Project 997 PRNGs Part 4
MRG32a Variant Algorithms
By
Namir C. Shammas

1/ Introduction

This study looks at two flavors of the MRG32a algorithm variants. The difference between the two variants is the modulia values used.

2/ MRG32a Algorithms (Version MRG32a PSO)

The MRG32A algorithm is defined as:

```
 % initialization of arrays ix() and iy()
 M1 = 2^32 - 209
 M2 = 2^32 - 22853
 ix = 1+fix((M2-2)*rand(1,4))
 iy = 1+fix((M2-2)*rand(1,4))

[r,ix,iy]  = function MRG32A(ix,iy,a11, a12, a13,a21, a22,a23)
% ix() and iy() are arrays with 4  elements and provide the function
% with an input of k elements each.
   M1 = 2^32 - 209
   M2 = 2^32 - 22853
   ix = 1+fix((M2-2)*rand(1,4))
   iy = 1+fix((M2-2)*rand(1,4))
   ix(4) = a11*ix(3)+a12*ix(2)+a13*ix(1) mod M1
   iy(4) = a21*iy(3)+a22*iy(2)+a23*iy(1) mod M2
   iz = ix(4) - iy(4) mod M1
   if iz <= 0, iz = iz + M1
   r = (iz+1)/(M1+1)
   ix(1:3) = ix(2:4)
   iy(1:3) = iy(2:4)                                    (2.1)
```


end

This section looks at a better version of the MRG32a algorithms that uses a11, a12, a13, a21, a22, a23, and arrays ix and iy that appear in equation 2.1. The calculations are all based on the value of M1 = 2^32 – 209 and M2 = 2^32 - 22853. The variable r is the sought uniform random value in the range (0, 1].

The approache that estimates the best coefficients for a MRG32a variant uses the following approach:

1. Optimize the penalty factor (see Appendix of Project 997 PRNGs Part 1) using particle swarm optimization algorithms. This optimization starts with a wide trust region and narrows it down.
2. Optimize the penalty using the narrowed optimum regions obtained in step 1. This step yields refined trust regions.
3. Perform a large run of generating random numbers using the refined trust regions from step 2. The calculations yield the statistics for the penalty factor. The upper confidence values for the mean penlty factor are the values we look at to determine the fitness of the algorithm.

The listing for do.m, which triggers the calculations for the first and second optimization phases is:

```
%
% Note: Uncomment one call to doAll() at a time!
%

% ----------------------- Wide Range Optimization

maxElems=10000;
maxIters=100;
lb=[0 1000 -2000000 1000 0 -2000000];
ub=[1 2000000 0 2000000 1 0];
sFilename='res1.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1 = 2^32 - 209;
xM2 = 2^32 - 22853;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM1
,xM2,bRound,nDigits)
```



```
maxElems=10000;
maxIters=100;
lb=[0 1000 -2000000 1000 0 -2000000];
ub=[100 2000000 0 2000000 100 0];
sFilename='res2.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1 = 2^32 - 209;
xM2 = 2^32 - 22853;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM1
,xM2,bRound,nDigits)

maxElems=10000;
maxIters=100;
lb=[0 1000 -2000000 1000 0 -2000000];
ub=[1000 2000000 0 2000000 1000 0];
sFilename='res3.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1 = 2^32 - 209;
xM2 = 2^32 - 22853;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM1
,xM2,bRound,nDigits)

maxElems=10000;
maxIters=100;
lb=[0 10000 -2000000 10000 0 -2000000];
ub=[1 2000000 0 2000000 1 0];
sFilename='res4.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1 = 2^32 - 209;
xM2 = 2^32 - 22853;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM1
,xM2,bRound,nDigits)

maxElems=10000;
maxIters=100;
lb=[0 10000 -2000000 10000 0 -2000000];
ub=[100 2000000 0 2000000 100 0];
sFilename='res5.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1 = 2^32 - 209;
```



```
xM2 = 2^32 - 22853;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM1
,xM2,bRound,nDigits)

maxElems=10000;
maxIters=100;
lb=[0 10000 -2000000 10000 0 -2000000];
ub=[1000 2000000 0 2000000 1000 0];
sFilename='res6.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1 = 2^32 - 209;
xM2 = 2^32 - 22853;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM1
,xM2,bRound,nDigits)

% ------------------------------------------------------------------------

% ---------------------- Narrow Range Optimization

r = 0.15;
rp = 1+r;
rm = 1-r;
maxElems=10000;
maxIters=100;
x = [0,1819719,-367294,550619,0,-1132162];
lb = round(rm*x,0);
ub = round(rp*x,0);
ub(1) = 3;
ub(5) = 3;
[lb,ub] = checkRange(lb,ub);
sFilename='res1b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1 = 2^32 - 209;
xM2 = 2^32 - 22853;
bRound=true;
nDigits = 10;
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM1
,xM2,bRound,nDigits)

maxElems=10000;
maxIters=100;
x = [55,1747366,-1491591,1614121,89,-957708];
lb = round(rm*x,0);
ub = round(rp*x,0);
[lb,ub] = checkRange(lb,ub);
sFilename='res2b.xlsb';
sSheetName='Sheet1';
```



```
POSpopsize=250;
POSmaxIters=350;
xM1 = 2^32 - 209;
xM2 = 2^32 - 22853;
bRound=true;
nDigits = 10;
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM1
,xM2,bRound,nDigits)

maxElems=10000;
maxIters=100;
x = [0,1997072,-24094,163844,447,-2000000];
lb = round(rm*x,0);
ub = round(rp*x,0);
[lb,ub] = checkRange(lb,ub);
ub(1) = 3;
sFilename='res3b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1 = 2^32 - 209;
xM2 = 2^32 - 22853;
bRound=true;
nDigits = 10;
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM1
,xM2,bRound,nDigits)

maxElems=10000;
maxIters=100;
x = [1,1054884,-1665985,1820628,0,-2000000];
lb = round(rm*x,0);
ub = round(rp*x,0);
lb(1) = 0;
ub(1) = 3;
lb(5) = 0;
ub(5) = 3;
[lb,ub] = checkRange(lb,ub);
sFilename='res4b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1 = 2^32 - 209;
xM2 = 2^32 - 22853;
bRound=true;
nDigits = 10;
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM1
,xM2,bRound,nDigits)

maxElems=10000;
maxIters=100;
x = [36,445951,-1257764,719268,77,-765093];
lb = round(rm*x,0);
ub = round(rp*x,0);
[lb,ub] = checkRange(lb,ub);
sFilename='res5b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
```



```
POSmaxIters=350;
xM1 = 2^32 - 209;
xM2 = 2^32 - 22853;
bRound=true;
nDigits = 10;
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM1
,xM2,bRound,nDigits)

maxElems=10000;
maxIters=100;
x = [365,2000000,-1435192,1330572,610,-1096141];
lb = round(rm*x,0);
ub = round(rp*x,0);
[lb,ub] = checkRange(lb,ub);
sFilename='res6b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1 = 2^32 - 209;
xM2 = 2^32 - 22853;
bRound=true;
nDigits = 10;
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM1
,xM2,bRound,nDigits)

system('shutdown /s')
```

Listing 2.1. The listing of file do.m.

Listing 2.2 shows the source code for file doAll.m. The function doAll() optimizes the function rng997Gen1() using the Particle Swarm Optimization (PSO) method by calling function particleswarm(). The function do() calls function doAll() for the first and second optimization phases.

```
function
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM1
,xM2,bRound,nDigits)
global gmaxElems
global bestFactor
global M1
global M2
global doRounding
global numDigits
global bestFactor
global bestXYrnd

gmaxElems = maxElems;
doRounding = bRound;
numDigits = nDigits;
M1 = xM1;
M2 = xM2;
bResetResMat = true;
% output file exists?
if isfile(sFilename)
```



```
  resMat = readmatrix(sFilename,"Sheet", sSheetName,"Range","A2:M251");
  [nrows,~] = size(resMat);
  if nrows > 0
    for nStart=1:nrows
      if resMat(nStart,1)>1e+99, break; end
    end
    if nStart <= nrows, bResetResMat = false; end
  end
end

if bResetResMat
  resMat=2e+99+zeros(maxIters,13);
  nStart = 1;
end
options =
optimoptions('particleswarm','SwarmSize',POSpopsize,'Display','off','MaxItera
tions',POSmaxIters,'FunctionTolerance',0.01);

for i=nStart:maxIters
  bestFactor = 1e+99;
  x = particleswarm(@rng997Gen1,length(lb),lb,ub,options);
  x = round(x, 0);
  resMat(i,1) = bestFactor;
  resMat(i,2:7) = x;
  resMat(i,8:13) = bestXYrnd;

  fprintf("itr = %i, Factor = %f, X= [", i , bestFactor);
  fprintf("%i, ", x);
  fprintf("]\n");

  resMat = sortrows(resMat,1);

  T1 = array2table(resMat);
  T1.Properties.VariableNames(1:13) = {'Factor', 'C11','C12', 'C13' ,'C21',
'C22' ,'C23', ...
      'IX1' 'IX2', 'IX3', 'IY1', 'IY2', 'IY3'};
  writetable(T1, sFilename, "Sheet", sSheetName);
end

fprintf("\n\n");
resMat = sortrows(resMat,1);

beep on;
for i=1:3
  beep;
  pause(1);
end

for i =1:6
  bestXYrnd(1) = resMat(1,7+i);
end

fprintf("Entire result matrix written to file %s\n",  sFilename)
```



```
T1 = array2table(resMat);
T1.Properties.VariableNames(1:13) = {'Factor', 'C11','C12', 'C13' ,'C21',
'C22' ,'C23', ...
     'IX1' 'IX2', 'IX3', 'IY1', 'Y2', 'Y3'};
writetable(T1, sFilename, "Sheet", sSheetName);
c = cell(26,2);
c(1,1:2) = {'Max Elements', maxElems};
c(2,1:2) = {'Max Iters', maxIters};
c(3,1:2) = {'a11Low', lb(1)};
c(4,1:2) = {'a11Hi', ub(1)};
c(5,1:2) = {'a12Low', lb(2)};
c(6,1:2) = {'a12Hi', ub(2)};
c(7,1:2) = {'a13Low', lb(3)};
c(8,1:2) = {'a13Hii', ub(3)};
c(9,1:2) = {'a21Low', lb(4)};
c(10,1:2) = {'a21Hi', ub(4)};
c(11,1:2) = {'a22Low', lb(5)};
c(12,1:2) = {'a22Hi', ub(5)};
c(13,1:2) = {'a23Low', lb(6)};
c(14,1:2) = {'a23Hi', ub(6)};
c(15,1:2) = {'ix1', bestXYrnd(1)};
c(16,1:2) = {'ix2', bestXYrnd(2)};
c(17,1:2) = {'ix3', bestXYrnd(3)};
c(18,1:2) = {'iy1', bestXYrnd(4)};
c(19,1:2) = {'iy2', bestXYrnd(5)};
c(20,1:2) = {'iy3', bestXYrnd(6)};

c(21,1:2) = {'PopSize', POSpopsize};
c(22,1:2) = {'PopMaxIters', POSmaxIters};
c(23,1:2) = {'M1', M1};
c(24,1:2) = {'M2', M2};
if doRounding
  c(25,1:2) = {'Rounded?', "True"};
  c(26,1:2) = {'Rounded', nDigits};
else
  c(25,1:2) = {'Rounded?', "False"};
  c(26,1:2) = {'Rounded', "N/A"};
end
T2 = cell2table(c);
writetable(T2, sFilename, "Sheet", "Params");
fprintf("-------------------------------------------------------\n\n");
end
```

Listing 2.2. The listing of file doAll.m.

Listing 2.3 shows the listing for Reg997Gen1.m.

```
function factor = rng997Gen1(c)
%UNTITLED2 Summary of this function goes here
  global gmaxElems
  global M1
  global M2
  global doRounding
```



```
  global numDigits
  global bestFactor
  global bestXYrnd

  maxElems = gmaxElems;
  rng('shuffle','twister');
  c = round(c, 0);
  a11 = c(1);
  a12 = c(2);
  a13 = c(3);
  a21 = c(4);
  a22 = c(5);
  a23 = c(6);
  ix = 1+fix((M2-2)*rand(1,4));
  iy = 1+fix((M2-2)*rand(1,4));
  xyRnd = [ix(1:3) iy(1:3)];
  x=zeros(maxElems,1);
  for i=1:maxElems
    ix(4) = mod(a11*ix(3)+a12*ix(2)+a13*ix(1),M1);
    iy(4) = mod(a21*iy(3)+a22*iy(2)+a23*iy(1),M2);
    iz = mod(ix(4) - iy(4),M1);
    if iz <= 0, iz = iz + M1; end
    x(i) = (iz+1)/(M1+1);
    ix(1:3) = ix(2:4);
    iy(1:3) = iy(2:4);
  end
  if doRounding, x = round(x,numDigits); end
  factor=calcFactor(x,false);
  if isnan(factor), factor=65535; end
  if factor < bestFactor
    bestFactor = factor;
    bestXYrnd = xyRnd;
  end
end

function x = frac(x)
  x=mod(x,1);
end

function factor = calcFactor(x, bShowResults)
% Calculate the factor statistic for the array of random nnumbers x.

  if nargin < 2, bShowResults = false; end
  maxElems=length(x);
  meanx=mean(x);
  sdevx=std(x);
  % get the first 100 autocorrelation values
  acArr=autocorrArr(x,1,100);
  % calculate the chisquare for the 10-bin histogram
  numBins=10;
  expval=maxElems/numBins;
  [N1,ev1]=histcounts(x,numBins);
  chiSq10=sum((N1-expval).^2/expval);
  numBins=20;
  expval=maxElems/numBins;
  [N2,ev2]=histcounts(x,numBins);
  chiSq20=sum((N2-expval).^2/expval);
```



```
  numBins=20;
  [N3,ev3]=histcounts(acArr,numBins);
  ev3c=ev3(2:length(ev3));
  autoCorrSum = sum(dot(N3,abs(ev3c)));
  chsStat=chs(x);
  [Kplus,Kminus]=KStest(x);
  factor = 1000*(abs(meanx-0.5)+abs(sdevx-1/sqrt(12)))+100*(max(acArr)-
min(acArr))+100*autoCorrSum+chiSq10+chiSq20/2;
  factor = factor + 10*chsStat + 10*(Kplus + Kminus);
  if bShowResults
    fprintf('Mean = %g\nSdev = %g\n', meanx, sdevx);
    fprintf('Min = %g\nMax = %g\n', min(x), max(x));
    fprintf('Max lags = 100\n');
    fprintf('Auto correlation array\n');
    disp(acArr');
    fprintf('10-Bin Histogram\n');
    disp(N1); disp(ev1);
    fprintf('Chi-Sqr10 = %g\n', chiSq10);
    fprintf('20-Bin Histogram\n');
    disp(N2); disp(ev2);
    fprintf('Chi-Sqr20 = %g\n', chiSq20);
    fprintf('20-Bin Autocorrelation Histogram\n');
    disp(N3); disp(ev3);
    fprintf('Sum autocorrel product = %g\n', autoCorrSum);
    fprintf('Change of sign stat = %g\n', chsStat);
    fprintf('K+ = %g and K- = %g\n', Kplus, Kminus);
    fprintf('Factor = %g\n', factor);
  end
end

function acArr=autocorrArr(xdata,fromLag,toLag)

numLags=toLag-fromLag+1;
acArr=zeros(numLags,1);
j=1;
for i=fromLag:toLag
  acArr(j)=autocor(xdata,i);
  j=j+1;
end
end

function res = autocor(xdata,lag)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
maxElems=length(xdata);
res=corrcoef(xdata(1:maxElems-lag),xdata(lag+1:maxElems));
res=res(1,2);
end

function sumx=chs(x)
% Function CHS calculates the change of sign (between subsequent random
% numbers) moment. The function counts the number of consequtive positive
% abd negative changes of sign. The last nested loop calculates the
% statistic returned by this function. This value is the sum of:
%
%  sum = sum of difference(count,:) * count / difference(1,:)
%
```



```
% Keeping in mind that difference(1,:) is a good value that counts the
% sign flips that happpens one neighbor down. The values for
% difference(n,:) for n>1 are not desirable. The smaller, the better. The
% value difference(2,:) is the number of sign flips that occur
% two neighbors down. The value difference(3,:) is the number of sign flips
% that occur three neighbors down, and so on.

  n=length(x);
  nby2=fix(n/2);
  Diff=zeros(nby2,2);
  countPos=0;
  countNeg=0;
  s1=sign(x(2)-x(1));
  if s1>0
    bIsPos=true;
    countPos=1;
  else
    bIsPos=false;
    countNeg=1;
  end

  for i=3:n
    s2=sign(x(i)-x(i-1));
    % was positive and is still positive
    if s2>0 && bIsPos
      countPos=countPos+1;
    % was negative and is now positive
    elseif s2>0 && ~bIsPos
      bIsPos=true;
      countPos=1;
      Diff(countNeg,2)=Diff(countNeg,2)+1;
      countNeg=0;
    % was negative and is still negative
    elseif s2<0 && ~bIsPos
      countNeg=countNeg+1;
    % was positive is and is now negative
    elseif s2<0 && bIsPos
      bIsPos=false;
      countNeg=1;
      Diff(countPos,1)=Diff(countPos,1)+1;
      countPos=0;
    end
  end

  if s2>0
    if countPos>0, Diff(countPos,1)=Diff(countPos,1)+1; end
  else
    if countNeg>0, Diff(countNeg,2)=Diff(countNeg,2)+1; end
  end

  i=2:nby2;
  d=Diff(2:nby2,:);
  sumx=0;
  for j=1:2
    sumx = sumx + dot(d(:,j),i)/Diff(1,j);
  end
end
```



```
function [Kplus,Kminus]=KStest(x)
  x=sort(x);
  n=length(x);
  diffMaxPlus=-1e+99;
  diffMaxMinus=-1e+99;
  i=1;
  for xv=0.001:.001:1
    F=xv;
    while x(i)<=xv && i<n
      i=i+1;
    end
    Fn=1;
    if i<n, Fn=(i-1)/n; end
    diff=Fn-F;
    if diff>diffMaxPlus, diffMaxPlus=diff; end
    diff=-diff;
    if diff>diffMaxMinus, diffMaxMinus=diff; end
  end
  Kplus=sqrt(n)*diffMaxPlus;
  Kminus=sqrt(n)*diffMaxMinus;
end
```

Listing 2.3. The listing of file rng997Gen1.m.

Tables 2.1a and 2.1b show the results of the wide-trust region optimization.

	res1	res2	res3
	Wide Range	Wide Range	Wide Range
Mean	111.9602243	111.9681908	112.0899548
Sdev	2.653348756	2.572539963	2.446305191
Min	103.4767638	102.2369659	105.9996306
Max	117.0005992	117.492739	116.6656944
Range	13.52383549	15.25577314	10.66606388
Count	100	100	100
Conf	0.594722314	0.576609809	0.548315513
CI Upper	112.5549466	112.5448006	112.6382703
CI Lower	111.365502	111.391581	111.5416392
Max Elements	10000	10000	10000
Max Iters	100	100	100
a11Low	0	0	0
a11Hi	1	100	1000
a12Low	1000	1000	1000
a12Hi	2000000	2000000	2000000
a13Low	-2000000	-2000000	-2000000
a13Hii	0	0	0
a21Low	1000	1000	1000
a21Hi	2000000	2000000	2000000
a22Low	0	0	0
a22Hi	1	100	1000
a23Low	-2000000	-2000000	-2000000
a23Hi	0	0	0
ix1	306041862	2916825201	1338843591
ix2	2442885337	473460703	4267614411

ix3	3350301901	1862950369	749557152
iy1	2880015722	759069798	1491848605
iy2	366832969	1359555121	1337544205
iy3	2536232089	2039413687	1445604250
PopSize	250	250	250
PopMaxIters	350	350	350
M1	4294967087	4294967087	4294967087
M2	4294944443	4294944443	4294944443
Rounded?	TRUE	TRUE	TRUE
Rounded	10	10	10

Table 2.1a. The results of the wide-trust region optimization.

	res4	res5	res6
	Wide Range	Wide Range	Wide Range
Mean	112.136792	111.9517767	112.3138481
Sdev	2.661015361	2.622816022	2.192777325
Min	105.5241272	103.0417954	106.1609055
Max	117.1699496	116.7517248	117.2531824
Range	11.64582238	13.70992947	11.09227697
Count	100	100	100
Conf	0.596440709	0.587878698	0.491489708
CI Upper	112.7332327	112.5396554	112.8053378
CI Lower	111.5403513	111.363898	111.8223584
Max Elements	10000	10000	10000
Max Iters	100	100	100
a11Low	0	0	0
a11Hi	1	100	1000
a12Low	10000	10000	10000
a12Hi	2000000	2000000	2000000
a13Low	-2000000	-2000000	-2000000
a13Hii	0	0	0
a21Low	10000	10000	10000
a21Hi	2000000	2000000	2000000
a22Low	0	0	0
a22Hi	1	100	1000
a23Low	-2000000	-2000000	-2000000
a23Hi	0	0	0
ix1	3932642767	2433053437	29994158
ix2	3016611303	2933393846	3897699583
ix3	2507901005	3274351493	267020838
iy1	1190650419	3160005557	665463533
iy2	282861080	2693398183	1007406415
iy3	1575948134	3057645457	2268571043
PopSize	250	250	250
PopMaxIters	350	350	350
M1	4294967087	4294967087	4294967087
M2	4294944443	4294944443	4294944443
Rounded?	TRUE	TRUE	TRUE
Rounded	10	10	10

Table 2.1b. The results of the wide-trust region optimization.

Table 2.2a and 2.2b show the results of the narrow-trust region optimization.

	res1b	res2b	res3b
	Narrow Range	Narrow Range	Narrow Range
Mean	112.2099733	112.1892352	112.2512264
Sdev	2.460676434	2.53908517	2.216337633
Min	105.5252904	104.3032961	104.7407355
Max	116.7114255	116.5857709	116.6727269
Range	11.18613506	12.28247477	11.93199136
Count	100	100	100
Conf	0.551536687	0.569111243	0.496770522
CI Upper	112.76151	112.7583465	112.7479969
CI Lower	111.6584366	111.620124	111.7544559
Max Elements	10000	10000	10000
Max Iters	100	100	100
a11Low	0	47	0
a11Hi	3	63	3
a12Low	1546761	1485261	1697511
a12Hi	2092677	2009471	2296633
a13Low	-422388	-1715330	-27708
a13Hii	-312200	-1267852	-20480
a21Low	468026	1372003	139267
a21Hi	633212	1856239	188421
a22Low	0	76	380
a22Hi	3	102	514
a23Low	-1301986	-1101364	-2300000
a23Hi	-962338	-814052	-1700000
ix1	4109167261	4114807346	2775398578
ix2	2513706028	2340996577	3699556250
ix3	2796599123	4251772451	3492636339
iy1	234157937	2067661477	2580017232
iy2	4150310498	3430316983	3228515308
iy3	3543431380	2046178337	683277733
PopSize	250	250	250
PopMaxIters	350	350	350
M1	4294967087	4294967087	4294967087
M2	4294944443	4294944443	4294944443
Rounded?	TRUE	TRUE	TRUE
Rounded	10	10	10

Table 2.2a. The results of the narrow-trust region optimization.

	res4b	res5b	res6b
	Narrow Range	Narrow Range	Narrow Range
Mean	112.2688177	112.0933148	112.2601284
Sdev	2.913525356	2.696182687	2.279049992
Min	103.7962965	102.8193555	105.6504789
Max	117.590651	118.6711102	117.6854461
Range	13.79435458	15.85175477	12.03496713
Count	100	100	100
Conf	0.653038368	0.604323123	0.510826887
CI Upper	112.921856	112.6976379	112.7709552
CI Lower	111.6157793	111.4889917	111.7493015

Max Elements	**10000**	**10000**	**10000**
Max Iters	**100**	**100**	**100**
a11Low	**0**	**31**	**310**
a11Hi	**3**	**41**	**420**
a12Low	**896651**	**379058**	**1700000**
a12Hi	**1213117**	**512844**	**2300000**
a13Low	**-1915883**	**-1446429**	**-1650471**
a13Hii	**-1416087**	**-1069099**	**-1219913**
a21Low	**1547534**	**611378**	**1130986**
a21Hi	**2093722**	**827158**	**1530158**
a22Low	**0**	**65**	**519**
a22Hi	**3**	**89**	**702**
a23Low	**-2300000**	**-879857**	**-1260562**
a23Hi	**-1700000**	**-650329**	**-931720**
ix1	**284863429**	**1023540786**	**1993605375**
ix2	**2741861592**	**421473029**	**2928175711**
ix3	**1010797766**	**3162667378**	**2779311712**
iy1	**2330287905**	**1391311901**	**151872576**
iy2	**1956201758**	**1140727739**	**2799878655**
iy3	**3259784308**	**3297665358**	**1690662562**
PopSize	**250**	**250**	**250**
PopMaxIters	**350**	**350**	**350**
M1	**4294967087**	**4294967087**	**4294967087**
M2	**4294944443**	**4294944443**	**4294944443**
Rounded?	**TRUE**	**TRUE**	**TRUE**
Rounded	**10**	**10**	**10**

Table 2.2b. The results of the narrow-trust region optimization.

Listing 2.4 shows the source code for file do2.m which performs the random-seed generation of one million sets of 10,000 random numbers for each tested version of the algorithm.

```
maxElems=10000;
maxIters=1000000;
xM1 = 2^32 - 209;
xM2 = 2^32 - 22853;
c=[0,1943832,-340972,486153,2,-962338,xM1,xM2];
sFilename='res1c.xlsb';
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
xM1 = 2^32 - 209;
xM2 = 2^32 - 22853;
c=[60,1659661,-1631406,1646104,78,-825525,xM1,xM2];
sFilename='res2c.xlsb';
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,bRound,nDigits)
```



```
maxElems=10000;
maxIters=1000000;
xM1 = 2^32 - 209;
xM2 = 2^32 - 22853;
c=[1,2030635,-23942,159496,380,-1840026,xM1,xM2];
sFilename='res3c.xlsb';
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
xM1 = 2^32 - 209;
xM2 = 2^32 - 22853;
c=[2,1162879,-1696917,1818501,2,-2046843,xM1,xM2];
sFilename='res4c.xlsb';
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
xM1 = 2^32 - 209;
xM2 = 2^32 - 22853;
c=[38,493464,-1119201,827158,74,-768574,xM1,xM2];
sFilename='res5c.xlsb';
bRound=true;
nDigits = 10;
doAll2(maxElems,maxIters,c,sFilename,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
xM1 = 2^32 - 209;
xM2 = 2^32 - 22853;
c=[378,2002198,-1441187,1415119,632,-1245802,xM1,xM2];
sFilename='res6c.xlsb';
bRound=true;
nDigits = 10;
doAll2(maxElems,maxIters,c,sFilename,bRound,nDigits)

% ------------------------------------------------------------------------

system('shutdown /s')
```

Listing 2.4. The source code of file do2.m.

Listing 2.5 shows the source code for file doAll2.m.

```
function doAll2(maxElems,maxIters,c,sFilename,bRound,nDigits)
global gmaxElems
global doRounding
global numDigits

gmaxElems = maxElems;
doRounding = bRound;
```



```
numDigits = nDigits;
bResetResMat = true;
% output file exists?
% if isfile(sFilename)
%   resMat = readmatrix(sFilename,"Sheet", sSheetName,"Range","A2:M251");
%   [nrows,~] = size(resMat);
%   if nrows > 0
%     for nStart=1:nrows
%       if resMat(nStart,1)>1e+99, break; end
%     end
%     if nStart <= nrows, bResetResMat = false; end
%   end
% end

if bResetResMat
  resMat=2e+99+zeros(maxIters,15);
  nStart = 1;
end

for i=nStart:maxIters
  [factor,XYrnd] = rng997Gen2(c);
  resMat(i,1) = factor;
  resMat(i,2:9) = c;
  resMat(i,10:15) = XYrnd;

  fprintf("itr = %i, Factor = %f, X= [", i , factor);
  fprintf("%i, ", c);
  fprintf("]\n");

  resMat = sortrows(resMat,1);

%   T1 = array2table(resMat);
%   T1.Properties.VariableNames(1:13) = {'Factor', 'C11','C12', 'C13' ,'C21',
'C22' ,'C23', ...
%       'M1', 'M2', 'IX1' 'IX2', 'IX3', 'IY1', 'IY2', 'IY3'};
%   writetable(T1, sFilename, "Sheet", sSheetName);
end

fprintf("\n\n");
resMat = sortrows(resMat,1);

beep on;
for i=1:3
  beep;
  pause(1);
end


fprintf("Entire result matrix written to file %s\n",  sFilename)
T1 = array2table(resMat);
T1.Properties.VariableNames(1:15) = {'Factor', 'C11','C12', 'C13' ,'C21',
'C22' ,'C23', ...
      'M1', 'M2', 'IX1' 'IX2', 'IX3', 'IY1', 'Y2', 'Y3'};
writetable(T1, sFilename, "Sheet", "Sheet1");

fprintf("-------------------------------------------------------\n\n");
end
```


Listing 2.5. The source code of file doAll2.m.

Listing 2.6 shows the source code for file rng997Gen2.m.

```
function [factor,XYrnd] = rng997Gen1(c)
%UNTITLED2 Summary of this function goes here
  global gmaxElems
  global doRounding
  global numDigits

  maxElems = gmaxElems;
  rng('shuffle','twister');
  c = round(c, 0);
  a11 = c(1);
  a12 = c(2);
  a13 = c(3);
  a21 = c(4);
  a22 = c(5);
  a23 = c(6);
  M1 = c(7);
  M2 = c(8);
  ix = 1+fix((M2-2)*rand(1,4));
  iy = 1+fix((M2-2)*rand(1,4));
  XYrnd = [ix(1:3) iy(1:3)];
  x=zeros(maxElems,1);
  for i=1:maxElems
    ix(4) = mod(a11*ix(3)+a12*ix(2)+a13*ix(1),M1);
    iy(4) = mod(a21*iy(3)+a22*iy(2)+a23*iy(1),M2);
    iz = mod(ix(4) - iy(4),M1);
    if iz <= 0, iz = iz + M1; end
    x(i) = (iz+1)/(M1+1);
    ix(1:3) = ix(2:4);
    iy(1:3) = iy(2:4);
  end
  if doRounding, x = round(x,numDigits); end
  factor=calcFactor(x,false);
  if isnan(factor), factor=65535; end
end

function x = frac(x)
  x=mod(x,1);
end

function factor = calcFactor(x, bShowResults)
% Calculate the factor statistic for the array of random nnumbers x.

  if nargin < 2, bShowResults = false; end
  maxElems=length(x);
  meanx=mean(x);
  sdevx=std(x);
  % get the first 100 autocorrelation values
  acArr=autocorrArr(x,1,100);
  % calculate the chisquare for the 10-bin histogram
  numBins=10;
  expval=maxElems/numBins;
  [N1,ev1]=histcounts(x,numBins);
```



```
  chiSq10=sum((N1-expval).^2/expval);
  numBins=20;
  expval=maxElems/numBins;
  [N2,ev2]=histcounts(x,numBins);
  chiSq20=sum((N2-expval).^2/expval);
  numBins=20;
  [N3,ev3]=histcounts(acArr,numBins);
  ev3c=ev3(2:length(ev3));
  autoCorrSum = sum(dot(N3,abs(ev3c)));
  chsStat=chs(x);
  [Kplus,Kminus]=KStest(x);
  factor = 1000*(abs(meanx-0.5)+abs(sdevx-1/sqrt(12)))+100*(max(acArr)-
min(acArr))+100*autoCorrSum+chiSq10+chiSq20/2;
  factor = factor + 10*chsStat + 10*(Kplus + Kminus);
  if bShowResults
    fprintf('Mean = %g\nSdev = %g\n', meanx, sdevx);
    fprintf('Min = %g\nMax = %g\n', min(x), max(x));
    fprintf('Max lags = 100\n');
    fprintf('Auto correlation array\n');
    disp(acArr');
    fprintf('10-Bin Histogram\n');
    disp(N1); disp(ev1);
    fprintf('Chi-Sqr10 = %g\n', chiSq10);
    fprintf('20-Bin Histogram\n');
    disp(N2); disp(ev2);
    fprintf('Chi-Sqr20 = %g\n', chiSq20);
    fprintf('20-Bin Autocorrelation Histogram\n');
    disp(N3); disp(ev3);
    fprintf('Sum autocorrel product = %g\n', autoCorrSum);
    fprintf('Change of sign stat = %g\n', chsStat);
    fprintf('K+ = %g and K- = %g\n', Kplus, Kminus);
    fprintf('Factor = %g\n', factor);
  end
end

function acArr=autocorrArr(xdata,fromLag,toLag)

numLags=toLag-fromLag+1;
acArr=zeros(numLags,1);
j=1;
for i=fromLag:toLag
  acArr(j)=autocor(xdata,i);
  j=j+1;
end
end

function res = autocor(xdata,lag)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
maxElems=length(xdata);
res=corrcoef(xdata(1:maxElems-lag),xdata(lag+1:maxElems));
res=res(1,2);
end

function sumx=chs(x)
% Function CHS calculates the change of sign (between subsequent random
% numbers) moment. The function counts the number of consequtive positive
```

```
% abd negative changes of sign. The last nested loop calculates the
% statistic returned by this function. This value is the sum of:
%
%  sum = sum of difference(count,:) * count / difference(1,:)
%
% Keeping in mind that difference(1,:) is a good value that counts the
% sign flips that happpens one neighbor down. The values for
% difference(n,:) for n>1 are not desirable. The smaller, the better. The
% value difference(2,:) is the number of sign flips that occur
% two neighbors down. The value difference(3,:) is the number of sign flips
% that occur three neighbors down, and so on.

  n=length(x);
  nby2=fix(n/2);
  Diff=zeros(nby2,2);
  countPos=0;
  countNeg=0;
  s1=sign(x(2)-x(1));
  if s1>0
    bIsPos=true;
    countPos=1;
  else
    bIsPos=false;
    countNeg=1;
  end

  for i=3:n
    s2=sign(x(i)-x(i-1));
    % was positive and is still positive
    if s2>0 && bIsPos
      countPos=countPos+1;
    % was negative and is now positive
    elseif s2>0 && ~bIsPos
      bIsPos=true;
      countPos=1;
      Diff(countNeg,2)=Diff(countNeg,2)+1;
      countNeg=0;
    % was negative and is still negative
    elseif s2<0 && ~bIsPos
      countNeg=countNeg+1;
    % was positive is and is now negative
    elseif s2<0 && bIsPos
      bIsPos=false;
      countNeg=1;
      Diff(countPos,1)=Diff(countPos,1)+1;
      countPos=0;
    end
  end

  if s2>0
    if countPos>0, Diff(countPos,1)=Diff(countPos,1)+1; end
  else
    if countNeg>0, Diff(countNeg,2)=Diff(countNeg,2)+1; end
  end

  i=2:nby2;
  d=Diff(2:nby2,:);
```



```
  sumx=0;
  for j=1:2
    sumx = sumx + dot(d(:,j),i)/Diff(1,j);
  end
end

function [Kplus,Kminus]=KStest(x)
  x=sort(x);
  n=length(x);
  diffMaxPlus=-1e+99;
  diffMaxMinus=-1e+99;
  i=1;
  for xv=0.001:.001:1
    F=xv;
    while x(i)<=xv && i<n
      i=i+1;
    end
    Fn=1;
    if i<n, Fn=(i-1)/n; end
    diff=Fn-F;
    if diff>diffMaxPlus, diffMaxPlus=diff; end
    diff=-diff;
    if diff>diffMaxMinus, diffMaxMinus=diff; end
  end
  Kplus=sqrt(n)*diffMaxPlus;
  Kminus=sqrt(n)*diffMaxMinus;
end
```

Listing 2.6. The source code for file rng997Gen2.m.

Tables 2.3a and 2.3b show the results of the penalty factor statistics for the different versions of the algorithm.

	res1c	res2c	res3c
	Random Seed	Random Seed	Random Seed
Mean	147.7564447	147.7575171	147.7557316
Sdev	11.76756118	11.77096018	11.77643722
Min	104.7177526	104.5977864	105.5538376
Max	237.5336647	234.1913261	231.2472902
Range	132.8159122	129.5935397	125.6934526
Count	1000000	1000000	1000000
Conf	0.026375844	0.026383462	0.026395739
CI Upper	147.7828206	147.7839005	147.7821273
CI Lower	147.7300689	147.7311336	147.7293358

Table 2.3a. The random-seed results for the penalty factor statistics.

	res4c	res5c	res6c
	Random Seed	Random Seed	Random Seed
Mean	147.7472893	147.7594847	147.7481452
Sdev	11.80407572	11.76787863	11.77605742
Min	105.3526096	102.2407685	100.0329552
Max	230.1528389	224.940695	236.1139163
Range	124.8002294	122.6999265	136.0809611
Count	1000000	1000000	1000000

Conf	0.026457688	0.026376555	0.026394887
CI Upper	147.773747	147.7858613	147.7745401
CI Lower	147.7208316	147.7331082	147.7217503

Table 2.3b. The random-seed results for the penalty factor statistics.

The column titled res4c in Table 2.3b has the lowest upper mean value. The best modified MRG32A equation is:

```
M1 = 2^32 - 209
M2 = 2^32 - 22853
a11 = 2
a12 = 1162879
a13 = -1696917
a21 = 1818501
a22 = 2
a23 = -2046843
ix = 1+fix((M2-2)*rand(1,4));
iy = 1+fix((M2-2)*rand(1,4));

for iter=1 to random numbers count
  ix(4) = a11*ix(3)+a12*ix(2)+a13*ix(1) mod M1
  iy(4) = a21*iy(3)+a22*iy(2)+a23*iy(1) mod M2
  iz = ix(4) - iy(4) mod M1
  if iz <= 0, iz = iz + M1
  x(iter) = (iz+1)/(M1+1)
  ix(1:3) = ix(2:4)
  iy(1:3) = iy(2:4)
end                                                    (2.2)
```

The algorithm first initializes the arrays ix and iy. You can use these values in the second set of equations for as many iterations as you need. The value x(iter) is the uniform random number generated in the range of 0 to 1(excluded) in each iteration.

3/ MRG32A ALGORITHMS (VERSION ACCORN 3 PSO)

Equation 2.1 shows a general form of the original MRG32a algorithm. In this section I present an extended version MRG32a algorithm. The extension comes from using different moduli values for M1 and M2. The list of various moduli value is:

xM1 = 2^16 - 209
xM2 = 2^32 - 22853
xM3 = 2^32 - 209
xM4 = 2^48 - 22853
xM5 = 2^48 - 209
xM6 = 2^64 - 22853
xM7 = 2^64 - 209

The calculations use different pairs of xM variables to to be the values of M1 and M2.

The approache that estimates the best coefficients for a MRG32a variant uses the following approach:

1. Optimize the penalty factor (see Appendix of Project 997 PRNGs Part 1) using particle swarm optimization algorithms. This optimization starts with a wide trust region and narrows it down.
2. Optimize the penalty using the narrowed optimum regions obtained in step 1. This step yields refined trust regions.
3. Perform a large run of generating random numbers using the refined trust regions from step 2. The calculations yield the statistics for the penalty factor. The upper confidence values for the mean penlty factor are the values we look at to determine the fitness of the algorithm.

All the phases involve obtaining initial random values for the first k elements of arrays ix and iy. This is followed by applying equation set 2.1.

The listing for do.m, which triggers the calculations for the first and second optimization phases is:

```
xM1 = 2^16 - 209;
xM2 = 2^32 - 22853;
xM3 = 2^32 - 209;
xM4 = 2^48 - 22853;
xM5 = 2^48 - 209;
xM6 = 2^64 - 22853;
xM7 = 2^64 - 209;


maxElems=10000;
maxIters=100;
```



```
lb=[0 1000 -2000000 1000 0 -2000000 xM2 xM1];
ub=[1 2000000 0 2000000 1 0 xM3 xM2];
sFilename='res1.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
bRound=true;
nDigits = 10;
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,bRo
und,nDigits)

maxElems=10000;
maxIters=100;
lb=[0 1000 -2000000 1000 0 -2000000 xM3 xM2];
ub=[100 2000000 0 2000000 100 0 xM4 xM3];
sFilename='res2.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,bRo
und,nDigits)

maxElems=10000;
maxIters=100;
lb=[0 1000 -2000000 1000 0 -2000000 xM4 xM3];
ub=[1000 2000000 0 2000000 1000 0 xM5 xM4];
sFilename='res3.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,bRo
und,nDigits)

maxElems=10000;
maxIters=100;
lb=[0 10000 -2000000 10000 0 -2000000 xM5 xM4];
ub=[1 2000000 0 2000000 1 0 xM6 xM5];
sFilename='res4.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,bRo
und,nDigits)
```



```
maxElems=10000;
maxIters=100;
lb=[0 10000 -2000000 10000 0 -2000000 xM6 xM5];
ub=[100 2000000 0 2000000 100 0 xM7 xM6];
sFilename='res5.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,bRo
und,nDigits)

% ------------------------------------------------------------------------

r = 0.15;
maxElems=10000;
maxIters=100;
x = [];
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
% lb(1) = 0;
% ub(1) = 3;
% lb(5) = 0;
% ub(5) = 3;
[lb,ub] = checkBounds(lb,ub);
sFilename='res1b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,bRo
und,nDigits)

maxElems=10000;
maxIters=100;
x = [];
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
[lb,ub] = checkBounds(lb,ub);
sFilename='res2b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,bRo
und,nDigits)
```



```
maxElems=10000;
maxIters=100;
x = [];
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
[lb,ub] = checkBounds(lb,ub);
sFilename='res3b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,bRo
und,nDigits)

maxElems=10000;
maxIters=100;
x = [];
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
lb(1) = 0;
lb(5) = 0;
ub(1) = 3;
ub(5) = 3;
[lb,ub] = checkBounds(lb,ub);
sFilename='res4b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,bRo
und,nDigits)

maxElems=10000;
maxIters=100;
x = [];
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
[lb,ub] = checkBounds(lb,ub);
sFilename='res5b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,bRo
und,nDigits)

system('shutdown /s')
```


Listing 3.1. The listing of file do.m.

Listing 3.1 shows seven values for the varies modulus values. Eacrh call to function doAll() uses a different pairs of values.

Listing 3.2 shows the source code for file doAl.m. The function doAll() optimizes the function rng997Gen1() using the Particle Swarm Optimization (PSO) method by calling function particleswarm(). The function do() calls function doAll() for the first and second optimization phases.

```
function
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,bRo
und,nDigits)
global gmaxElems
global bestFactor

global doRounding
global numDigits
global bestFactor
global bestXYrnd

gmaxElems = maxElems;
doRounding = bRound;
numDigits = nDigits;
bResetResMat = true;
% output file exists?
if isfile(sFilename)
  resMat = readmatrix(sFilename,"Sheet", sSheetName,"Range","A2:O251");
  [nrows,~] = size(resMat);
  if nrows > 0
    for nStart=1:nrows
      if resMat(nStart,1)>1e+99, break; end
    end
    if nStart <= nrows, bResetResMat = false; end
  end
end

if bResetResMat
  resMat=2e+99+zeros(maxIters,15);
  nStart = 1;
end
options =
optimoptions('particleswarm','SwarmSize',POSpopsize,'Display','off','MaxItera
tions',POSmaxIters,'FunctionTolerance',0.01);

for i=nStart:maxIters
  bestFactor = 1e+99;
%   [x, factor] = PSO3(@rng997Gen1, lb, ub, POSmaxIters, POSpopsize);
  x = particleswarm(@rng997Gen1,length(lb),lb,ub,options);
  x = round(x, 0);
```

```
  resMat(i,1) = bestFactor;
  resMat(i,2:9) = x;
  resMat(i,10:15) = bestXYrnd;

  fprintf("itr = %i, Factor = %f, X= [", i , bestFactor);
  fprintf("%i, ", x);
  fprintf("]\n");

  resMat = sortrows(resMat,1);

  T1 = array2table(resMat);
  T1.Properties.VariableNames(1:15) = {'Factor', 'C11','C12', 'C13' ,'C21',
'C22' ,'C23', ...
      'M1', 'M2', 'IX1' 'IX2', 'IX3', 'IY1', 'IY2', 'IY3'};
  writetable(T1, sFilename, "Sheet", sSheetName);
end

fprintf("\n\n");
resMat = sortrows(resMat,1);

beep on;
for i=1:3
  beep;
  pause(1);
end

fprintf("Entire result matrix written to file %s\n",  sFilename)
T1 = array2table(resMat);
  T1.Properties.VariableNames(1:15) = {'Factor', 'C11','C12', 'C13' ,'C21',
'C22' ,'C23', ...
      'M1', 'M2', 'IX1' 'IX2', 'IX3', 'IY1', 'IY2', 'IY3'};
writetable(T1, sFilename, "Sheet", sSheetName);
c = cell(22,2);
c(1,1:2) = {'Max Elements', maxElems};
c(2,1:2) = {'Max Iters', maxIters};
c(3,1:2) = {'a11Low', lb(1)};
c(4,1:2) = {'a11Hi', ub(1)};
c(5,1:2) = {'a12Low', lb(2)};
c(6,1:2) = {'a12Hi', ub(2)};
c(7,1:2) = {'a13Low', lb(3)};
c(8,1:2) = {'a13Hii', ub(3)};
c(9,1:2) = {'a21Low', lb(4)};
c(10,1:2) = {'a21Hi', ub(4)};
c(11,1:2) = {'a22Low', lb(5)};
c(12,1:2) = {'a22Hi', ub(5)};
c(13,1:2) = {'a23Low', lb(6)};
c(14,1:2) = {'a23Hi', ub(6)};
c(15,1:2) = {'M1Low', lb(7)};
c(16,1:2) = {'M1Hi', ub(7)};
c(17,1:2) = {'M2Low', lb(8)};
c(18,1:2) = {'M2Hi', ub(8)};
c(19,1:2) = {'PopSize', POSpopsize};
c(20,1:2) = {'PopMaxIters', POSmaxIters};
```



```
if doRounding
  c(21,1:2) = {'Rounded?', "True"};
  c(22,1:2) = {'Rounded', nDigits};
else
  c(21,1:2) = {'Rounded?', "False"};
  c(22,1:2) = {'Rounded', "N/A"};
end
T2 = cell2table(c);
writetable(T2, sFilename, "Sheet", "Params");
fprintf("-------------------------------------------------------\n\n");
end
```

Listing 3.2. The listing of file doAll.m.

Listing 3.3 shows the listing of Reg997Gen1.m.

```
function factor = rng997Gen1(c)
%UNTITLED2 Summary of this function goes here
  global gmaxElems

  global doRounding
  global numDigits
  global bestFactor
  global bestXYrnd

  maxElems = gmaxElems;
  rng('shuffle','twister');
  c = round(c, 0);
  a11 = c(1);
  a12 = c(2);
  a13 = c(3);
  a21 = c(4);
  a22 = c(5);
  a23 = c(6);
  M1 = c(7);
  M2 = c(8);
  ix = 1+fix((M2-2)*rand(1,4));
  iy = 1+fix((M2-2)*rand(1,4));
  bestXYrnd = [ix(1:3) iy(1:3)];
  x=zeros(maxElems,1);
  for i=1:maxElems
    ix(4) = mod(a11*ix(3)+a12*ix(2)+a13*ix(1),M1);
    iy(4) = mod(a21*iy(3)+a22*iy(2)+a23*iy(1),M2);
    iz = mod(ix(4) - iy(4),M1);
    if iz <= 0, iz = iz + M1; end
    x(i) = (iz+1)/(M1+1);
    ix(1:3) = ix(2:4);
    iy(1:3) = iy(2:4);
  end
  if doRounding, x = round(x,numDigits); end
  factor=calcFactor(x,false);
  if isnan(factor), factor=65535; end
  if factor < bestFactor
    bestFactor = factor;
  end
end
```



```
function x = frac(x)
  x=mod(x,1);
end

function factor = calcFactor(x, bShowResults)
% Calculate the factor statistic for the array of random nnumbers x.

  if nargin < 2, bShowResults = false; end
  maxElems=length(x);
  meanx=mean(x);
  sdevx=std(x);
  % get the first 100 autocorrelation values
  acArr=autocorrArr(x,1,100);
  % calculate the chisquare for the 10-bin histogram
  numBins=10;
  expval=maxElems/numBins;
  [N1,ev1]=histcounts(x,numBins);
  chiSq10=sum((N1-expval).^2/expval);
  numBins=20;
  expval=maxElems/numBins;
  [N2,ev2]=histcounts(x,numBins);
  chiSq20=sum((N2-expval).^2/expval);
  numBins=20;
  [N3,ev3]=histcounts(acArr,numBins);
  ev3c=ev3(2:length(ev3));
  autoCorrSum = sum(dot(N3,abs(ev3c)));
  chsStat=chs(x);
  [Kplus,Kminus]=KStest(x);
  factor = 1000*(abs(meanx-0.5)+abs(sdevx-1/sqrt(12)))+100*(max(acArr)-
min(acArr))+100*autoCorrSum+chiSq10+chiSq20/2;
  factor = factor + 10*chsStat + 10*(Kplus + Kminus);
  if bShowResults
    fprintf('Mean = %g\nSdev = %g\n', meanx, sdevx);
    fprintf('Min = %g\nMax = %g\n', min(x), max(x));
    fprintf('Max lags = 100\n');
    fprintf('Auto correlation array\n');
    disp(acArr');
    fprintf('10-Bin Histogram\n');
    disp(N1); disp(ev1);
    fprintf('Chi-Sqr10 = %g\n', chiSq10);
    fprintf('20-Bin Histogram\n');
    disp(N2); disp(ev2);
    fprintf('Chi-Sqr20 = %g\n', chiSq20);
    fprintf('20-Bin Autocorrelation Histogram\n');
    disp(N3); disp(ev3);
    fprintf('Sum autocorrel product = %g\n', autoCorrSum);
    fprintf('Change of sign stat = %g\n', chsStat);
    fprintf('K+ = %g and K- = %g\n', Kplus, Kminus);
    fprintf('Factor = %g\n', factor);
  end
end

function acArr=autocorrArr(xdata,fromLag,toLag)

numLags=toLag-fromLag+1;
acArr=zeros(numLags,1);
```



```
j=1;
for i=fromLag:toLag
  acArr(j)=autocor(xdata,i);
  j=j+1;
end
end

function res = autocor(xdata,lag)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
maxElems=length(xdata);
res=corrcoef(xdata(1:maxElems-lag),xdata(lag+1:maxElems));
res=res(1,2);
end

function sumx=chs(x)
% Function CHS calculates the change of sign (between subsequent random
% numbers) moment. The function counts the number of consequtive positive
% abd negative changes of sign. The last nested loop calculates the
% statistic returned by this function. This value is the sum of:
%
%  sum = sum of difference(count,:) * count / difference(1,:)
%
% Keeping in mind that difference(1,:) is a good value that counts the
% sign flips that happpens one neighbor down. The values for
% difference(n,:) for n>1 are not desirable. The smaller, the better. The
% value difference(2,:) is the number of sign flips that occur
% two neighbors down. The value difference(3,:) is the number of sign flips
% that occur three neighbors down, and so on.

  n=length(x);
  nby2=fix(n/2);
  Diff=zeros(nby2,2);
  countPos=0;
  countNeg=0;
  s1=sign(x(2)-x(1));
  if s1>0
    bIsPos=true;
    countPos=1;
  else
    bIsPos=false;
    countNeg=1;
  end

  for i=3:n
    s2=sign(x(i)-x(i-1));
    % was positive and is still positive
    if s2>0 && bIsPos
      countPos=countPos+1;
    % was negative and is now positive
    elseif s2>0 && ~bIsPos
      bIsPos=true;
      countPos=1;
      Diff(countNeg,2)=Diff(countNeg,2)+1;
      countNeg=0;
    % was negative and is still negative
    elseif s2<0 && ~bIsPos
```



```
      countNeg=countNeg+1;
    % was positive is and is now negative
    elseif s2<0 && bIsPos
      bIsPos=false;
      countNeg=1;
      Diff(countPos,1)=Diff(countPos,1)+1;
      countPos=0;
    end
  end

  if s2>0
    if countPos>0, Diff(countPos,1)=Diff(countPos,1)+1; end
  else
    if countNeg>0, Diff(countNeg,2)=Diff(countNeg,2)+1; end
  end

  i=2:nby2;
  d=Diff(2:nby2,:);
  sumx=0;
  for j=1:2
    sumx = sumx + dot(d(:,j),i)/Diff(1,j);
  end
end

function [Kplus,Kminus]=KStest(x)
  x=sort(x);
  n=length(x);
  diffMaxPlus=-1e+99;
  diffMaxMinus=-1e+99;
  i=1;
  for xv=0.001:.001:1
    F=xv;
    while x(i)<=xv && i<n
      i=i+1;
    end
    Fn=1;
    if i<n, Fn=(i-1)/n; end
    diff=Fn-F;
    if diff>diffMaxPlus, diffMaxPlus=diff; end
    diff=-diff;
    if diff>diffMaxMinus, diffMaxMinus=diff; end
  end
  Kplus=sqrt(n)*diffMaxPlus;
  Kminus=sqrt(n)*diffMaxMinus;
end
```

Listing 3.3. The listing of file rng997Gen1.m.

Table 3.1 shows the results for the wide range trust-region optimization.

	res1	res2	res3	res4	res5
	Wide Range	Wide Range	Wide Range	Wide Range	Wide Range
Mean	111.7820496	112.4881526	112.316101	111.7958147	111.2539347
Sdev	2.957623951	2.118757032	2.426300674	2.616661222	5.611912942
Min	100.127892	105.0434258	104.6282817	101.1654103	73.18909649
Max	117.480752	117.3012365	116.4838304	116.5937259	116.8491649

Range	17.35285995	12.25781068	11.85554865	15.4283156	43.66006836
Count	100	100	100	100	100
Conf	0.662922639	0.474898779	0.543831695	0.58649916	1.257855698
CI Upper	112.4449723	112.9630514	112.8599327	112.3823139	112.5117904
CI Lower	111.119127	112.0132538	111.7722693	111.2093156	109.996079
Max Elements	10000	10000	10000	10000	10000
Max Iters	100	100	100	100	100
a11Low	0	0	0	0	0
a11Hi	1	100	1000	1	100
a12Low	1000	1000	1000	10000	10000
a12Hi	2000000	2000000	2000000	2000000	2000000
a13Low	-2000000	-2000000	-2000000	-2000000	-2000000
a13Hii	0	0	0	0	0
a21Low	1000	1000	1000	10000	10000
a21Hi	2000000	2000000	2000000	2000000	2000000
a22Low	0	0	0	0	0
a22Hi	1	100	1000	1	100
a23Low	-2000000	-2000000	-2000000	-2000000	-2000000
a23Hi	0	0	0	0	0
M1Low	4294944443	4294967087	2.81475E+14	2.81475E+14	1.84E+19
M1Hi	4294967087	2.81475E+14	2.81475E+14	1.84E+19	1.84E+19
M2Low	65327	4294944443	4294967087	2.81475E+14	2.81475E+14
M2Hi	4294944443	4294967087	2.81475E+14	2.81475E+14	1.84E+19
PopSize	250	250	250	250	250
PopMaxIters	350	350	350	350	350
Rounded?	TRUE	TRUE	TRUE	TRUE	TRUE
Rounded	10	10	10	10	10

Table 3.1. The results of the wide-trust region optimization.

Table 3.2 shows the results for the narrow-trust region optimization.

	res1b	res2b	res3b	res4b	res5b
	Narrow Range	Narrow Range	Narrow Range	Narrow Range	Narrow Range
Mean	112.3575791	111.7662617	112.3085114	112.6512593	112.6219421
Sdev	2.498937171	2.684448362	2.573331406	2.343072652	2.435951649
Min	103.7446326	100.2812571	104.9032768	104.5921891	104.6752721
Max	117.216855	116.3869502	117.404054	118.5260186	117.5981218
Range	13.47222232	16.10569313	12.50077727	13.93382949	12.92284966
Count	100	100	100	100	100
Conf	0.560112459	0.601692988	0.576787203	0.525176943	0.545994867
CI Upper	112.9176916	112.3679547	112.8852986	113.1764362	113.1679369
CI Lower	111.7974667	111.1645687	111.7317242	112.1260824	112.0759472
Max Elements	10000	10000	10000	10000	10000
Max Iters	100	100	100	100	100
a11Low	0	78	769	0	43
a11Hi	3	106	1041	3	59
a12Low	1700000	850	1270146	1268101	1151475
a12Hi	2300000	1150	1718432	1715665	1557877

a13Low	-1148949	-2300000	0	0	-2300000
a13Hii	-849223	-1700000	3	3	-1700000
a21Low	580463	1700000	850	171485	1078315
a21Hi	785333	2300000	1150	232009	1458897
a22Low	0	85	268	1	32
a22Hi	3	115	362	3	44
a23Low	-815460	-1269668	-940452	-414053	0
a23Hi	-602732	-938450	-695116	-306039	3
M1Low	3650721297	3650722024	2.39254E+14	9.43E+18	1.57E+19
M1Hi	4939211167	4939212150	3.23696E+14	1.28E+19	2.12E+19
M2Low	2909242768	3650711305	1.11455E+14	2.39254E+14	2.39254E+14
M2Hi	3936034334	4939197647	1.50793E+14	3.23696E+14	3.23696E+14
PopSize	250	250	250	250	250
PopMaxIters	350	350	350	350	350
Rounded?	TRUE	TRUE	TRUE	TRUE	TRUE
Rounded	10	10	10	10	10

Table 3.2. The results of the narrow *trus- region optimization.*

Listing 3.4 shows the source code for file do2.m which performs the random-seed generation of one million sets of 10,000 random numbers for each tested version of the algorithm.

```
maxElems=10000;
maxIters=1000000;
c = [2,2300000,-1004650,773587,1,-602732,4440148753,3648033672];
sFilename='res1c.xlsb';
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c=[104,1094,-2105947,1700000,85,-1044872,4776406650,3650711305];
sFilename='res2c.xlsb';
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c=[938,1474678,2,1001,296,-862211,2.48312E+14,1.29046E+14];
sFilename='res3c.xlsb';
bRound=true;
nDigits = 10;
doAll2(maxElems,maxIters,c,sFilename,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c=[1,1552114,1,177528,3,-344769,1.22718E+19,2.92088E+14];
sFilename='res4c.xlsb';
bRound=true;
nDigits = 10;
doAll2(maxElems,maxIters,c,sFilename,bRound,nDigits)
```



```
maxElems=10000;
maxIters=1000000;
c=[51,1354676,-2000000,1268606,38,0,1.84467E+19,2.81475E+14];
sFilename='res5c.xlsb';
bRound=true;
nDigits = 10;
doAll2(maxElems,maxIters,c,sFilename,bRound,nDigits)

% ------------------------------------------------------------------------

system('shutdown /s')
```

Listing 3.4. The source code file do2.m.

Listing 3.5 shows the source code for file doAll2.m.

```
function doAll2(maxElems,maxIters,c,sFilename,bRound,nDigits)
global gmaxElems
global doRounding
global numDigits

gmaxElems = maxElems;
doRounding = bRound;
numDigits = nDigits;
bResetResMat = true;
% output file exists?
% if isfile(sFilename)
%   resMat = readmatrix(sFilename,"Sheet", sSheetName,"Range","A2:101");
%   [nrows,~] = size(resMat);
%   if nrows > 0
%     for nStart=1:nrows
%       if resMat(nStart,1)>1e+99, break; end
%     end
%     if nStart <= nrows, bResetResMat = false; end
%   end
% end

if bResetResMat
  resMat=2e+99+zeros(maxIters,15);
  nStart = 1;
end

for i=nStart:maxIters
  [factor,XYrnd] = rng997Gen2(c);
  resMat(i,1) = factor;
  resMat(i,2:9) = c;
  resMat(i,10:15) = XYrnd;

  fprintf("itr = %i, Factor = %f, X= [", i , factor);
  fprintf("%i, ", c);
  fprintf("]\n");

  resMat = sortrows(resMat,1);

%   T1 = array2table(resMat);
%   T1.Properties.VariableNames(1:15) = {'Factor', 'C11','C12', 'C13' ,'C21',
'C22' ,'C23', ...
%      'M1', 'M2', 'IX1' 'IX2', 'IX3', 'IY1', 'IY2', 'IY3'};
```



```
%   writetable(T1, sFilename, "Sheet", "Sheet1");
end

fprintf("\n\n");
resMat = sortrows(resMat,1);

beep on;
for i=1:3
  beep;
  pause(1);
end

fprintf("Entire result matrix written to file %s\n",  sFilename)
T1 = array2table(resMat);
  T1.Properties.VariableNames(1:15) = {'Factor', 'C11','C12', 'C13' ,'C21',
'C22' ,'C23', ...
     'M1', 'M2', 'IX1' 'IX2', 'IX3', 'IY1', 'IY2', 'IY3'};
writetable(T1, sFilename, "Sheet", "Sheet1");

fprintf("-------------------------------------------------------\n\n");
end
```

Listing 3.5. The source code file doAll2.m.

Listing 3.6 shows the source code for file rng997Gen2.m.

```
function [factor,XYrnd] = rng997Gen2(c)
%UNTITLED2 Summary of this function goes here
  global gmaxElems
  global doRounding
  global numDigits

  maxElems = gmaxElems;
  rng('shuffle','twister');
  c = round(c, 0);
  a11 = c(1);
  a12 = c(2);
  a13 = c(3);
  a21 = c(4);
  a22 = c(5);
  a23 = c(6);
  M1 = c(7);
  M2 = c(8);
  ix = 1+fix((M2-2)*rand(1,4));
  iy = 1+fix((M2-2)*rand(1,4));
  XYrnd = [ix(1:3) iy(1:3)];
  x=zeros(maxElems,1);
  for i=1:maxElems
    ix(4) = mod(a11*ix(3)+a12*ix(2)+a13*ix(1),M1);
    iy(4) = mod(a21*iy(3)+a22*iy(2)+a23*iy(1),M2);
    iz = mod(ix(4) - iy(4),M1);
    if iz <= 0, iz = iz + M1; end
    x(i) = (iz+1)/(M1+1);
    ix(1:3) = ix(2:4);
    iy(1:3) = iy(2:4);
  end
  if doRounding, x = round(x,numDigits); end
```



```
  factor=calcFactor(x,false);
  if isnan(factor), factor=65535; end
end

function x = frac(x)
  x=mod(x,1);
end

function factor = calcFactor(x, bShowResults)
% Calculate the factor statistic for the array of random nnumbers x.

  if nargin < 2, bShowResults = false; end
  maxElems=length(x);
  meanx=mean(x);
  sdevx=std(x);
  % get the first 100 autocorrelation values
  acArr=autocorrArr(x,1,100);
  % calculate the chisquare for the 10-bin histogram
  numBins=10;
  expval=maxElems/numBins;
  [N1,ev1]=histcounts(x,numBins);
  chiSq10=sum((N1-expval).^2/expval);
  numBins=20;
  expval=maxElems/numBins;
  [N2,ev2]=histcounts(x,numBins);
  chiSq20=sum((N2-expval).^2/expval);
  numBins=20;
  [N3,ev3]=histcounts(acArr,numBins);
  ev3c=ev3(2:length(ev3));
  autoCorrSum = sum(dot(N3,abs(ev3c)));
  chsStat=chs(x);
  [Kplus,Kminus]=KStest(x);
  factor = 1000*(abs(meanx-0.5)+abs(sdevx-1/sqrt(12)))+100*(max(acArr)-
min(acArr))+100*autoCorrSum+chiSq10+chiSq20/2;
  factor = factor + 10*chsStat + 10*(Kplus + Kminus);
  if bShowResults
    fprintf('Mean = %g\nSdev = %g\n', meanx, sdevx);
    fprintf('Min = %g\nMax = %g\n', min(x), max(x));
    fprintf('Max lags = 100\n');
    fprintf('Auto correlation array\n');
    disp(acArr');
    fprintf('10-Bin Histogram\n');
    disp(N1); disp(ev1);
    fprintf('Chi-Sqr10 = %g\n', chiSq10);
    fprintf('20-Bin Histogram\n');
    disp(N2); disp(ev2);
    fprintf('Chi-Sqr20 = %g\n', chiSq20);
    fprintf('20-Bin Autocorrelation Histogram\n');
    disp(N3); disp(ev3);
    fprintf('Sum autocorrel product = %g\n', autoCorrSum);
    fprintf('Change of sign stat = %g\n', chsStat);
    fprintf('K+ = %g and K- = %g\n', Kplus, Kminus);
    fprintf('Factor = %g\n', factor);
  end
end

function acArr=autocorrArr(xdata,fromLag,toLag)
```



```
numLags=toLag-fromLag+1;
acArr=zeros(numLags,1);
j=1;
for i=fromLag:toLag
  acArr(j)=autocor(xdata,i);
  j=j+1;
end
end

function res = autocor(xdata,lag)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
maxElems=length(xdata);
res=corrcoef(xdata(1:maxElems-lag),xdata(lag+1:maxElems));
res=res(1,2);
end

function sumx=chs(x)
% Function CHS calculates the change of sign (between subsequent random
% numbers) moment. The function counts the number of consequtive positive
% abd negative changes of sign. The last nested loop calculates the
% statistic returned by this function. This value is the sum of:
%
%  sum = sum of difference(count,:) * count / difference(1,:)
%
% Keeping in mind that difference(1,:) is a good value that counts the
% sign flips that happpens one neighbor down. The values for
% difference(n,:) for n>1 are not desirable. The smaller, the better. The
% value difference(2,:) is the number of sign flips that occur
% two neighbors down. The value difference(3,:) is the number of sign flips
% that occur three neighbors down, and so on.

  n=length(x);
  nby2=fix(n/2);
  Diff=zeros(nby2,2);
  countPos=0;
  countNeg=0;
  s1=sign(x(2)-x(1));
  if s1>0
    bIsPos=true;
    countPos=1;
  else
    bIsPos=false;
    countNeg=1;
  end

  for i=3:n
    s2=sign(x(i)-x(i-1));
    % was positive and is still positive
    if s2>0 && bIsPos
      countPos=countPos+1;
    % was negative and is now positive
    elseif s2>0 && ~bIsPos
      bIsPos=true;
      countPos=1;
      Diff(countNeg,2)=Diff(countNeg,2)+1;
```



```
      countNeg=0;
    % was negative and is still negative
    elseif s2<0 && ~bIsPos
      countNeg=countNeg+1;
    % was positive is and is now negative
    elseif s2<0 && bIsPos
      bIsPos=false;
      countNeg=1;
      Diff(countPos,1)=Diff(countPos,1)+1;
      countPos=0;
    end
  end

  if s2>0
    if countPos>0, Diff(countPos,1)=Diff(countPos,1)+1; end
  else
    if countNeg>0, Diff(countNeg,2)=Diff(countNeg,2)+1; end
  end

  i=2:nby2;
  d=Diff(2:nby2,:);
  sumx=0;
  for j=1:2
    sumx = sumx + dot(d(:,j),i)/Diff(1,j);
  end
end

function [Kplus,Kminus]=KStest(x)
  x=sort(x);
  n=length(x);
  diffMaxPlus=-1e+99;
  diffMaxMinus=-1e+99;
  i=1;
  for xv=0.001:.001:1
    F=xv;
    while x(i)<=xv && i<n
      i=i+1;
    end
    Fn=1;
    if i<n, Fn=(i-1)/n; end
    diff=Fn-F;
    if diff>diffMaxPlus, diffMaxPlus=diff; end
    diff=-diff;
    if diff>diffMaxMinus, diffMaxMinus=diff; end
  end
  Kplus=sqrt(n)*diffMaxPlus;
  Kminus=sqrt(n)*diffMaxMinus;
end
```

Listing 3.6. The source code of file rng997Gen2.m.

Table 3.3 shows the results of the penalty factor statistics for the different versions of the algorithm.

	res1c	res2c	res3c	res4c	res5c
	Random Seed	Random Seed	Random Seed	Random Seed	Random Seed
Mean	147.7716607	147.7556833	147.746099	147.7567523	147.763349

Sdev	11.78106126	11.76976662	11.76614899	11.78860867	11.77946147
Min	106.326982	103.5051193	103.419189	104.4942822	103.5144782
Max	231.6587204	229.2869733	226.5668259	228.8651511	230.8426102
Range	125.3317384	125.7818541	123.1476369	124.3708688	127.328132
Count	1000000	1000000	1000000	1000000	1000000
Conf	0.026406103	0.026380787	0.026372678	0.02642302	0.026402517
CI Upper	147.7980668	147.7820641	147.7724717	147.7831753	147.7897515
CI Lower	147.7452546	147.7293025	147.7197263	147.7303293	147.7369465

Table 3.3. The random-seed results for the penalty factor statistics.

The column titled res3c in Table 3.3 has the lowest upper mean value. The best modified MRG32A equation is:

```
 M1 = 2.48312E+14
 M2 = 1.29046E+14
 for i=1:3
   ix = 1+fix((M2-2)*rand(1,4))
   iy = 1+fix((M2-2)*rand(1,4))
end
a11 = 938
a12 = 1474678
a13 = 2
a21 = 1001
a22 = 296
a23 = -862211

 for iter=1 to random numbers count
   ix(4) = a11*ix(3)+a12*ix(2)+a13*ix(1) mod M1
   iy(4) = a21*iy(3)+a22*iy(2)+a23*iy(1) mod M2
   iz = ix(4) - iy(4) mod M1
   if iz <= 0, iz = iz + M1;
   x(iter) = (iz+1)/(M1+1)
   ix(1:3) = ix(2:4)
   iy(1:3) = iy(2:4)
 end                                                    (3.1)
```

The first loop initializes the arrays ix and iy. You can use these values in the second set of equations for as many iterations as you need. The value x(iter) is the uniform random number generated in the range of 0 to 1(excluded) in each iteration.

DOCUMENT HISTORY

Date	*Version*	*Comments*
2/10/2023	1.00.00	Initial release.