INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
MINAS GERAIS
Campus Sabará

Cálculo Numérico
Trabalho Final - Turma: Sistemas de Informação

Professor: Carlos Alexandre Silva **Valor**: 25 pontos

1. Deverá ser feito um *benchmark* de uma série de funções-teste referentes a um conjunto de programas (diferentes linguagens de programação) em função do tempo de cálculo e o número de iterações. Utilize sempre a versão mais atual do programa. Todos os programas que serão utilizados para gerar o *benchmark* deverão ser gratuitos. Para alguns programas existem algumas bibliotecas que podem facilitar o cálculo numérico. Faça uma pesquisa pela internet para encontrar algumas dessas bibliotecas (caso existam) e as utilize. Por exemplo, as bibliotecas **LAPACK** e **BLAS** para C são úteis para fazer operações que envolvam álgebra linear. Dependendo do problema, faça o teste em C puro e com o uso da biblioteca. Utilize dois sistemas operacionais: Windows e Linux para a realização dos testes. A relação dos programas a serem utilizados é a seguinte:

   1. C++
   2. C
   3. Java
   4. Scilab
   5. Octave
   6. R
   7. Fortran
   8. Python
   9. Ruby
   10. Julia
   11. OCaml
   12. Perl
   13. Lisp
   14. Haskel
   15. LuaJIT
   16. Go

   O trabalho deve conter um breve resumo de cada programa informando algumas características como:

   1. Desenvolvedor.
   2. Ano de início.
   3. Ano da última versão.
   4. Características gerais.

   Para maiores informações sobre softwares numéricos acesse:

   1. `http://en.wikipedia.org/wiki/List_of_numerical_analysis_software`.
   2. `http://en.wikipedia.org/wiki/Category:Numerical_programming_languages`.
   3. `http://en.wikipedia.org/wiki/List_of_programming_languages_by_type`.
   4. `http://en.wikipedia.org/wiki/Comparison_of_numerical_analysis_software`.
   5. `http://en.wikipedia.org/wiki/List_of_programming_languages_by_type`.
   6. `http://en.wikipedia.org/wiki/List_of_numerical_libraries`.
   7. `http://rosettacode.org/wiki/Language_Comparison_Table`

   As funções a serem implementadas são:

1. Fibonacci recursivo.
2. Parse Int (converte uma string em um inteiro).
3. Quicksort.
4. Conjunto de Mandelbrot (Fractal).
5. Geração do π pela fórmula de Euler ($\sum_{n=1}^{\infty} 1/n^2 = \pi^2/6$)
6. Estatística em Matriz Randômica.
7. Método de Relaxação Sucessiva (SOR).
8. Método de Newton.

A relação de funções de 1 a 8 está no Anexo I, feitas em Octave. Demais detalhes das implementações em outras linguagens acesse:
`https://github.com/JuliaLang/julia/tree/master/test/perf/micro`.

O trabalho deve estar no formato do template de artigo disponível no site da SBC (Sociedade Brasileira de Computação). Todas as figuras usadas no trabalho devem estar no formato EPS. Deve ser entregue uma pasta contendo as figuras em formato JPEG e EPS e uma pasta contendo o artigo em TEX. No *benchmark* deve conter as colunas:

1. Medida em Bytes do código fonte: Utilize apenas o código fonte, removendo todos os comentários, espaços em branco duplicados, e , em seguida, aplique a compressão mínima do GZip. A medida do código é o tamanho em bytes do arquivo do código-fonte comprimido com o GZip. A extensão gerada pelo Gzip é o .gz.
2. Tempo de processamento do início ao fim da execução de cada programa. Este tempo deve ser dado em segundos. Especifique o compilador e o hardware utilizado.
3. Complexidade em função da entrada (movimentações, comparações, atribuições e operações aritmética).

Para cada algoritmo utilize algumas entradas (pequena, média e grande). Faça uma análise estatística como feita na disciplina de Probabilidade e Estatística para comparar os algoritmos. Lembre-se que alguns algoritmos são determinísticos e outros probabilísticos.

Anexo I

```
function assert(bool)
  if ~bool
    error('Assertion failed')
  end
end

%1. Fibonacci (use n>=20)
function f = fib(n)
  if n < 2
    f = n;
    return
  else
    f = fib(n-1) + fib(n-2);
  end
end

%2. Parse Int (use t>=1000)
function n = parseintperf(t)
  for i = 1:t
    n = randi([0,2^32-1],1,'uint32'); %Gera 1 número aleatório inteiro
    s = dec2hex(n); %Converte decimal para hexadecimal
    m = hex2dec(s); %Converte hexadecimal para decimal
    assert(m == n); %Verifica que m é igual a n
  end
end

%3. Quicksort (use n>=5000)
function b = qsort(a) %Função agregada do quicksort
  b = qsort_kernel(a, 1, length(a));
end

function a = qsort_kernel(a, lo, hi) %Função agregada do quicksort
  i = lo;
  j = hi;
  while i < hi
    pivot = a(floor((lo+hi)/2));   %floor significa funcao piso
    while i <= j
      while a(i) < pivot, i = i + 1; end
      while a(j) > pivot, j = j - 1; end
      if i <= j
        t = a(i);
        a(i) = a(j);
        a(j) = t;
        i = i + 1;
        j = j - 1;
      end
```

```
      end
      if lo < j; a=qsort_kernel(a, lo, j); end
      lo = i;
      j = hi;
    end
end

function v = sortperf(n) %Função principal de chamada do quicksort
    v = rand(n,1);  %Gera uma matriz nx1 de numeros aleatórios entre 0 e 1
    v = qsort(v);
end

%4. Mandelbrot
function n = mandel(z) %Função agregada do mandelperf
    n = 0;
    c = z;
    for n=0:79
      if abs(z)>2 %Se z=a+bi então abs(z)=sqrt(a^2+b^2)
        return
      end
      z = z^2+c;
    end
    n = 80;
end

function M = mandelperf(ignore) %Função principal
    M = zeros(length(-2.0:.1:0.5), length(-1:.1:1));
    count = 1;
    for r = -2:0.1:0.5
      for i = -1:.1:1
        M(count) = mandel(complex(r,i)); %complex(a,b)=a+bi
        count = count + 1;
      end
    end
end

%5. Geração de \pi
function sum = pisum(ignore) %Forma não vetorizada
    sum = 0.0;
    for j=1:500
      sum = 0.0;
      for k=1:10000
        sum = sum + 1.0/(k*k);
      end
    end
end

function s = pisumvec(ignore) %Forma vetorizada
```

```
  a = [1:10000]
  for j=1:500
    s = sum( 1./(a.^2));
  end
end

%6. Estatística em matriz aleatória (use t>=1000)
function randmatstat(t)
  n = 5;
  v = zeros(t); %Gera uma matriz txt de zeros
  w = zeros(t);
  for i = 1:t
    a = randn(n,n); %Matriz nxn de nº aleatórios com distribuição normal
    b = randn(n,n);
    c = randn(n,n);
    d = randn(n,n);
    P = [a b c d];
    Q = [a b; c d];
    v(i) = trace((P'*P)^4); %trace=Traço de matriz e P'=transposta de A
    w(i) = trace((Q'*Q)^4);
  end
  std(v)/mean(v), %desvio e média são calculados por coluna da matriz.
  std(w)/mean(w),
end

%7. Relação sucessiva
% Faça um teste para
% A=[4 -2 1 3 0;-1 10 0 8 1;-1 1 15 2 4;0 1 10 5 1; 2 -3 1 2 20];
% b=[15;56;74;57;107];w=1.6;Toler=1e-5;IterMax=500;
function [Iter,x,NormaRel]=SOR(A,b,w,Toler,IterMax)
  n=length(A);
  for i=1:n
    r=1/A(i,i);
    for j=1:n
      if (i~=j), A(i,j)=A(i,j)*r; end
    end
    b(i)=b(i)*r; x(i)=b(i);
  end
  Iter=0; NormaRel=1000;
  while (NormaRel > Toler && Iter < IterMax)
    Iter=Iter+1;
    for i=1:n
      soma=0;
      for j=1:n
        if (i~=j), soma=soma+A(i,j)*x(j); end
      end
      v(i)=x(i); x(i)=w*(b(i)-soma)+(1-w)*x(i);
    end
```

```
      NormaNum=0;  NormaDen=0;
      for  i=1:n
         t=abs(x(i)-v(i))
         if  (t>NormaNum),  NormaNum=t;  end
         if  abs(x(i))>NormaDen,  NormaDen=abs(x(i));  end
      end
      NormaRel=NormaNum/NormaDen;  Iter ,x,NormaRel,
   end
   if  NormaRel <=  Toler
      CondErro  =0;
   else
      CondErro=1;
   end
end


%8. Método de Newton
%Faça um teste com x0=4;Toler=1.0000e-05;IterMax=100;
%function [f]=f(x)
%f=x^4+2*x^3-13*x^2-14*x+24;
%end
%
%function [f]=df(x)
%f=4*x^3+6*x^2-26*x-14;
%end

function [Raiz,Iter,CondErro]=Newton(x0,Toler,IterMax)
%Saida: Raiz é a raiz da funcao;
%           Iter é a quantidade de itearacoes feitas;
%           CondErro=0 se a raiz foi encontrada e
%           CondErro=1 se nao for encontrada
   Fx=f(x0); DFx=df(x0);x=x0;Iter=0; %f=função e df=derivada de f
   DeltaX=1000;
   DF=1;
   while ( (abs(DeltaX)>Toler || abs(Fx)>Toler) && (DF~=0) && (Iter <IterMax)
      DeltaX=-Fx/DFx,x=x+DeltaX;Fx=f(x);DFx=df(x);Iter=Iter+1;
   end
   Raiz=x;
   if abs(DeltaX)<=Toler && abs(Fx)<=Toler
      CondErro=0;
   else
      CondErro=1;
   end
end
```