ANEXO 1: PROCEDIMENTOS QUE CALCULAM O MULTIPLICADOR E O INCREMENTO PARA O MÉTODO DE CONGRUÊNCIA LINEAR E UM GERADOR DE NÚMEROS RANDÔMICOS CRIADO A PARTIR DESSE MÉTODO

A. Programa que calcula os multiplicadores para o método de congruência linear

O procedimento abaixo calcula os multiplicadores para o método de congruência linear, obedecendo às condições do Teorema acima. Observamos que podem ser calculados com a seguinte regra. Seja

$$m = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_s^{\alpha_s}$$

a decomposição canônica de m em fatores primos. Definimos $M = p_1 p_2 \ldots p_s$ se 4 não divide m, e $M = 4 p_2 \ldots p_s$ se 4 divide m com $p_1 = 2$. Então os multiplicadores são os números da forma $a = k M + 1$, onde k = 0, 1, 2, 3, 4, ...

Sintaxe:

O nosso procedimento tem a seguinte sintaxe:

multiplicador (m , n)

Onde *m* = módulo do gerador de números pseudo-aleatórios
n = um intervalo de números inteiros positivos ou um número inteiro positivo.

O argumento *n* é opcional. Se o usuário não fornecê-lo, o procedimento retorna todos os possíveis multiplicadores para o gerador. Se o usuário digitar um número, o procedimento retorna todos os multiplicadores que estão compreendidos entre 1 e esse número e se o usuário digitar um intervalo, o procedimento retorna os multiplicadores que estão no intervalo especificado.

Procedimento multiplicador

```
> multiplicador:= proc ( m, x::anything)
```

```
> local a,b,aux,i,t,p,M,h,u;
> if nargs=0 then ERROR(`Você deve digitar pelo menos um
  argumento no procedimento multiplicador(). O primeiro
  argumento é o módulo do gerador e o segundo argumento, que
  é opcional, é um número inteiro positivo ou um intervalo
  de inteiros positivos menores do que o módulo`)
> elif not type(m,posint) or m<=1 then ERROR(`O módulo deve
  ser um inteiro maior ou igual a 2`)
> elif nargs=2 then
>   if not typematch(x,{b::posint,a::posint..b::posint})
  then
>    ERROR(`O segundo argumento deve ser um inteiro positivo
  ou um intervalo de inteiros positivos, mas recebeu `, x)
>   fi;
>   if assigned(a) then
>   if b<a then ERROR(`O segundo argumento deve ser um
  intervalo não vazio de inteiros positivos, mas recebeu o
  valor `.a.`..`.b);
>   fi;
>   fi;
>   if b>=m then ERROR(`O segundo argumento deve ser um
  número menor que o módulo ou um intervalo de números que
  estejam entre [ 1, `.m.` [`);
>   fi:
> elif nargs>2 then ERROR( `Você deve digitar no máximo dois
  argumentos no procedimento multiplicador(). O primeiro
  argumento é o módulo (inteiro maior ou igual a 2) e o
  segundo, que é opcional, é um inteiro positivo ou um
  intervalo de inteiros positivos, menores do que o
  módulo`);
> fi:
> aux := proc (w,z,r)
>     local j,i,M,k,k1,k2,h;
>     global A_,cont;
> M:=r;
> k1 := ceil (( w - 1 ) / M );
> k2 := floor (( z - 1 ) / M );
> if k2<k1 then print(`Não existem multiplicadores no
  intervalo dado`);
> else
> A_ := array ( 1 .. k2-k1+1 );
> i := 1;
> for k from k1 to k2 do
>  A_[i] := k * M + 1;
>  i := i + 1;
> od:
```

```
>       print(`Os multiplicadores são números da forma
  `.M.`k+1  onde k=`.k1.`, ..., `.k2);
>       print(`Foram calculados  `.(k2-k1+1).`
  multiplicadores que estão estocados na lista A_ .`);
>   while h<>s and h<>n do
>    h:=readstat(`Deseja ver a lista de multiplicadores?
  [s;/n;] : `);
>     if h=s then RETURN(eval(A_)) fi;
>     if h<>s and h<>n then print(`Você deve digitar   s;
  ou   n; , tente novamente`) fi;
>   od:
> fi;
> end:
> readlib(ifactors);
> t := ifactors ( m ) ;
> for i to nops(t[2]) do
>  p[i] := t[2,i,1];
> od:
> M := product ( p[u], u=1..nops(t[2]) );
>  if m mod 4 = 0 then M:=2*M fi;
>   if nargs=1 then aux(1,m,M)
>   elif nargs=2 and not assigned(a) then aux(1,b,M)
>   elif nargs=2 and assigned(a) then aux(a,b,M)
>   fi;
> end:
```

```
>
```

B. Programa que calcula os incrementos para o método de congruência linear

O procedimento abaixo calcula os incrementos para o método de congruência linear de acordo com a condição (i) do Teorema 1. Observamos que é válida a seguinte regra. Seja

$$m = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_s^{\alpha_s}$$

a decomposição canônica de m em fatores primos. Definimos $M = p_1 p_2 \ldots p_s$. Então os incrementos são os números da forma $c = k M + r$, onde k = 0,1, 2, 3, 4, ... e r está no conjunto dos números inteiros do intervalo $[1,M)$ que são relativamente primos com M.

Sintaxe:

O nosso procedimento tem a seguinte sintaxe:

incremento (m , n)

Onde *m* = módulo do gerador de números pseudo-aleatórios
n = um intervalo de números positivos inteiros ou um número positivo inteiro

O argumento *n* é opcional. Se o usuário não fornecê-lo, o procedimento retorna todos os possíveis incrementos para o gerador. Se o usuário digitar um número, o procedimento retorna todos os incrementos que estão compreendidos entre 1 e esse número, e se o usuário digitar um intervalo, o nosso procedimento retorna os incrementos que estão no intervalo especificado.

Procedimento incremento

```
> incremento:=proc ( m, x::anything)
> local a,b,aux,i,t,p,M,h;
> global C_,tam;
> if nargs=0 then ERROR(`Você deve digitar pelo menos um
  argumento no procedimento incremento(). O primeiro
  argumento é o módulo do gerador e o segundo argumento, que
  é opcional, é um número inteiro positivo ou um intervalo
  de inteiros positivos menores do que o módulo`)
> elif not type(m,posint) or m<=1 then ERROR(`O módulo deve
  ser um inteiro maior ou igual a 2`)
> elif nargs=2 then
>  if not typematch(x, {b::posint,a::posint..b::posint})
  then ERROR(`O segundo argumento deve ser um inteiro
  positivo ou um intervalo de inteiros positivos menores do
  que o módulo `);
>  fi;
>  if assigned(a) then if a>b then ERROR(`Você deve digitar
  um intervalo não vazio`) fi fi;
>  if b>=m then ERROR(`O segundo argumento deve ser um
  número menor que o módulo ou um intervalo de números que
  esteja entre [ 1, `.m.` [`);
> fi;
> elif nargs>2 then ERROR(`Você deve digitar no máximo dois
  argumentos no procedimento incremento(). O primeiro
  argumento é o módulo e o segundo, que é opcional, é um
  número inteiro positivo ou um intervalo de números
  inteiros positivos menores do que o módulo.`)
> fi:
> aux := proc(w1,z1,g)
> local w,z,M,i,j,v,r,k,j1,j2,k1,k2,tr,h;
> global C_,tam;
```

```
> w:=w1;
> z:=z1;
> M:=g;
> j:=1;
> for i to M-1 do
>  if igcd(i,M)=1 then
>   r[j]:=i;
>   j:=j+1;
>  fi;
> od;
>
> tr:=j-1;
> j:=1;
> while igcd(w,M)<>1 do w:=w+1 od;
> k1:=floor(w/M);
> while (k1*M+r[j])<>w do j:=j+1 od;
> j1:=j;
> j:=1;
> while (igcd(z,M)<>1) do z:=z-1 od;
> k2:=floor(z/M);
> while (k2*M+r[j])<>z do j:=j+1 od;
> j2:=j;
> i:=1;
> v:=tr;
> C_:=array(1..tr*(k2-k1-1)+j2+tr-j1+1);
> tam:=tr*(k2-k1-1)+j2+tr-j1+1;
> if tam=0 then print(`Não existem incrementos no intervalo
  dado`);
> else
>  print(`Os incrementos são números da forma `.M.`k+r onde
  k = `.k1.` .. `.k2.` e r é relativamente primo com `.M.`
  1<=r e r<M`);
> print(`Foram calculados  `.tam.`  incrementos que estão
  estocados na lista C_ .`);
> for k1 from k1 to k2 do
>   if k1=k2 then v:=j2 fi:
>   for j1 from j1 to v do
>     C_[i]:=k1*M+r[j1];
>   i:=i+1;
>  od:
>  j1:=1;
> od;
> while h<>n and h<>s do
>  h:=readstat(`Deseja ve lista de incrementos? [s;/n;] :
```

O conjunto R=sistema reduzido de resíduos (SRR) módulo M

tamanho do vetor $r = phi(M)$

calcula o1º número que satisfaz o teorema

encontra o valor do 1º k

encontra o valor do 1º r

calcula o último número que satisfaz o teorema

encontra o valor do último k

encontra o valor do último r

```
    `);
>   if h=s then RETURN(eval(C_)) fi;
>   if h<>s and h<>n then print(`Você deve digitar s; ou n;,
    tente novamente`) fi
> od;
> fi;
> end;
> readlib(ifactors);
> t:=ifactors(m);
> for i to nops(t[2]) do
>  p[i]:=t[2,i,1];
> od:
> M:=product(p[u],u=1..nops(t[2]));
> if nargs=1 then
> aux(1,m,M);
> elif nargs=2 then
>   if assigned(a) then
>    aux(a,b,M);
>   else
>    aux(1,b,M);
>   fi;
> fi;
> end:
>
```

C. Gerador de números pseudo-randômicos que se utiliza do Método de Congruência Linear

O gerador de números randômicos é definido como um procedimento, e dado o módulo pelo usuário e o incremento e o multiplicador calculados pelos procedimentos acima, podemos gerar números pseudo-randômicos inteiros entre zero e o módulo, sendo que a semente é calculada pela hora do sistema se não foi atribuída pela função randsem.

A sintaxe do gerador é:

random (m, a, c)

onde m é o módulo fornecido pelo usuário
a é o multiplicador
c é o incremento

Gerador

```
> random:=proc(m::posint,a::nonnegint,c::nonnegint)
> local RANDOMp;
> global _m1,_rsem;
> RANDOMp:=proc(x,y,z)
> local _modulo,_multip,_increm,aux;
> global _sement,_rsem,_m1;
> _modulo:=x;
> _multip:=y;
> _increm:=z;
>
> if not(assigned(_sement)) then
>  _sement:=modp(iolib(25),_modulo);
> fi:
>   _rsem:=`false`;
>   _m1:=_modulo;
>   _sement:=(_multip*_sement+_increm) mod (_modulo);
>   RETURN(_sement);
> end:
>
> if a>=m then ERROR(`O multiplicador fora do intervalo
  [0,módulo[`);
> elif c>=m then ERROR(`O incremento fora do intervalo
  [0,módulo[`);
> elif nargs>3 then ERROR(`Random só aceita 3 argumentos`);
> elif (_rsem=`true`) and (nargs>=0) and (_sement>=m) then
  ERROR(`A semente escolhida está fora do intervalo
  [0,módulo[`);
> else
> RANDOMp(m,a,c);
> fi:
> end:
```

iolib(25) fornece a hora do sistema

>

D. Programa que troca a semente do gerador gerador de números randômicos

O programa abaixo troca a semente do nosso gerador de números pseudo-randômicos. A semente do gerador de números randômicos, quando não atribuída pelo referido procedimento, é calculada a partir da hora do sistema módulo *m*.

A sintaxe do nosso procedimento é:

randsem (x);

onde *x* é a semente fornecida pelo usuário, que deve ser do tipo inteira não negativa.

Procedimento randsem

```
> randsem:=proc(x::nonnegint)
> global _sement,_rsem;
> if nargs>1 then ERROR(`Você deve digitar um argumento`);
> else
>   _sement:=x;
>   _rsem:=`true`;
>   RETURN(`semente = `._sement);
> fi;
> end:
>
```

EXEMPLOS:

Vamos gerar uma seqüência de 100 números pseudo-randômicos a partir dos procedimentos ja vistos.

Vamos usar o módulo $m = 100000$ e calcular os multiplicadores.

```
> multiplicador(100000,45550..48000);
```

Os multiplicadores são números da forma 20k+1 onde k=2278, ..., 2399

Foram calculados 122 multiplicadores que estão estocados na lista _A .

```
Deseja ver a lista de multiplicadores? [s;/n;] : s;
```

[45561, 45581, 45601, 45621, 45641, 45661, 45681, 45701, 45721, 45741, 45761, 45781, 45801, 45821, 45841, 45861, 45881, 45901, 45921, 45941, 45961, 45981, 46001, 46021, 46041, 46061, 46081, 46101, 46121, 46141, 46161, 46181, 46201, 46221, 46241, 46261, 46281, 46301, 46321, 46341, 46361, 46381, 46401, 46421, 46441, 46461, 46481, 46501, 46521, 46541, 46561, 46581, 46601, 46621, 46641, 46661, 46681, 46701, 46721, 46741, 46761, 46781, 46801, 46821, 46841, 46861, 46881, 46901, 46921, 46941, 46961, 46981, 47001, 47021, 47041, 47061, 47081, 47101, 47121, 47141, 47161, 47181, 47201, 47221, 47241, 47261, 47281, 47301, 47321, 47341, 47361, 47381, 47401, 47421, 47441, 47461, 47481, 47501, 47521, 47541, 47561, 47581, 47601, 47621, 47641, 47661, 47681, 47701, 47721, 47741, 47761, 47781, 47801, 47821, 47841, 47861, 47881, 47901, 47921, 47941, 47961, 47981]

Da mesma forma, vamos calcular os incrementos para o gerador.

```
> incremento(100000,3250..3500);
```

Os incrementos são números da forma 10k+r onde k = 325 .. 349 e r, positivo inteiro, são\ os números menores que 10 que satisfazem a seguinte propriedade: r é relativamente pri\ mo com 10

Foram calculados 100 incrementos que estão estocados na lista _C .

```
Deseja ver a lista de incrementos? [s;/n;] : s;
```

[3251, 3253, 3257, 3259, 3261, 3263, 3267, 3269, 3271, 3273, 3277, 3279, 3281, 3283, 3287, 3289, 3291, 3293, 3297, 3299, 3301, 3303, 3307, 3309, 3311, 3313, 3317, 3319, 3321, 3323, 3327, 3329, 3331, 3333, 3337, 3339, 3341, 3343, 3347, 3349, 3351, 3353, 3357, 3359, 3361, 3363, 3367, 3369, 3371, 3373, 3377, 3379, 3381, 3383, 3387, 3389, 3391, 3393, 3397, 3399, 3401, 3403, 3407, 3409, 3411, 3413, 3417, 3419, 3421, 3423, 3427, 3429, 3431, 3433, 3437, 3439, 3441, 3443, 3447, 3449, 3451, 3453, 3457, 3459, 3461, 3463, 3467, 3469, 3471, 3473, 3477, 3479, 3481, 3483, 3487, 3489, 3491, 3493, 3497, 3499]

Agora, vamos escolher um multiplicador e um incremento.

Foram escolhidos o multiplicador 46581 e o incremento 3407. Agora o usuário pode escolher a semente se quiser. Vamos escolhe-la:

```
> randsem(89706);
```

semente = 89706

Vamos gerar os 100 números randômicos a partir dos dados acima.

```
> H:=array(1..100):
> for i to 100 do
> H[i]:=random(100000,46581,3407):
> od:
> evalm(H);
```

Abaixo estão os 100 números calculados.

[98593, 63940, 92547, 35214, 6741, 5928, 35575, 22482, 37449, 15276, 74763, 38710, 53917, 11184, 65311, 55098, 23345, 36852, 6419, 6846, 96933, 39480, 21287, 73154, 89881, 50268, 37115, 57222, 61389, 64416, 65103, 66250, 94657, 21124, 80451, 91438, 76885, 83592, 2359, 87986, 79273, 19020, 74027, 55094, 37021, 78608, 42655, 15962, 29329, 77556, 39443, 97790, 59397, 75064, 59591, 11778, 34425, 54332, 42299, 33126, 45613, 2560, 50767, 81034, 48161, 90948, 52195, 98702, 41269, 54696, 97783, 33330, 48137, 73004, 2731, 16118, 95965, 49072, 26239, 42266, 95953, 90100, 51507, 50974, 23301, 87288, 65735, 5442, 97209, 95836, 40123, 72870, 60877, 14944, 9871, 4458, 61505, 67812, 54179, 15406]

Se você precisar de números entre zero e um inclusos, por exemplo, basta dividir o número encontrado pelo módulo do gerador.

```
> for i to 100 do
> H[i]:=H[i]/(100000-1):
```

```
> od:
> evalf(evalm(H));
```

Os números abaixo são números entre [0 , 1]

```
[.9859398594, .6394063941, .9254792548, .3521435214, .06741067411, .05928059281, .3557535575,
    .2248222482, .3744937449, .1527615276, .7476374764, .3871038710, .5391753918, .1118411184,
    .6531165312, .5509855099, .2334523345, .3685236852, .06419064191, .06846068461, .9693396934,
    .3948039480, .2128721287, .7315473155, .8988189882, .5026850269, .3711537115, .5722257223,
    .6138961390, .6441664417, .6510365104, .6625066251, .9465794658, .2112421124, .8045180452,
    .9143891439, .7688576886, .8359283593, .02359023590, .8798687987, .7927379274, .1902019020,
    .7402774028, .5509455095, .3702137021, .7860878609, .4265542655, .1596215962, .2932929329,
    .7755677557, .3944339443, .9779097791, .5939759398, .7506475065, .5959159592, .1177811778,
    .3442534425, .5433254333, .4229942299, .3312633126, .4561345613, .02560025600, .5076750768,
    .8103481035, .4816148161, .9094890949, .5219552196, .9870298703, .4126941269, .5469654697,
    .9778397784, .3333033330, .4813748137, .7300473005, .02731027310, .1611816118, .9596595966,
    .4907249072, .2623926239, .4226642266, .9595395954, .9010090101, .5150751508, .5097450975,
    .2330123301, .8728887289, .6573565736, .05442054421, .9720997210, .9583695837, .4012340123,
    .7287072871, .6087760878, .1494414944, .09871098711, .04458044580, .6150561506, .6781267813,
    .5417954180, .1540615406]
>
```

BIBLIOGRAFIA

KNUTH, Donald E. , The Art of Computer Programing, Volume 2: Seminumerical Algorithms. 2 impressão. Reading, Addison-Wesley Publishing Company, 1971.

REDFERN, E. J., Introduction to Pascal for Computational Mathematics.

TOCHER, K. D. , The Art of Simulation, 1ª impressão. Unibooks, Hodder and Stoughton, 1963.

NIEVERGELT, Jurg; FARRER, J. Craig; REINGOLT, Edward M. , Computer Approaches to Mathematical Problems. Englewood Cliffs, Prentice-Hall, 1974.

BANKS, Jerry; CARSON, John S., Discrete-Event System Simulation. Englewood Cliffs, Prentice-Hall, New Jersey, 1984.


```
>
```