1 Introdução

O objetivo desse relatório inicial do curso de Introdução à Física Computacional é ambientar o aluno à linguagem Fortran (90) e a comandos básicos do sistema operacional Linux, o qual se utilizará durante o curso nas aulas de laboratório. O Fortran, até hoje, carrega importância no meio científico por ser uma linguagem de alta performance. Para certas aplicações, envolvendo cálculo númerico por exemplo, ainda é uma das melhores linguagens para se utilizar. E é por mais uma série de motivos que se escolhe essa linguagem para realizar o curso. Visto isso, foram propostos 5 problemas para serem resolvidos. Primeiramente se apresentará os detalhes de cada programa na seção de Metodologia e só depois se apresentará o obtido com esses na seção Resultados.

2 Metodologia

2.1 Exercício 1

2.1.1 (a)

Pediu-se a construção de uma tabela para dispor o fatorial dos números inteiros entre 1 e 20. Só foi necessário declarar duas variáveis. Uma variável inteira i, para representar as iterações de um loop, ou seja, percorrer os 20 inteiros, e uma real de dupla precisão fac, a qual é o fatorial para um dado i.

$integer\ i$
$real*8\ fac$

Armazenou-se os dados em um arquivo .txt, o qual foi aberto pelo comando a seguir:

$open(10, file = "n!.txt")$

Depois, a estratégia mais simples foi executar um loop com o comando *do*. Usou-se a condição inicial $fac = 1.0d0$ para que se pudesse multiplicar sequencialmente os inteiros de 1 a 20 (comando $fac = fac * dfloat(i)$). O comando dfloat() retorna a conversão de i a um float de precisão dupla. Assim só restou escrever, a cada iteração, o número da iteração com o fatorial correspondente usando o comando *write*.

$do\ i = 1, 20$
$fac = fac * dfloat(i)$
$write(10, *)i, fac$
$enddo$
$close(10)$

2.1.2 (b)

Agora o objetivo era fazer uma tabela do logaritmo do fatorial dos inteiros de 2 a 30. Assim, adotou-se uma estrutura muito similar, com as únicas diferenças sendo: (i) a variável inteira i era iterada de 2 a 30. (ii) o loop do comando *do* imprime $LOG(fac)$ no lugar de fac. LOG() é um comando intrínseco do Fortran, o qual calcula o logaritmo na

base e de um número real.

$do\ i = 2, 30$
$fac = fac * dfloat(i)$
$write(10, *)i, LOG(fac)$
$enddo$

2.1.3 (c)

A letra (c) tem como objetivo verificar a aproximação de Stirling. Ela afirma que:

$$ln(n!)' \approx n \cdot ln(n) - n + \frac{1}{2} ln(2\pi n) \tag{1}$$

Percebe-se que há uma grande vantagem em utilizá-la porque não há a necessidade de calcular o fatorial dos números, o que ficaria computacionalmente cada vez mais difícil para inteiros grandes (é o que torna essa aproximação tão interessante em física estatística). No código denotou-se $ln(n!)'$ pela variável *approx* (real*8). Além disso, declarou-se também $ir = real(i, 8)$, para deixar a notação do código mais limpa.Houve a necessidade de definir pi também. Isso foi feito com a função intrínseca DACOS(), conhecida como "arco-cosseno", isto é, a função inversa do cosseno. O prefixo "D"indica o dobro de precisão.

Como foi pedido, construiu-se uma tabela com quatro colunas dentro do loop, similarmente ao item (b). A primeira coluna indica o número inteiro (i) para o qual se quer avaliar o logaritmo do fatorial, a segunda indica LOG(fac), usando a função intrínseca do Fortran, a terceira indica o valor da aproximação de stirling para o dado i. Por fim, a quarta coluna indica a diferença percentual dos valores (d=$[LOG(ir) - approx]/LOG(ir)$). A variável d também foi introduzida para facilitar a leitura do código.

Declaração de variáveis, definição de π e condição inicial de fac

$integer\ i$
$real * 8\ fac, pi, approx, ir, d$
$open(10, file = "1.1c.txt")$
$fac = 1.0d0$
$pi = DACOS(-1.0d0)$

Iterações do comando *do*. A cada iteração se calcula ir, fac, $approx$ e d. Imprime-se as 4 colunas com o comando $write$.

$do\ i = 2, 30$
$ir = real(i, 8)$
$fac = fac * ir$
$approx = (ir * LOG(ir)) - ir + (LOG(2 * pi * ir)/2)$
$d = (LOG(fac) - approx)/(LOG(fac))$
$write(10, *)\ i,\ LOG(fac),\ approx,\ d$
$enddo$

2.2 Exercício 2

Nesse exercício, o objetivo era calcular cos(x) a partir de sua série de Taylor centrada em x=0 com uma precisão de 10^{-6}. Trabalhará-se em dois loops diferentes. Para isso, declarou-se as seguintes variáveis:

$integer\ i, j, n$
$real * 8\ fac, soma, x, inc$

Um dos loops será o da série de Taylor, envolvendo a variável i. Análogo ao Exercício 1, fac é o fatorial de $2i$, calculado a cada iteração de um loop. *soma* é o valor da série de taylor até o i-ésimo termo, sendo também outra variável que é calculada a cada iteração. Além disso, *inc*, abreviação de incremento, é o i-ésimo termo. Por exemplo, sabe-se que a série de taylor do cosseno é dada por

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} \quad = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots \tag{2}$$

Ou seja, no loop, quando $i = 1$, *soma* será equivalente a $1 - \frac{x^2}{2!}$, enquanto *inc* será $-\frac{x^2}{2!}$. Porém, quando $|inc| < 10^{-6}$, a soma deixará de ser atualizada e o loop acaba.

Quanto às variáveis n e j, elas existem porque a entrada dos valores de x será feita por um arquivo. A primeira linha do arquivo descreve quantos valores de x há para se calcular. Atribui-se esse valor a n com o comando *read*. E n será, portanto, o número de vezes que se calculará a série de Taylor.

Abre-se o arquivo que possui, primeiro o número n de valores a serem calculados e, depois, os valores. Abre-se também o arquivo de saída dos valores de cosseno.

```
open(10,file="input.txt")
read(10,*)n
open(20,file="output.txt")
```

O primeiro *do* $j = 1, n$ itera sobre os n valores de x para os quais se quer cos(x). fac e *soma* são colocados nas suas condições iniciais a cada iteração de j. O loop interno *do* $while(.TRUE.)$, atualiza fac, *inc*, i e *soma* a cada iteração, compondo a série de Taylor. O comando *write* imprime o valor de x sendo usado (x), a ordem da expansão necessária (i), o valor *soma* da série de Taylor, e o valor obtido diretamente pela função DCOS(X) intrínseca do Fortran, para que se possa comparar com *soma*.

```
do j = 1,n
read(10,*)x              ! lendo o valor de x dentro do arquivo
fac = 1.0d0              ! condições iniciais da série de taylor
soma = 1.0d0
i = 1

    do while(.TRUE.)
    fac = fac*dfloat(2*i)*dfloat((2*i)-1)

    inc = ((-1.0)**(i))*((x)**dfloat(2*i))/fac
```

$if(abs(inc) >= (10.0 * *(-6)))then$
$soma = soma + inc$

$else$
$EXIT$
$end\ if$

$enddo$
$write(20, *)x, i, soma, DCOS(x)$
$i = i + 1$! $iterador$
$enddo$

2.3 Exercício 3

Nesse exercício o objetivo era, dada uma lista com n números reais, calcular a média e o desvio padrão desses números. Adicionalmente, devia-se verificar a equivalência de duas formas de calcular o desvio padrão. Isto é,

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \langle x \rangle)}{N}} = \sqrt{\langle x^2 \rangle - \langle x \rangle^2} \tag{3}$$

Há como verificar algebricamente, mas também há a opção de se verificar numericamente. No caso, optou-se pelo método numérico por conta da natureza do presente trabalho. As variáveis declaradas são as seguintes

$integer\ n, i$
$real * 8\ sumxi,\ sumxi2,\ avg,\ avg2,\ sumd$
$real * 8, dimension(:), allocatable :: x$

n -> será a quantidade de números que se calculará os parâmetros estatísticos
i -> variável de iteração do vetor x
x -> um vetor em que cada entrada será um elemento da lista de input (cada entrada é x_i)
sumxi= $x_1 + x_2 + ... + x_i$ avg= $(\sum_{i=1}^{n} x_i)/n$

sumxi2= $(x_1)^2 + (x_2)^2 + ... + (x_i)^2$ avg2=$(\sum_{i=1}^{n} x_i^2)/n$

sumd= $\sum_{i=1}^{n}(x_i - avg)^2$

Depois de declarar as variáveis, o próximo passo é definir o tamanho do vetor. Isso pode ser feito colocando a primeira entrada do programa como o número n de termos. Assim, basta ler e usar o comando *allocate*

$open(2, file = "vetor.txt")$
$read(2, *)n$
$allocate(x(n))$

Agora vão ser calculadas as duas médias (avg e avg2)

$doi = 1, n$
$read(2, *)x(i)$
$sumxi = sumxi + x(i)$
$sumxi2 = sumxi2 + (x(i)) * *2$
$enddo$

$avg = sumxi/n$
$avg2 = sumxi2/n$

Depois basta abrir um outro arquivo ($open(20, file = "result.txt")$) e escrever o resultado de avg. Segue depois o cálculo do desvio padrão. Há primeiro a necessidade de calcular $sumd$.

$doi = 1, n$
$sumd = sumd + (x(i) - avg) * *2$
$enddo$

Agora basta printar os resultados no arquivo

$write(20, *)$ "$desvio\, padrao(1) :$ ", $sqrt(sumd/n)$, "$desvio\, padrao(2) :$ ", $sqrt(avg2 - (avg * *2))$

2.4 Exercício 4

O objetivo desse exercício era ordenar os M primeiros números de uma lista com N números reais ($M \leq N$). O programa deveria necessariamente usar um vetor para ler os N números, entretanto, decidiu-se montá-lo diferentemente do mostrado no Exercício 3. No começo da lista colocava-se na primeira linha o N, depois, na segunda linha, o M. Posteriormente, colocava-se os elementos da lista. Assim, o programa ficou dessa maneira:

$program\ sort$
$implicit\ none$
$integer\ n, m$

$open(10, file = "lista.txt")$
$read(10, *)n$
$read(10, *)m$

$CALL\ Bsort(n, m)$

$close(10)$
$endprogram$

Dessa maneira, leu-se os valores de N e M e depois chamou-se a subrotina $Bsort$, que executa um algoritmo conhecido como "bubble sort". Ele foi construído da seguinte maneira.

Primeiro declara-se as variáveis necessárias. Os inteiros x e y serão ocupados por n e m respectivamente. O vetor $v(x)$ será a lista, e a é uma variável auxiliar.
$subroutine\ Bsort(x, y)$

```
integer x, y
real*8 v(x), a
```

Agora ocorre a leitura da lista a partir da terceira linha.

```
do i = 1, x
read(10, *)v(i)
enddo
```

Segue o ordenamento da lista. Abre-se um novo arquivo, no qual já serão impressos os resultados na ordem crescente até o y-ésimo termo (repare que o primeiro loop só ocorre y vezes). O que o algoritmo faz é basicamente fixar um termo por vez, e tornar esse termo o menor possível vendo todas as opções de índice maior. Por exemplo, escolhe-se o primeiro termo da lista desordenada. Olha-se o segundo, o terceiro..., até que, se um termo maior é encontrado, faz-se se esse trocar de lugar com o primeiro (condição do if). Executa-se isso até o último termo da lista. Somente quando o primeiro termo é de fato o menor, fixa-se o segundo e se repete o algoritmo. Ou seja, no segundo loop ($do\ j = i, x$), só há necessidade de olhar os elementos a partir de i, pois todos os termos com índice menor do que i já foram ordenados. Isso é chamado em algumas situações de "Bubble sort melhorado", mesmo que o grau de complexidade não diminua tanto. Ademais, a variável a é usada para fazer a troca dos valores de $v(j)$ com $v(i)$.

```
open(20, file = "sorted.txt")
do i = 1, y
      do j = i, x
      if (v(i) > v(j)) then
            a = v(j)
            v(j) = v(i)
            v(i) = a
      endif
      enddo
write(20, *)v(i)
enddo
close(20)
end subroutine
```

2.5 Exercício 5

Nesse exercício, o objetivo é, dada uma matriz quadrada M, calcular o autovalor/autovetor dominante. Existe um algoritmo, costumeiramente chamado de método da potência, que consegue computar, até uma certa precisão, esse autovalor. Sabemos que o autovalor dominante segue o seguinte limite:

$$\lambda = \lim_{k \to \infty} \vec{x}_k \cdot (M \times \vec{x}_k) \tag{4}$$

$$x_k = \frac{M^k \times x}{|M^k \times x|} \tag{5}$$

Onde x é qualquer vetor não nulo. Como na prática não se tem como chegar a k infinito, avalia-se sucessivamente para ks maiores até que se chegue numa certa precisão desejada (no código como *eps*)ou até que se chegue em um k específico (no código como c). A figura 1 abaixo explica como funciona o programa. Na prática, a cada iteração, aumentava-se o k de 1, pois gostaria-se de analisar exatamente em qual iteração que o lambda obtinha a precisão desejada.

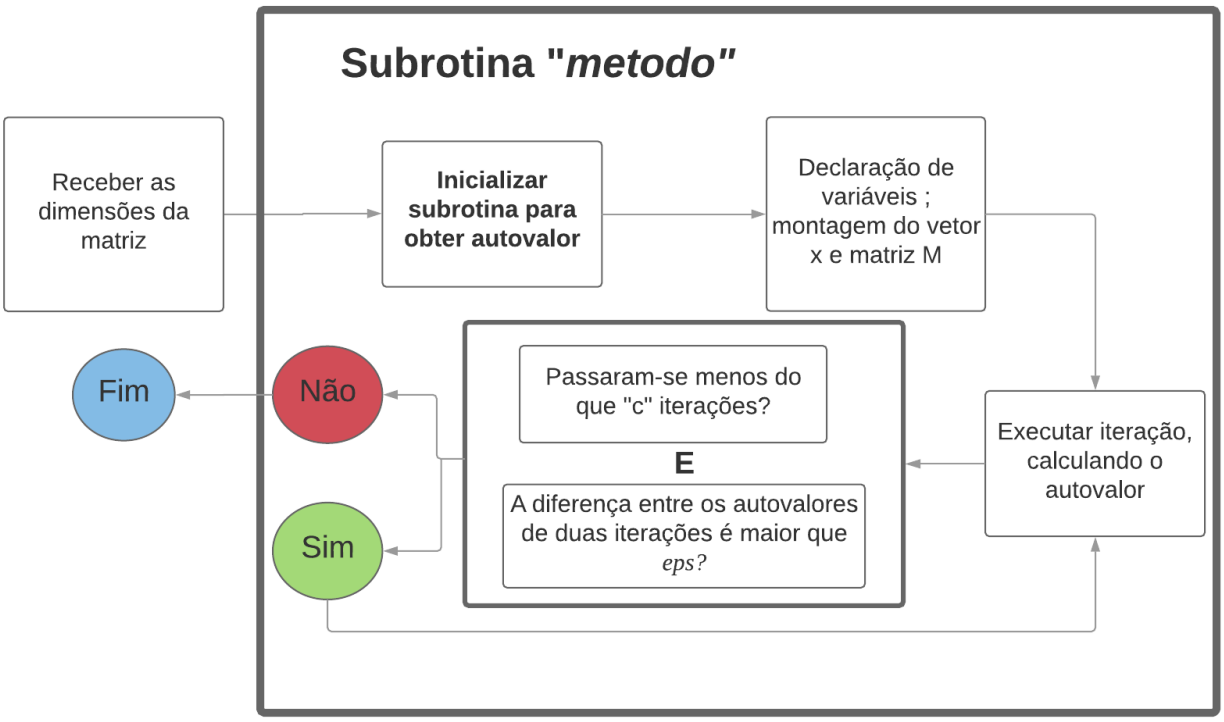

Figura 1: fluxograma do programa

Um exemplo de input do programa é esse
3 (dimensão d da matriz)
1
1 (três entradas do vetor x)
1
2 8 10
8 4 5 (matriz M)
10 5 7
Os primeiros comandos do programa são destinados a ler todos esses valores. Fora da subrotina, lê-se primeiramente d. Este é então o corpo do programa:

program quantica
implicit none
integer d
$open(10, file = "entrada.txt")$
$read(10, *)d$
$CALL\ metodo(d)$
$close(10)$

end program

Agora, pode-se mostrar o que há dentro da subrotina. Primeiramente, a declaração de variáveis.

subroutine metodo(d)
integer d, c
real $*8$ $M(d,d),\ x(d),\ x1(d),\ suma,\ lambda,\ lambdac,\ eps$

M e x já foram descritos. $x1$ armazena o valor de $M \times x$ para a iteração atual e depois gera o vetor x da próxima iteração quando normalizado. *suma* é uma variável auxiliar para executar a multiplicação de matriz em cada linha, somando o resultado das d multiplicações. *lambda* é o autovalor dominante na iteração k e lambdac o é na iteração $k-1$. *eps* é a precisão especificada e c é o contador da iteração. Agora, segue a "Montagem do vetor x e da matriz M".

$doi = 1, d$
$read(10, *)x(i)$
enddo

$doi = 1, d$
$read(10, *)(M(i,j), j = 1, d)$
enddo

As seguintes condições iniciais são estabelecidas antes de entrar no loop

$x = x/(norm2(x))$ Nomarlização do vetor x inserido
$eps = 0.000000000000001$ Declaração da precisão
$c = 0$ Inicialização do contador
$lambda = 0.0d0$ Inicialização do lambda
$open(20, file = "output2.txt")$

O código do loop foi dividido em três partes. (I) é um loop interno que faz a multiplicação $M \times x$ e atribui o vetor resultante a $x1$. (II) é um loop para obter $\lambda = \vec{x} \cdot \vec{x}_1$. Ou seja, o descrito pela equação (4). Depois disso, (III) atualiza os parâmetros para que um novo loop possa começar. $x1$ é normalizado e atribuído a x, o lambda da iteração anterior é atualizado e o contador aumenta de 1. Depois, escreve-se no arquivo c, *lambda* e x. Lembrando que quanto maior a iteração, mais o valor de x tende para o autovetor dominante (consequência natural de o limite da equação (4) resultar no autovalor dominante). Além disso tudo, verifica-se se já foi encontrada a precisão que se definiu no começo. Caso positivo, o programa acaba.

do while $(c < 200)$

! (I)
do $i = 1, d$
$suma = 0.0d0$
do $j = 1, d$
$suma = suma + M(i,j) * x(j)$
enddo

```
        x1(i) = suma
        enddo

!       (II)
        lambda = 0
        do i = 1, d
            lambda = lambda + x(i) * x1(i)
        enddo

!       (III)
        x = x1/norm2(x1)
        c = c + 1
        lambdac = lambda
        write(20, *)"iteracao", c, "autovalor", lambda, "autovetor", x, abs(lambdac−lambda)

        if (abs(lambdac − lambda) < eps) EXIT
    enddo
```

2.6 Adendo

Vários programas desse relatório poderiam ter sido feitos de forma que não seria necessário colocar o número N de entradas no começo do *input*. Basta, por exemplo, fazer um vetor alocável e, depois, ler a lista uma vez para contar o número de linhas. Por exemplo, com o código a seguir:

```
OPEN(1, file =' file.txt')
DO
    READ(1, *, END = 10)
    n = n + 1
ENDDO
10 CLOSE(1)
```

Esse código lê as linhas do programa e soma um ao n cada leitura. Quando se tenta ler a última linha, a especificação $END = 10$ aciona o comando 10 fora do loop e fecha o arquivo. Dessa forma, tem-se o número N total de entradas.

3 Resultados e Discussão

3.1 1(a)

Obteve-se o resultado a seguir

n	$n!$
1	1.0000000000000000
2	2.0000000000000000
3	6.0000000000000000
4	24.000000000000000
5	120.00000000000000
6	720.00000000000000
7	5040.0000000000000
8	40320.000000000000
9	362880.00000000000
10	3628800.0000000000
11	39916800.000000000
12	479001600.00000000
13	6227020800.0000000
14	87178291200.000000
15	1307674368000.0000
16	20922789888000.000
17	355687428096000.00
18	6402373705728000.0
19	1.2164510040883200E+017
20	2.4329020081766400E+018

Os resultados possuem exatidão, isso pode ser comparado diretamente com o "Wolfram alpha", em https://www.wolframalpha.com. Entretanto, a precisão vai até o 16º algarismo significativo. Isso fica claro para fatoriais de números grandes, como 60!, quando esse algarismo varia. Porém, a partir de 22! já se percebe variação no 17º algarismo do número fornecido (17 no total).

3.2 1(b)

Tabela 1: My caption

n	$LOG(n!)$
2	0.69314718055994529
3	1.7917594692280550
4	3.1780538303479458
5	4.7874917427820458
6	6.5792512120101012
7	8.5251613610654147
8	10.604602902745251
9	12.801827480081469
10	15.104412573075516
11	17.502307845873887
12	19.987214495661885
13	22.552163853123425
14	25.191221182738680
15	27.899271383840890
16	30.671860106080672
17	33.505073450136891
18	36.395445208033053
19	39.339884187199495
20	42.335616460753485
21	45.380138898476908
22	48.471181351835227
23	51.606675567764377
24	54.784729398112319
25	58.003605222980518
26	61.261701761002001
27	64.557538627006338
28	67.889743137181540
29	71.257038967168015
30	74.658236348830158

Os valores também concordam todos com o calculado no $wolfram$, estando corretos dentro do limite de precisão das variáveis reais utilizadas.

3.3 1(c)

Tabela 2: My caption

n	$LOG(n!)$	$Stirling(n)$	diferença percentual
2	0.69314718055994529	0.65180648460453594	5.9642017041767491E-002
3	1.7917594692280550	1.7640815435430570	1.5447344445693059E-002
4	3.1780538303479458	3.1572631582441804	6.5419508962468237E-003
5	4.7874917427820458	4.7708470515922246	3.4767038950857614E-003
6	6.5792512120101012	6.5653750831870301	2.1090741751494313E-003
7	8.5251613610654147	8.5132646511195222	1.3954820843890348E-003
8	10.604602902745251	10.594191637483277	9.8176851669556312E-004
9	12.801827480081469	12.792572017898760	7.2297976184338883E-004
10	15.104412573075516	15.096082009642155	5.5153177212670672E-004
11	17.502307845873887	17.494734170385936	4.3272439010014370E-004
12	19.987214495661885	19.980271655554681	3.4736406659921081E-004
13	22.552163853123425	22.545754858935421	2.8418533271324276E-004
14	25.191221182738680	25.185269812625918	2.3624778130404078E-004
15	27.899271383840890	27.893716650288930	1.9909959208389360E-004
16	30.671860106080672	30.666652450161063	1.6978611344723750E-004
17	33.505073450136891	33.500172054188454	1.4628817202061515E-004
18	36.395445208033053	36.390816054283718	1.2719046910608648E-004
19	39.339884187199495	39.335498626950255	1.1147872800975242E-004
20	42.335616460753485	42.331450141061481	9.8411693044942686E-005
21	45.380138898476908	45.376170944258263	8.7438124143291159E-005
22	48.471181351835227	48.467393733766784	7.8141649590723533E-005
23	51.606675567764377	51.603052607539688	7.0203325147961227E-005
24	54.784729398112319	54.781257376729350	6.3375714749597394E-005
25	58.003605222980518	58.000272067343786	5.7464628688476174E-005
26	61.261701761002001	61.258496790773947	5.2316049602376052E-005
27	64.557538627006338	64.554452348323721	4.7806634953166525E-005
28	67.889743137181540	67.886767073197987	4.3836724754421305E-005
29	71.257038967168015	71.254165517805660	4.0325130036325496E-005
30	74.658236348830158	74.655458673900412	3.7205204215749244E-005

Verificou-se por certo o ótimo desempenho da aproximação de Stirling, visto que conforme cresce o número n, menor o erro percentual cometido na aproximação. E os resultados indicam que essa relação é seguida indefinidamente para n grande.

3.4 Exercício 2

Abaixo consta um exemplo de *input* do programa, e depois, na tabela 4, o resultado gerado. O índice "Ordem i", diz qual é a ordem final da expansão da série de Taylor para obter a precisão de 10^{-6}.

Tabela 3: Um exemplo de input do programa.

5
0.5
1.0
1.57
5.0
10.0

Tabela 4: Resultados de acordo com o input dado.

x	Ordem i	Expansão de ordem i	$COS(x)$
0.50000000000000000	3	0.87758246527777772	0.87758256189037276
1.0000000000000000	4	0.54030257936507931	0.54030230586813977
1.5700000000000001	5	7.9586475794951204E-004	7.9632671073326335E-004
5.0000000000000000	11	0.28366209297230688	0.28366218546322625
10.000000000000000	18	-0.83907134946059092	-0.83907152907645244

Escolheu-se dois valores pequenos de x (0.5 e 1.0), e percebeu-se que o valor era obtido dentro da precisão esperada. Já para x se aproximando de $\pi/2$ (1.57), quanto mais se aproxima, menor o valor real é. Então, caso se queira que a série de Taylor gere o resultado real, é necessário que se aumente a precisão declarada de 10^{-6}, cobrindo mais casas decimais. Para 1.57, nota-se que houve concordância de resultado. Percebe-se também que o programa também funcionou para valores grandes de x. Porém, necessitou de ordens muito maiores para a expansão. Então, não há como concluir exatamente qual é a ordem de expansão necessária, pois isso depende muito do valor de x. Em geral, quanto maior x, maior a ordem de expansão.

3.5 Exercício 3

Há um exemplo do input para o programa a seguir. Lembrando que a primeira entrada é o número de elementos que se realizará as operações estatísticas.

14
2.0
3.0
4.4
4.2
4.3
4.4
5.7
6.7
0.5
0.5
0.4
7.8
102.2
104.2

Agora, os resultados, exatamente como saem do programa.
media aritmetica : 17.878571428571430
desvio padrao(1) : 34.902111654729197 *desvio padrao*(2) : 34.902111654729197

Pode-se usar como comparação o site http://www.calculator.net/statistics-calculator.html, o qual possui uma calculadora estatística com precisão maior do que o resultado final do programa. Percebe-se que o programa tem resultados concordantes até o primeiro algarismo significativo duvidoso, onde se encontra o limite da precisão. Dessa forma, obteve-se sucesso no cálculo do desvio padrão. Além disso, as duas maneiras de se calcular o desvio padrão concordam até a última casa decimal. Verifica-se então, numericamente, que elas são equivalentes.

3.6 Exercício 4

Um *input* típico é apresentado a seguir. Lembrando que a primeira entrada é o número N total de elementos a serem organizados e a segunda diz quantos se quer visualizar na resposta final (M).

10
8
0.05
0.01
0.02
0.3
0.0345
0.042
0.0101
-0.1
0.2
0.21

Agora, o resultado gerado pelo *input*.
-0.10000000000000001
1.0000000000000000E-002
1.0100000000000000E-002
2.0000000000000000E-002
3.4500000000000003E-002
4.2000000000000003E-002
5.0000000000000003E-002
0.20000000000000001

A ordenação está de fato correta. O único detalhe é que o programa gera um erro no último algarismo. Isso é natural da precisão dos números do tipo real*8, pois o último algarismo é duvidoso. Caso números com muitos algarismo significativos estivessem na lista, seria necessário aumentar o precisão (para um real de 32-bits, por exemplo) .

3.7 Exercício 5

Como já foi mostrado o tipo de *input* do programa, vai se mostrar somente as respostas obtidas. Usou-se como referência o seguinte site http://www.arndt-bruenner.de/mathe/scripts/en_eigenwert2.htm, visto que representava os resultados com diversos alagarismo significativos.

Para a matriz 3x3, a convergência até os 16 algarismos significativos foi bem rápida. Assim, vai se mostrar o resultado integral do programa, com todas as iterações. O autovetor dominante x teve suas três componentes separadas nas colunas.

iteração	autovalor(λ)	$x(1)$	$x(2)$	$x(3)$
1	19.666666666666664	0.58395716247378382	0.49636358810271625	0.64235287872116209
2	19.884057971014492	0.58148525902372283	0.49631822613640997	0.64462633512983425
3	19.884219060289229	0.58235995694473386	0.49588474649808367	0.64417016287457696
4	19.884233816752324	0.58204398248288947	0.49603463733465725	0.64434031460070307
5	19.884235738609640	0.58215805966697298	0.49598043468402792	0.64427897837460746
6	19.884235989086637	0.58211687751218344	0.49600000224740870	0.64430112423154773
7	19.884236021731514	0.58213174500294973	0.49599293821572527	0.64429312948374085
8	19.884236025986148	0.58212637765862896	0.49599548844627367	0.64429601572179984
9	19.884236026540659	0.58212831534600862	0.49599456778078138	0.64429497375377198
10	19.884236026612928	0.58212761581483830	0.49599490015389686	0.64429534991885351
11	19.884236026622347	0.58212786835517427	0.49599478016271487	0.64429521411818746
12	19.884236026623576	0.58212777718467779	0.49599482348117235	0.64429526314408581
13	19.884236026623736	0.58212781009847026	0.49599480784261962	0.64429524544506933
14	19.884236026623757	0.58212779821614280	0.49599481348835034	0.64429525183465564
15	19.884236026623757	0.58212780250582496	0.49599481145016461	0.64429524952792794

Valores obtidos pela fonte citada:
$\lambda_f = 19.884236026623757$
$x_f = [0.903510931116852\ ;\ 0.7698253430910478\ ;\ 1]$

Percebe-se que o autovalor concordou totalmente, mas resta ajustar a disposição dos resultados dos autovetores para verificar como a precisão desses é em relação à dos autovalores. Pode-se dividir o resultado da última iteração pelo escalar de módulo igual ao da terceira componente. Assim, teria-se x=[0.9035109337409320 ; 0.7698253429830153 ; 1.0000...]. É natural observar que há uma divergência dos valores a partir de alguma casa decimal (oitava ou nona). Isso quer dizer que a precisão do cálculo dos autovetores é menor em relação a do cálculo dos autovalores.

O mesmo arquivo foi gerado para a matriz 4x4, porém, pela extensão, decidiu-se apresentar só a informação da iteração final.

Número da iteração: 103
λ = 14.199731058871633
x =[0.63858773230697730 ; -0.61048933755372314 ; 0.46761685125730906 ; 2.9033726947263489E-002]

ou, reformatado,

x= [21.99468685046533; -21.02690221832727; 16.10598777437986; 1]

Agora, de acordo com a fonte

λ_f=14.199731058871647
x_f=[21.99471424182236 ; -21.026930326193607 ; 16.106007960000234 ; 1]

Novamente, uma ótima concordância para λ, pois o erro permanece no limite da precisão do tipo de variável real declarada. Já para x, o autovetor perdeu precisão em relação ao caso da matriz 3x3 (acompanhando o resultado exato até o quinto ou sexto algarismo).

Por último, para a matriz 5x5

Número da iteração : 60
λ = -17.764516417326490
x =[0.27703962144505750 ; 0.23506094463394908 ; 0.31813190793521323 ; 0.46122747156109062 ; 0.74434985664554210]

Reformatando:

x=[0.3721900648890474 ; 0.3157936318994746 ; 0.4273956730090525 ; 0.6196380202713281 ; 1]

Informação da fonte

λ_f= -17.764516417326515
x_f=[0.37219004359682095 ; 0.31579365636774576 ; 0.42739565740759844 ; 0.6196380221928419 ; 1]

Observa-se que a diferença entre os dois autovalores $\lambda_f - \lambda = 2,9 \cdot 10^{-14}$ ocorre ainda no primeiro algarismo duvidoso, mas é maior do que a dos casos anteriores. Isso pode ter ocorrido por conta da acumulação de erros ser maior, visto que para a matriz 5x5 faz-se mais operações por iteração. Uma outra explicação seria um erro de convergência do programa. Para verificar se isso ocorre para um dado algarismo significativo, bastaria aumentar a precisão do número real utilizado. Porém, esse não deve ser o caso, pois os autovetores convergiram razoavelmente bem, melhor até do que para a matriz 4x4. Outro dado interessante é que a convergência da matriz 5x5 foi mais rápida do que para a matriz 4x4.

Isso traz um questionamento sobre o que afeta a convergência. Para todos os três resultados acima utilizou-se os vetores unitários ((1, 1, 1),(1, 1, 1, 1) e (1, 1, 1, 1, 1) respectivamente). Percebeu-se que o número de iterações costumava aumentar conforme se colocava uma das entradas como nula, ou também quando se trocava qualquer outro número. E de fato, existem certas matrizes que pode-se fazer uma estimativa sobre a taxa de convergência dos autovalores e dos autovetores. Para o caso das matrizes hermitianas, existe um teorema, enunciado nesse documento http://people.inf.ethz.ch/arbenz/ewp/Lnotes/chapter7.pdf na página 9, que cita como a convergência depende da "qualidade"do autovetor (significado matemático explicitado no documento), e da razão, em módulo, entre o segundo

maior e o maior autovalor. Como a razão é uma propriedade intrínseca da matriz, o único fator que se pode de fato mudar para que ocorra uma melhor convergência é a escolha do vetor. Ademais, o autor afirma que para matrizes simétricas, o vetor unitário tem uma convergência rápida, frente a diversas outras escolhas arbitrárias.

Em geral, propriedades interessantes sobre o cálculo dos autovalores/autovetores só surgem quando a matriz possui certas propriedades. Como o programa escrito recebe uma matriz arbitrária, não há perda de generalidade ao se usar um vetor unitário (menos quando esse é ortogonal ao autovetor dominante), ou também um vetor com as entradas sendo geradas aletoriamente, como já é feito por muitos que se utilizam do algoritmo do programa.

Por fim, pode-se afirmar que o Exercício 5 teve resultados concordantes com o esperado. Muitos aprimoramentos poderiam ser feitos, mas isso sempre depende da intenção do programa. Ele foi escrito então com uma forma bem geral, de acordo com as intenções do exercício.