1 - \frac{1}{8} + \frac{1}{27} - \frac{1}{64} + \frac{1}{125} \cdots$$

(14) Escrever abaixo um algoritmo para uma macro em *Excel-VBA*, onde o usuário fornece o número “n” de termos desejados e o programa informa via *MsgBox* a soma da série abaixo:

$$S = \frac{1}{2} - \frac{1}{14} + \frac{1}{35} - \frac{1}{65} + \cdots$$

(15) Fazer um programa em VBA onde o usuário entra com o número de termos para a série do cosseno e o valor de x. O programa imprime em *MsgBox* o valor do cosseno usando a série

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots + (-1)^n \frac{x^{2n}}{(2n)!} \pm \cdots$$

(16) Fazer um programa em VBA onde o usuário entra com o número de termos para a série do seno e o valor de x. O programa imprime em *MsgBox* o valor do seno usando a série

$$\text{sen}(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots + (-1)^n \frac{x^{2n\pm1}}{(2n+1)!} \pm \cdots$$

(17) Fazer um programa em VBA onde o usuário entra com o número de termos para a série do seno hiperbólico (senh(x)) e o valor de x. O programa imprime em *MsgBox* o valor do seno usando a série

$$\text{senh}(x) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \cdots \frac{x^{2n+1}}{(2n+1)!} + \cdots$$

(18) A integral abaixo não pode ser resolvida utilizando as técnicas usuais de cálculo diferencial e integral. Mas, pela aproximação da integral em N termos da série abaixo, tem-se uma boa aproximação do seu valor exato.

$$\int_0^x e^{-u^2}\,du = x - \frac{x^3}{3*1!} + \frac{x^5}{5*2!} - \frac{x^7}{7*3!} + \cdots$$

Assim, pede-se que seja desenvolvido uma macro em VBA-Excel onde o usuário entra com a variável N (que representa a quantidade de termos) e x (que representa o limite de integração) e o programa imprime numa *MsgBox* o resultado da integral.