Project 997 PRNGs Part 6
LCGM Squared Algorithms
By
Namir C. Shammas

1/ Introduction

This study looks at two versions of LCGM that use quadratic expressions of the previous random numbers to calculate new random numbers.

2/ LCGM Squared Method Algorithms (Version PSO)

The power method algorithm is defined as:

```
ix(1) = round(rand*seed,0);
ix(2) = a0+a1*ix(1) mod M
for i=1 to maximum number of random numbers
  ix(3) = a0+a1*ix(1)+a2*ix(2)^2 mod M
  x(i) = ix(3)/M
  ix(1:2) = ix(2:3)
end
```
(2.1)

This section looks at optimizing the values of a0, a1, and a2 used in equation 2.1. The array x() is the sought array of uniform random values in the range (0, 1]. The new random number x(i) obtains its value from the previous random integer values ix(1) and ix(2). The random number generating loops keeps the newer two values of the integer array ix(). M is the modulus value.

The approache that estimates the best coefficients for the power method 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:

```
maxElems=10000;
maxIters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res1.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^64-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res2.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^54-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res3.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^48-1;
bRound=true;
nDigits = 10;
```



```
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res4.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^40-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res5.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^32-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
% maxIters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res6.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^24-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res7.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
```



```
xM=2^16-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

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

r = 0.15;
rp = 1+r;
rm = 1-r;

maxElems=10000;
maxIters=100;
c = [79889,987318,3850,994554];
lb=round(c*rm,0);
ub=round(c*rp,0);
sFilename='res1b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^64-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
c = [80274,572095,367511,11];
lb=round(c*rm,0);
ub=round(c*rp,0);
sFilename='res2b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^54-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
c = [6862,636797,983034,818601];
lb=round(c*rm,0);
ub=round(c*rp,0);
sFilename='res3b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^48-1;
bRound=true;
nDigits = 10;
```



```
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
c = [36751,322033,1000000,270980];
lb=round(c*rm,0);
ub=round(c*rp,0);
sFilename='res4b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^40-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
c = [147,692578,126011,512462];
lb=round(c*rm,0);
ub=round(c*rp,0);
sFilename='res5b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^32-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
c = [147,692578,126011,512462];
lb=round(c*rm,0);
ub=round(c*rp,0);
sFilename='res6b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^24-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
c = [147,692578,126011,512462];
lb=round(c*rm,0);
ub=round(c*rp,0);
```



```
sFilename='res7b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^16-1;
bRound=true;
nDigits = 10;
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
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,xM,
bRound,nDigits)
global gmaxElems
global bestFactor
global M
global Xrnd
global doRounding
global numDigits

gmaxElems = maxElems;
M = xM;
doRounding = bRound;
numDigits = nDigits;
options =
optimoptions('particleswarm','SwarmSize',POSpopsize,'Display','off','MaxItera
tions',POSmaxIters,'FunctionTolerance',0.01);
% options =
optimoptions('particleswarm','Display','off','FunctionTolerance',0.01);
resMat=zeros(maxIters,6);
for i=1:maxIters
  bestFactor = 1e+99;
%   xrnd = round(i/(maxIters+1),n);
  x = particleswarm(@rng997Gen1,length(lb),lb,ub,options);
  resMat(i,1) = bestFactor;
  x = round(x, 0);
  resMat(i,2:5) = x(1:4);
  resMat(i,6) = Xrnd;

  fprintf("Itr: %d, Factor=%f, [", i, bestFactor);
  fprintf("%d, ", x);
  fprintf("%d]\n", Xrnd);
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(1:6) = {'Factor', 'IX', 'A0','A1', 'A2' ,'Xrnd'};
writetable(T1, sFilename, "Sheet", sSheetName);
pause(10);
c = cell(15,2);
c(1,1:2) = {'Max Elements', maxElems};
c(2,1:2) = {'Max Iters', maxIters};
c(3,1:2) = {'Xrndlow', lb(1)};
c(4,1:2) = {'Xrndhi', ub(1)};
c(5,1:2) = {'A01low', lb(2)};
c(6,1:2) = {'A0hi', ub(2)};
c(7,1:2) = {'A1low', lb(3)};
c(8,1:2) = {'A1hi', ub(3)};
c(9,1:2) = {'A2low', lb(4)};
c(10,1:2) = {'A2hi', ub(4)};
c(11,1:2) = {'PopSize', POSpopsize};
c(12,1:2) = {'PopMaxIters', POSmaxIters};
c(13,1:2) = {'M', M};
if doRounding
  c(14,1:2) = {'Rounded?', "True"};
  c(15,1:2) = {'Rounded', nDigits};
else
  c(14,1:2) = {'Rounded?', "False"};
  c(15,1:2) = {'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 of Reg997Gen1.m.

```
function factor = rng997Gen1(c)
%UNTITLED2 Summary of this function goes here
  global gmaxElems
  global bestFactor
  global Xrnd
  global M
  global doRounding
  global numDigits
```



```
  maxElems = gmaxElems;
  c = round(c, 0);
  ix = zeros(3,1);
  ix(1) = round(rand*c(1),0);
  Xrnd = ix(1);

  a0 = c(2);
  a1 = c(3);
  a2 = c(4);
  ix(2) = mod(a0+a1*ix(1),M);
  x=zeros(maxElems,1);
  for i=1:maxElems
    ix(3) = mod(a0+a1*ix(1)+a2*ix(2)^2,M);
    x(i) = ix(3)/M;
    ix(1:2) = ix(2:3);
  end
  if doRounding, x = round(x,numDigits); end
  factor=calcFactor(x,false);
  if isnan(factor), factor=65535; end

  if factor < bestFactor
    bestFactor = factor;
    bestX = x;
  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	res4
	Wide Range	Wide Range	Wide Range	Wide Range
Mean	46062.07635	39926.06588	112.1786809	112.1381499
Sdev	735.9119166	696.6596028	2.248421188	2.7802811
Min	45134.53882	37403.76704	106.0196619	101.9170744
Max	48289.02731	41610.13565	117.3939148	117.2354813
Range	3154.488489	4206.368608	11.3742529	15.31840689
Count	100	100	100	100
Conf	164.9474977	156.1494734	0.503961738	0.623172964
CI Upper	46227.02384	40082.21535	112.6826427	112.7613229
CI Lower	45897.12885	39769.9164	111.6747192	111.514977
Factor	45134.53882	37403.76704	106.0196619	101.9170744
IX	79889	80274	6862	36751
A0	987318	572095	636797	322033
A1	3850	367511	983034	1000000
A2	994554	11	818601	270980
Xrnd	53839	71103	4863	1900
Max Elements	10000	10000	10000	10000
Max Iters	100	100	100	100
Xrndlow	100	100	100	100
Xrndhi	100000	100000	100000	100000
A01low	11	11	11	11
A0hi	1000000	1000000	1000000	1000000
A1low	11	11	11	11
A1hi	1000000	1000000	1000000	1000000
A2low	11	11	11	11
A2hi	1000000	1000000	1000000	1000000
PopSize	250	250	250	250
PopMaxIters	350	350	350	350
M	1.84E+19	1.80E+16	2.81475E+14	1.09951E+12
Rounded?	TRUE	TRUE	TRUE	TRUE
Rounded	10	10	10	10

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

	res5	res6	res7
	Wide Range	Wide Range	Wide Range
Mean	112.0725989	112.5322403	110.8592244
Sdev	2.698548197	2.471990974	6.443903123
Min	103.7590022	103.5219427	79.25099673
Max	117.2751869	117.8631885	117.4083586
Range	13.51618465	14.34124581	38.15736187
Count	100	100	100
Conf	0.604853329	0.554072731	1.444338204
CI Upper	112.6774523	113.0863131	112.3035626
CI Lower	111.4677456	111.9781676	109.4148862
Factor	103.7590022	103.5219427	79.25099673
IX	147	67644	6107
A0	692578	11309	634823
A1	126011	184515	168165
A2	512462	194868	795158
Xrnd	84	28002	1121
Max Elements	10000	10000	10000
Max Iters	100	100	100
Xrndlow	100	100	100
Xrndhi	100000	100000	100000
A01low	11	11	11
A0hi	1000000	1000000	1000000
A1low	11	11	11
A1hi	1000000	1000000	1000000
A2low	11	11	11
A2hi	1000000	1000000	1000000
PopSize	250	250	250
PopMaxIters	350	350	350
M	4294967295	16777215	65535
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	res4b	res1b
Narrow Range	Narrow Range	Narrow Range	Narrow Range	Narrow Range
45993.98308	39410.58159	112.4758373	112.4539674	45993.98308
771.2855795	723.9127491	2.300342268	2.875536361	771.2855795
44105.56882	37814.49616	106.8695883	103.2268901	44105.56882
47374.75519	40914.32552	116.5633322	117.2524693	47374.75519
3269.186372	3099.829357	9.693743893	14.0255792	3269.186372
100	100	100	100	100
172.8761602	162.258001	0.515599343	0.644523504	172.8761602
46166.85924	39572.83959	112.9914367	113.0984909	46166.85924
45821.10692	39248.32359	111.960238	111.8094438	45821.10692
44105.56882	37814.49616	106.8695883	103.2268901	44105.56882
69085	76287	5833	38381	69085
1106528	486281	638208	273728	1106528
4388	398207	1038789	1000855	4388

912379	9	762730	280778	912379
67406	36670	2341	250	67406
10000	10000	10000	10000	10000
100	100	100	100	100
67906	68233	5833	31238	67906
91872	92315	7891	42264	91872
839220	486281	541277	273728	839220
1135416	657909	732317	370338	1135416
3273	312384	835579	850000	3273
4428	422638	1130489	1150000	4428
845371	9	695811	230333	845371
1143737	13	941391	311627	1143737
250	250	250	250	250
350	350	350	350	350
1.84E+19	1.80E+16	2.81475E+14	1.09951E+12	1.84E+19
TRUE	TRUE	TRUE	TRUE	TRUE
10	10	10	10	10

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

	res5b	res6b	res7b
	Narrow Range	Narrow Range	Narrow Range
Mean	111.8502337	112.3967238	110.8111228
Sdev	2.859265917	2.247067198	5.375349184
Min	101.8047429	104.6805082	87.60763399
Max	117.0027393	117.2057858	116.7325333
Range	15.19799647	12.52527762	29.12489932
Count	100	100	100
Conf	0.640876643	0.503658255	1.204832232
CI Upper	112.4911103	112.900382	112.015955
CI Lower	111.209357	111.8930655	109.6062906
Factor	101.8047429	104.6805082	87.60763399
IX	128	77791	5877
A0	588691	12304	658575
A1	114476	212192	188768
A2	544632	174063	793873
Xrnd	107	68311	3220
Max Elements	10000	10000	10000
Max Iters	100	100	100
Xrndlow	125	57497	5191
Xrndhi	169	77791	7023
A01low	588691	9613	539600
A0hi	796465	13005	730046
A1low	107109	156838	142940
A1hi	144913	212192	193390
A2low	435593	165638	675884
A2hi	589331	224098	914432
PopSize	250	250	250
PopMaxIters	350	350	350
M	4294967295	16777215	65535
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;
c=[67406,1106528,4388,912379];
sFilename='res1c.xlsb';
xM=2^64-1;
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c=[36670,486281,398207,9];
sFilename='res2c.xlsb';
xM=2^54-1;
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c=[2341,638208,1038789,762730];
sFilename='res3c.xlsb';
xM=2^48-1;
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c=[2501,273728,1000855,280778];
sFilename='res4c.xlsb';
xM=2^40-1;
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c=[107,588691,114476,544632];
sFilename='res5c.xlsb';
xM=2^32-1;
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c=[107,588691,114476,544632];
sFilename='res6c.xlsb';
```



```
xM1=2^24-1;
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c=[107,588691,114476,544632];
sFilename='res7c.xlsb';
xM1=2^16-1;
bRound=true;
nDigits = 10;
doAll2(maxElems,maxIters,c,sFilename,xM,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,xM,bRound,nDigits)
global gmaxElems
global doRounding
global numDigits
global M

gmaxElems = maxElems;
M = xM;
doRounding = bRound;
numDigits = nDigits;
resMat=zeros(maxIters,7);
for i=1:maxIters
  [factor,XYrnd] = rng997Gen2(c);
  resMat(i,1) = factor;
  c = round(c, 0);
  resMat(i,2:5) = c(1:4);
  resMat (i,6) = M;
  resMat(i,7) = XYrnd;

  fprintf("Itr: %d, Factor=%f, [", i, factor);
  fprintf("%d, ", c);
  fprintf("%d]\n", XYrnd);
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(1:7) = {'Factor', 'IX', 'A0', 'A1', 'A2' , 'M',
'Xrnd'};
```



```
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 of file rng997Gen2.m.

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

  maxElems = gmaxElems;
  c = round(c, 0);
  ix = zeros(3,1);
  ix(1) = round(rand*c(1),0);
  XYrnd = ix(1);

  a0 = c(2);
  a1 = c(3);
  a2 = c(4);
  ix(2) = mod(a0+a1*ix(1),M);
  x=zeros(maxElems,1);
  for i=1:maxElems
    ix(3) = mod(a0+a1*ix(1)+a2*ix(2)^2,M);
    x(i) = ix(3)/M;
    ix(1:2) = ix(2:3);
  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 of 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	res4c
	Random Seed	Random Seed	Random Seed	Random Seed
Mean	46558.71626	39230.16654	147.6587563	147.9650123
Sdev	3956.886492	2799.25667	11.98806947	11.9546134
Min	44067.44745	37808.78644	115.7791671	111.9146601
Max	57958.69282	96036.24682	205.1406386	210.221311
Range	13891.24538	58227.46038	89.36147151	98.30665092
Count	1000000	1000000	1000000	1000000
Conf	8.868976175	6.274261536	0.026870092	0.026795103
CI Upper	46567.58523	39236.4408	147.6856263	147.9918074
CI Lower	46549.84728	39223.89228	147.6318862	147.9382172
Factor	44067.44745	37808.78644	115.7791671	111.9146601
IX	67406	36670	2341	2501
A0	1106528	486281	638208	273728
A1	4388	398207	1038789	1000855
A2	912379	9	762730	280778
M	1.84467E+19	1.80144E+16	2.81475E+14	1.09951E+12
Xrnd	113	33484	507	1176

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

	res5c	res6c	res7c
	Random Seed	Random Seed	Random Seed
Mean	147.4482878	147.8049076	147.6084877

Sdev	12.63537406	11.75237684	11.43154576
Min	117.5720673	110.0113383	117.4702964
Max	185.0903215	206.1843207	195.1487125
Range	67.51825416	96.17298241	77.67841609
Count	1000000	1000000	1000000
Conf	0.028320962	0.02634181	0.025622698
CI Upper	147.4766087	147.8312494	147.6341104
CI Lower	147.4199668	147.7785658	147.582865
Factor	117.5720673	110.0113383	117.4702964
IX	107	68311	3220
A0	588691	12304	658575
A1	114476	212192	188768
A2	544632	174063	793873
M	4294967295	4294967295	4294967295
Xrnd	31	6628	2352

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

The column titled res6c in Table 2.3b has the lowest upper mean value. The best modified power method equation is:

```
M = 4294967295
a0 = 658575
a1 = 188768
a2 = 793873
 ix(1) = round(rand*3220, 0);
 ix(2) = a0+a1*ix(1) mod M
 for i=1 to maximum number of random numbers
   ix(3) = a0+a1*ix(1)+a2*ix(2)^2 mod M
   x(i) = ix(3)/M
   ix(1:2) = ix(2:3)
 end                                                    (2.2)
```

The value x(i) is the uniform random number generated in the range of 0 to 1 (excluded) in each iteration.

3/ LCGM SQUARED ALGORITHM (VERSION VER2 PSO)

The power method algorithm is defined as:

```
 ix(1) = round(rand*seed0);
 ix(2) = a0+a1*ix(1) mod M
 for i=1 to maximum number of random numbers
   ix(3) = a0+a1*ix(1) ^2 +a2*ix(2) mod M
```



```
  x(i) = ix(3)/M
  ix(1:2) = ix(2:3)
end                                                    (3.1)
```

The difference between equations 2.1 and 3.1 is that in equation 2.1 the value for x(2) is squared. In equation 3.1 it is value for x(1) that is squared.

This section looks at optimizing the values of a0, a1, and a2 used in equation 3.1. The array x() is the sought array of uniform random values in the range (0, 1]. The new random number x(i) obtains its value from two previous random integer values ix(1) and ix(2). The random number generating loops keeps the newer three values of the integer array ix(). M is the modulus value.

The approache that estimates the best coefficients for the power method 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:

```
maxElems=10000;
max
Iters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res1.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^64-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)
```



```
maxElems=10000;
maxIters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res2.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^54-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res3.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^48-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res4.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^40-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res5.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^32-1;
bRound=true;
nDigits = 10;
```



```
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res6.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1=2^24-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res7.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1=2^16-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

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

r= 0.15;
rp = 1 + r;
rm = 1 - r;
maxElems=10000;
maxIters=100;
c = [27127,1000000,1000000,11];
lb=round(c*rm,0);
ub=round(c*rp,0);
sFilename='res1b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^64-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
```



```
c = [50511,1000000,11,35023];
lb=round(c*rm,0);
ub=round(c*rp,0);
sFilename='res2b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^54-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
c = [84452,573659,483402,263049];
lb=round(c*rm,0);
ub=round(c*rp,0);
sFilename='res3b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^48-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
c = [80137,507057,327887,235520];
lb=round(c*rm,0);
ub=round(c*rp,0);
sFilename='res4b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^40-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
c = [39608,128678,741654,130176];
lb=round(c*rm,0);
ub=round(c*rp,0);
sFilename='res5b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^32-1;
bRound=true;
```



```
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
c = [29129,1000000,956697,752822];
lb=round(c*rm,0);
ub=round(c*rp,0);
sFilename='res6b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1=2^24-1;
bRound=true;
nDigits = 10;
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
c = [24360,194207,193839,864890];
lb=round(c*rm,0);
ub=round(c*rp,0);
sFilename='res7b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1=2^16-1;
bRound=true;
nDigits = 10;
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

% system('shutdown /s')
```

Listing 3.1. The listing of file do.m.

Listing 3.2 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.

```
maxElems=10000;
max
Iters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res1.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^64-1;
```



```
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res2.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^54-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res3.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^48-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res4.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^40-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
```



```
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res5.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^32-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res6.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1=2^24-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
lb=[100 11 11 11];
ub=[100000 1000000 1000000 1000000];
sFilename='res7.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1=2^16-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

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

r= 0.15;
rp = 1 + r;
rm = 1 - r;
maxElems=10000;
maxIters=100;
c = [27127,1000000,1000000,11];
lb=round(c*rm,0);
ub=round(c*rp,0);
```



```
sFilename='res1b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^64-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
c = [50511,1000000,11,35023];
lb=round(c*rm,0);
ub=round(c*rp,0);
sFilename='res2b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^54-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
c = [84452,573659,483402,263049];
lb=round(c*rm,0);
ub=round(c*rp,0);
sFilename='res3b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^48-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
c = [80137,507057,327887,235520];
lb=round(c*rm,0);
ub=round(c*rp,0);
sFilename='res4b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^40-1;
```



```
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
c = [39608,128678,741654,130176];
lb=round(c*rm,0);
ub=round(c*rp,0);
sFilename='res5b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=2^32-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
c = [29129,1000000,956697,752822];
lb=round(c*rm,0);
ub=round(c*rp,0);
sFilename='res6b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1=2^24-1;
bRound=true;
nDigits = 10;
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=100;
c = [24360,194207,193839,864890];
lb=round(c*rm,0);
ub=round(c*rp,0);
sFilename='res7b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1=2^16-1;
bRound=true;
nDigits = 10;
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

% system('shutdown /s')
```


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 bestFactor
  global Xrnd
  global M
  global doRounding
  global numDigits

  maxElems = gmaxElems;
  c = round(c, 0);
  ix = zeros(3,1);
  ix(1) = round(rand*c(1),0);
  Xrnd = ix(1);
  a0 = c(2);
  a1 = c(3);
  a2 = c(4);
  ix(2) = mod(a0+a1*ix(1),M);
  x=zeros(maxElems,1);
  for i=1:maxElems
    ix(3) = mod(a0+a1*ix(1)^2+a2*ix(2),M);
    x(i) = ix(3)/M;
    ix(1:2) = ix(2:3);
  end
  if doRounding, x = round(x,numDigits); end
  factor=calcFactor(x,false);
  if isnan(factor), factor=65535; end

  if factor < bestFactor
    bestFactor = factor;
    bestX = x;
  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.

Tables 3.1a and 3.1b show the results of the wide-trust region optimization.

	res1	res2	res3	res4
	Wide Range	Wide Range	Wide Range	Wide Range
Mean	128267.3209	56400.30929	112.5354831	112.0927421
Sdev	7804.567913	5186.64644	2.374539653	2.244310779
Min	122693.7282	47016.46697	103.8859558	106.6552764
Max	139090.9445	65431.29423	117.4740267	116.263131
Range	16397.21632	18414.82726	13.58807084	9.607854583
Count	100	100	100	100
Conf	1749.317981	1162.536348	0.532229966	0.50304043
CI Upper	130016.6389	57562.84564	113.0677131	112.5957825
CI Lower	126518.0029	55237.77295	112.0032531	111.5897017
Factor	122693.7282	47016.46697	103.8859558	106.6552764
IX	27127	50511	84452	80137
A0	1000000	1000000	573659	507057
A1	1000000	11	483402	327887
A2	11	35023	263049	235520
Xrnd	7096	31722	85886	71737
Max Elements	10000	10000	10000	10000
Max Iters	100	100	100	100
Xrndlow	100	100	100	100
Xrndhi	100000	100000	100000	100000
A01low	11	11	11	11
A0hi	1000000	1000000	1000000	1000000

A1low	11	11	11	11
A1hi	1000000	1000000	1000000	1000000
A2low	11	11	11	11
A2hi	1000000	1000000	1000000	1000000
PopSize	250	250	250	250
PopMaxIters	350	350	350	350
M	1.84E+19	1.80144E+16	2.81475E+14	1.09951E+12
Rounded?	TRUE	TRUE	TRUE	TRUE
Rounded	10	10	10	10

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

	res5	res6	res7
	Wide Range	Wide Range	Wide Range
Mean	112.5005362	112.3148022	112.4799776
Sdev	2.524830854	2.555291447	2.427610263
Min	103.8476213	103.2985543	104.1389801
Max	118.0497282	116.9439587	117.81347
Range	14.20210685	13.64540442	13.67448996
Count	100	100	100
Conf	0.565916276	0.572743722	0.544125227
CI Upper	113.0664525	112.8875459	113.0241028
CI Lower	111.93462	111.7420584	111.9358524
Factor	103.8476213	103.2985543	104.1389801
IX	39608	29129	24360
A0	128678	1000000	194207
A1	741654	956697	193839
A2	130176	752822	864890
Xrnd	23691	10718	3779
Max Elements	10000	10000	10000
Max Iters	100	100	100
Xrndlow	100	100	100
Xrndhi	100000	100000	100000
A01low	11	11	11
A0hi	1000000	1000000	1000000
A1low	11	11	11
A1hi	1000000	1000000	1000000
A2low	11	11	11
A2hi	1000000	1000000	1000000
PopSize	250	250	250
PopMaxIters	350	350	350
M	4294967295	4294967295	4294967295
Rounded?	TRUE	TRUE	TRUE
Rounded	10	10	10

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

Table 3.2a and 3.2b show the results of the narrow-trust region optimization.

	res1b	res2b	res3b	res4b
	Narrow Range	Narrow Range	Narrow Range	Narrow Range
Mean	121328.2671	49473.89777	112.3081939	112.0475236
Sdev	5424.792311	2473.056994	2.485696975	2.524014336
Min	91169.0941	45490.82815	103.3403327	102.508672

Max	122549.1457	54981.37313	118.3168519	116.6826268
Range	31380.05164	9490.54498	14.97651919	14.17395473
Count	100	100	100	100
Conf	1215.914428	554.3116691	0.557144798	0.565733262
CI Upper	122544.1815	50028.20944	112.8653387	112.6132569
CI Lower	120112.3526	48919.58611	111.7510491	111.4817904
Factor	91169.0941	45490.82815	103.3403327	102.508672
IX	31196	58088	77379	87803
A0	1085465	1005670	569515	496003
A1	1111091	9	499562	351752
A2	11	37093	255790	248090
Xrnd	14196	46905	22193	57927
Max Elements	10000	10000	10000	10000
Max Iters	100	100	100	100
Xrndlow	23058	42934	71784	68116
Xrndhi	31196	58088	97120	92158
A01low	850000	850000	487610	430998
A0hi	1150000	1150000	659708	583116
A1low	850000	9	410892	278704
A1hi	1150000	13	555912	377070
A2low	9	29770	223592	200192
A2hi	13	40276	302506	270848
PopSize	250	250	250	250
PopMaxIters	350	350	350	350
M	1.84E+19	1.80E+16	2.81475E+14	1.09951E+12
Rounded?	TRUE	TRUE	TRUE	TRUE
Rounded	10	10	10	10

Table 3.2a. The results of the narrow-trust region optimization.

	res5b	res6b	res7b
	Narrow Range	Narrow Range	Narrow Range
Mean	112.2944833	112.3202751	111.6258855
Sdev	2.33868273	2.247361296	2.454008282
Min	106.5411114	104.488291	105.9908005
Max	117.0769167	116.7913485	116.2009202
Range	10.53580535	12.30305754	10.21011963
Count	100	100	100
Conf	0.524192985	0.503724174	0.550042086
CI Upper	112.8186763	112.8239993	112.1759276
CI Lower	111.7702903	111.8165509	111.0758434
Factor	106.5411114	104.488291	105.9908005
IX	39058	33498	25752
A0	118192	884181	183052
A1	660966	858793	222915
A2	135761	654486	892545
Xrnd	20849	3632	20541
Max Elements	10000	10000	10000
Max Iters	100	100	100
Xrndlow	33667	24760	20706
Xrndhi	45549	33498	28014

A01low	109376	850000	165076
A0hi	147980	1150000	223338
A1low	630406	813192	164763
A1hi	852902	1100202	222915
A2low	110650	639899	735157
A2hi	149702	865745	994623
PopSize	250	250	250
PopMaxIters	350	350	350
M	4294967295	4294967295	4294967295
Rounded?	TRUE	TRUE	TRUE
Rounded	10	10	10

Table 3.2b. The results of the narrow-trust 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 = [14196,1085465,1111091,11];
sFilename='res1c.xlsb';
xM=2^64-1;
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c = [46905,1005670,9,37093];
sFilename='res2c.xlsb';
xM=2^54-1;
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c = [22193,569515,499562,255790];
sFilename='res3c.xlsb';
xM=2^48-1;
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c = [57927,496003,351752,248090];
sFilename='res4c.xlsb';
xM=2^40-1;
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,xM,bRound,nDigits)

maxElems=10000;
```



```
maxIters=1000000;
c = [20849,118192,660966,135761];
sFilename='res5c.xlsb';
xM=2^32-1;
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c = [3632,884181,858793,      654486];
sFilename='res6c.xlsb';
xM1=2^24-1;
bRound=true;
nDigits = 10;
doAll2(maxElems,maxIters,c,sFilename,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c = [20541,183052,222915,892545];
sFilename='res7c.xlsb';
xM1=2^16-1;
bRound=true;
nDigits = 10;
doAll2(maxElems,maxIters,c,sFilename,xM,bRound,nDigits)

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

% system('shutdown /s')
```

Listing 3.4. The source code of file do2.m.

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

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

gmaxElems = maxElems;
M = xM;
doRounding = bRound;
numDigits = nDigits;
resMat=zeros(maxIters,6);
for i=1:maxIters
  [factor,Xrnd] = rng997Gen2(c);
  resMat(i,1) = factor;
  c = round(c, 0);
  resMat(i,2:5) = c(1:4);
  resMat(i,6) = Xrnd;

  fprintf("Itr: %d, Factor=%f, [", i, factor);
  fprintf("%d, ", c);
  fprintf("%d]\n", Xrnd);
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(1:6) = {'Factor', 'IX', 'A0','A1', 'A2' ,'Xrnd'};
writetable(T1, sFilename, "Sheet", "Sheet1");

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

Listing 3.5. The source code of file doAll2.m.

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

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

  maxElems = gmaxElems;
  c = round(c, 0);
  ix = zeros(3,1);
  ix(1) = round(rand*c(1),0);
  Xrnd = ix(1);
  a0 = c(2);
  a1 = c(3);
  a2 = c(4);
  ix(2) = mod(a0+a1*ix(1),M);
  x=zeros(maxElems,1);
  for i=1:maxElems
    ix(3) = mod(a0+a1*ix(1)^2+a2*ix(2),M);
    x(i) = ix(3)/M;
    ix(1:2) = ix(2:3);
  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.

Tables 3.3a and 3.3b show the results of the penalty factor statistics for the different versions of the algorithm.

	res1c	res2c	res3c	res4c
	Random Seed	Random Seed	Random Seed	Random Seed
Mean	98210.00274	57028.71936	147.6852617	147.7832947
Sdev	7396.269874	11257.88835	11.75190742	11.72332197
Min	91168.91057	45476.62449	103.3403327	102.508672
Max	106012.1063	73524.52853	221.5771966	209.6702707
Range	14843.19572	28047.90404	118.2368639	107.1615986
Count	1000000	1000000	1000000	1000000
Conf	16.57801947	25.23346165	0.026340757	0.026276686
CI Upper	98226.58076	57053.95282	147.7116025	147.8095713
CI Lower	98193.42472	57003.4859	147.658921	147.757018
Factor	91168.91057	45476.62449	103.3403327	102.508672
IX	14196	46905	22193	57927
A0	1085465	1005670	569515	496003

A1	1111091	9	499562	351752
A2	11	37093	255790	248090
M	3053	31044	19704	32814
Xrnd	98210.00274	57028.71936	147.6852617	147.7832947

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

	res5c	res6c	res7c
	Random Seed	Random Seed	Random Seed
Mean	147.804857	147.0998616	147.7074167
Sdev	11.9316947	11.83522271	11.68714137
Min	106.5269793	114.5317024	105.9908005
Max	214.4601131	197.5658746	202.2172447
Range	107.9331338	83.03417222	96.22644414
Count	1000000	1000000	1000000
Conf	0.026743733	0.0265275	0.026195591
CI Upper	147.8316007	147.1263891	147.7336123
CI Lower	147.7781132	147.0733341	147.6812212
Factor	106.5269793	114.5317024	105.9908005
IX	20849	3632	20541
A0	118192	884181	183052
A1	660966	858793	222915
A2	135761	654486	892545
M	500	575	1545
Xrnd	147.804857	147.0998616	147.7074167

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

The column titled res6c in Table 3.3b has the lowest upper mean value. The best modified power method equation is:

M = 2^24-1
a0 = 884181
a1 = 858793
a2 = 654486
ix(1) = round(rand*3632, 0);
ix(2) = a0+a1*ix(1) mod M
for i=1 to maximum number of random numbers
  ix(3) = a0+a1*ix(1)^2 +a2*ix(2) mod M
  x(i) = ix(3)/M
  ix(1:2) = ix(2:3)
end (3.2)

The value x(i) 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.

