Gabarito do teste 1 do semestre letivo 2020/2

Questão 1 (cópia dos slides)
Erros de Truncamento (ou de discretização): erros que aparecem devido à resolução do problema matemático via um método aproximado.

Erros de Arredondamento (ou de quantização): erros que aparecem devido à representação aproximada dos valores reais pelos computadores, isto é, devido à quantidade limitada de dígitos que as máquinas conseguem armazenar.

Questão 2
Usando *3 dígitos na mantissa , com arredondamento para o mais próximo e o código abaixo.*

```
x(n)= b(n)/ A(n,n);
Para i de (n-1) ate 1, passo (-1)
{
     s = 0;
    Para j de (i+1) até n
    {
            s = s + A(i,j)*x(j)
    }
    MOSTRAR (s)
    x(i)= (  b(i) - s )/ A(i,i);
    MOSTRAR (x(i) )
} % Fim do para i
MOSTRAR (x)
FIM
```

$$
\begin{aligned}
x_1 + 2x_2 + 3x_3 + 4x_4 &= 1\\
101x_2 + 51x_3 + 2003x_4 &= -70\\
3x_3 + (-40)x_4 &= 2\\
7x_4 &= 300
\end{aligned}
$$

Ultima equação

$7\ x_4 = 300$

$x(4) = 300/7 = 42{,}857\ldots \rightarrow 42{,}9$

i=3

$3x_3 + (-40)\,x_4 = 2$

$s = s + A(4,3)*x(4)\ :\quad 0 + (-40)(42{,}9) = -1716 \rightarrow -1720$

$x(3) = (2 - (-1720))\,/\,3 = 1722/3 = 1720/3 = 573.333\ldots \rightarrow 573$

i=2

$101x_2 + 51x_3 + 2003x_4 = -70$

$s = s + A(2,3)*x(3) = 51(573) = 29\ 223 \rightarrow 29200$

$s = s + A(2,4)*x(4) = 29200 + 2003\ (42{,}9) = 29200 + 85900 = 115\ 100$

$x(2) = (-70 - 115000)\ /101$

$= (-115070)/101$

$= (-115000)/101 = -1140$

`i=1`

$x_1 + \quad 2\,x_2 \quad + \; 3\,x_3 \; + \quad 4\,x_4 \quad = \; 1$

    s = s + A(1,2)*x(2)= 0+ 2(-1140)= -2280
    s = s + A(1,3)*x(3)= -2280 + 3(573)= -2280 + 1719 → -2280+ 1720 = -560
    s = s + A(1,4)*x(4) = -560 +172= -388

    s = -388
    x(1) =(1-(-388))/1= -389

Questão 3

4 dígitos na mantissa . Com arredondamento por corte

e^1 ~ (((((1/0!) + (1/1!)) + (1/2!)) + (1/3!)) + (1/4!))

Usando 4 dígitos na mantissa tem-se

(((((1/0!) + (1/1!)) + (1/2!)) + (1/3!)) + (1/4!))
((((1 + 1) + 0.5) + 0.1666) + 0.04166)
(((2.0 + 0.5) + 0.1666) + 0.04166)
*((**2.5** + **0.1666**) + 0.04166)*
*(**2.666** + 0.04166) (Obs.:o valor seria 2.70766)*
2.707
e^1 ~ 2.707

<u>Extras:</u>

Com N=4 <u>Valor com ordem inversa</u>

começa se colocando a contribuição dos termos pequenos, então a soma deles não perde tantos dígitos quando é acrecentado aos termos de maior ordem

(0.04166 + 0.1666) = 0.2082 (o valor seria 0.20826)
0.5 + 0.2082 = 0.7082
1/1! + *0.7082 = 1.708*
(1/0!) + 1.708 = 2.708

Obs: Valor e com 16 digitos
2.718 281 828 459 045

Questão 4

a) Pseudo código. Versão que começa o **for Com N e vai até 1**
(serie do termo *(1/N!) e vai até o termo (1/0!))*

```
Código versao Decrescente
N=input('Digite o valor de N: ')
printf('Calculando uma aproximacao para e \n' );
  s=0;
  for i = N:(-1):1
    termo= 1/( factorial(i));
    s = s + termo;
  endfor %i
  s= 1.0 + s;
  erro_rel= abs ( (s-(e^1)) )/ (e^1);
```

```
b)  para N =  5 Pela versao  DECrescente
O valor de e: 2.7166666666666668 ;        erro_rel =  0.00059418
-----------------------
formaCrescente
Digite o valor de N: 5
termo =  1
termo =  0.50000
termo =  0.16667
termo =  0.041667
termo =  0.0083333
Pela versao    Crescente
O valor de e: 2.7166666666666663 ;       erro_rel =  0.00059418

formaDecrescente
Digite o valor de N: 5
termo =  0.0083333
termo =  0.041667
termo =  0.16667
termo =  0.50000
termo =  1
Pela versao  DECrescente
O valor de e: 2.7166666666666668 ;        erro_rel =  0.00059418
```

c) Para N = 10

formaCrescente
Digite o valor de N: 10
termo = 1
termo = 0.50000
termo = 0.16667
termo = 0.041667
...
termo = 0.0000027557
termo = 0.00000027557
Pela versao Crescente
O valor de e: 2.7182818011463845 ; erro_rel = 0.000000010048

formaDecrescente
Digite o valor de N: 10
termo = 0.00000027557
termo = 0.0000027557
...
termo = 0.041667
termo = 0.16667
termo = 0.50000
termo = 1
Pela versao DECrescente
O valor de e: 2.7182818011463845 ; erro_rel = 0.000000010048

--Calculando a aproximacao para N=5 --
Pela versao Crescente
O valor de e: 2.7166666666666663 ; erro_rel = 5.941848175817597e-04
Pela versao DECrescente
O valor de e: 2.7166666666666668 ; erro_rel = 5.941848175815963e-04

--Calculando a aproximacao para N=10 --
Pela versao Crescente
O valor de e: 2.7182818011463845 ; erro_rel = 1.004776631021105e-08
Pela versao DECrescente
O valor de e: 2.7182818011463845 ; erro_rel = 1.004776631021105e-08

--Calculando a aproximacao para N=20 --
Pela versao Crescente
O valor de e: 2.7182818284590455 ; erro_rel = 1.633712903499084e-16
Pela versao DECrescente
O valor de e: 2.7182818284590451 ; erro_rel = 0

--Calculando a aproximacao para N=25 --
Pela versao Crescente
O valor de e: 2.7182818284590455 ; erro_rel = 1.633712903499084e-16
Pela versao DECrescente
O valor de e: 2.7182818284590451 ; erro_rel = 0

e/f) Há erros de arredondamento e de truncamento em ambos os casos. Estes erros estão "acontecendo "ao mesmo tempo, isto é, há a contribuição dos dois erros.

Ao usar poucos termos (N pequeno) o erro de truncamento é muito maior que o erro de arredondamento e tem a maior contribuição ao erro total (praticamente a contribuição total).

À medida que se aumenta a qte de termos (quando N aumenta) o erro de trucamento cai e o erro existente é praticamente devido apenas à limitação da máquina (ao erro de arredondamento).
A partir de um certo N, não adianta usar mais termos (melhorar o método) pois estes não contribuem mais na soma (são "inócuos" à soma).

g) A ordem em que os termos são adicionados à soma vai afetar levemente o resultado.
Quando se soma termos de ordem de grandeza pequena a termos de ordem de grandeza maior, partir de um ponto estes temos não alteram mais a soma. Os dígitos dos termos de menor grandeza são muito "menos" significativos e a partir de um certo ponto não contribuirem mais à soma.
Ao começar colocando as contribuições "pequenas" inicialmente o acúmulo (a soma destes "pequenos") irá perder menos dígitos significativos já que os termos de maior ordem entram por último.

Para perceber é interessante fazer "na mão" o cálculo de e, com N=4 e com N=5, por exemplo, com as duas ordens. Por exemplo, simulando uma máquina 4 dígitos na mantissa, para N=5:

Na ordem crescente, tem-se

$e^1 \sim ((((((1/0!) + (1/1!)) + (1/2!)) + (1/3!)) + (1/4!) + (1/5!))$

Usando 4 dígitos na mantissa tem-se

$((((((1/0!) + (1/1!)) + (1/2!)) + (1/3!)) + (1/4!) + 0.008333)$

(((((1 + 1) + 0.5) + 0.1666) + 0.04166 + 0.008333)
((((2.0 + 0.5) + 0.1666) + 0.04166 + 0.008333)
(((***2.5*** ***+*** ***0.1666*** *) + 0.04166 + 0.008333)*
((***2.666*** *+ 0.04166 + 0.008333)*
(2.707 + 0.008333)
2.715

Na ordem inversa
começa se colocando a contribuição dos termos pequenos
(0.04166 + 0.008333)
….+ 0.04999

Questão 5
Código:

```
x(1)= b(1)/A(1,1);
for i = 2:1:n
   s=0;
   for j =1:1:(i-1)
     s = s + A(i,j)*x(j);
   endfor
   x(i)=( b(i) - s)/ A(i,i);
endfor
```

Ver os códigos do octave na pasta.
Há 2 versões, uma delas é uma function (rotina computacional)

Para a versão na forma de function, a execucão se faz chamando a função na linha de comandos. A matriz A e o vetor b, devem ser passados como dados de entrada (argumentos) tal como:
>> [x]=resolve_triangular_inferior(A,b);