Project 997 PRNGs Part 3
ACORN Variant Algorithms
By
Namir C. Shammas

1/ Introduction

This study looks at two flavors of the ACORN algorithm variants. The first flavor uses four parameters, while the second one uses six parameters.

2/ ACORN Algorithms (Version ACORN PSO)

The ACORN algorithm is defined as:

```
r  = function Acorn(k,ix1(),ix2(), a11, a12, a21, a22)
% ix1() and ix2() are arrays with k+1 sizes and provide the function
% with an input of k elements each.
    M = 2^64 - 253
    for i=2:k
      ip1 = i + 1
      ix1(ip1) = a11*ix1(ip1) + a12*ix1(i)
      ix2(ip1) = a21*ix2(ip1) + a22*ix2(i)
      if ix2(ip1) > M
        ix2(ip1) = mod(ix2(ip1), M)
        ix1(ip1) = ix1(ip1) + 1
      end
      if ix1(ip1) > M, ix1(ip1) = ix1(ip1) mod M)
    end
    r = mod(ix1(kp1) + ix2(kp1), M) / M
    ix1(1:k) = ix1(2:kp1)
    ix2(1:k) = ix2(2:kp1)                                        (2.1)
end
```


This section looks at a better version of the ACORN algorithms that uses a11, a12, a21, a22, and arrays ix1 and ix2 that appear in equation 2.1. The calculations are all based on the value of k = 17 and M = 2^64 - 253. The variable r is the sought uniform random value in the range (0, 1].

The approache that estimates the best coefficients for a ACORN 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 ix1 and ix2. 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 = 2^10;
xM2 = 2^11;

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

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

maxElems=10000;
maxIters=100;
lb=[xM1 xM1 1 1 1 1];
ub=[xM2 xM2 3 3 3 3];
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=[xM1 xM1 1 1 1 1];
ub=[xM2 xM2 50 50 50 50];
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=[xM1 xM1 1 1 1 1];
ub=[xM2 xM2 100 100 100 100];
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=[xM1 xM1 1 1 1 1];
ub=[xM2 xM2 150 150 150 150];
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=[xM1 xM1 1 1 1 1];
ub=[xM2 xM2 200 200 200 200];
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)
```



```
% ---------------------------------------------------------------------------

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

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

maxElems=10000;
maxIters=100;
x = [1024,1862.890818,3,3,1.623492618,3];
lb = [rm*1024,rm*1862.890818,1,1,rm*1.623492618,1];
ub = [rp*1024,rp*1862.890818,5,5,rp*1.623492618,5];
lb = round(lb,0);
ub = round(ub,0);
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 = [1467.227863,2048,43.69714684,36.41318664,37.0598521,8.939806147];
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
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 =
[1462.005557,1469.746096,98.86282552,71.36861391,16.80578153,3.796809624];
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
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 = [1464.879714,1024,120.432172,31.42460212,6.447786846,137.2491565];
r = 0.15;
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
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 = [1507.523062,1619.171286,1,200,200,200];
lb = [rm*1507.523062,rm*1619.171286,0,rm*200,rm*200,rm*200];
ub = [rp*1507.523062,rp*1619.171286,3,rp*200,rp*200,rp*200];
lb = round(lb,0);
ub = round(ub,0);
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 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, bIsRound, nDigits)
global gmaxElems
global bestFactor
global bRound
global numDigits

gmaxElems = maxElems;
bRound = bIsRound;
numDigits = nDigits;

bResetResMat = true;
% output file exists?
```

```
if isfile(sFilename)
  resMat = readmatrix(sFilename,"Sheet", sSheetName,"Range","A2:G101");
  [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,7);
  nStart = 1;
end
options = optimoptions('particleswarm', 'SwarmSize', POSpopsize, 'Display',
'off', 'MaxIterations', POSmaxIters, 'FunctionTolerance', 0.01);

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

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

  resMat = sortrows(resMat,1);

  T1 = array2table(resMat);
  T1.Properties.VariableNames(1:7) = {'Factor', 'iX1','iX2', 'A11' ,'A12',
'A21' ,'A22'};
  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(1:7) = {'Factor', 'iX1','iX2', 'A11' ,'A12',
'A21' ,'A22'};
writetable(T1, sFilename, "Sheet", sSheetName);

c = cell(18,2);
```

```
c(1,1:2) = {'Max Elements', maxElems};
c(2,1:2) = {'Max Iters', maxIters};
c(3,1:2) = {'iX1Low', lb(1)};
c(4,1:2) = {'iX1Hi', ub(1)};
c(5,1:2) = {'iX2Low', lb(2)};
c(6,1:2) = {'iX2Hi', ub(2)};
c(7,1:2) = {'a11Low', lb(3)};
c(8,1:2) = {'a11Hi', ub(3)};
c(9,1:2) = {'a12Low', lb(4)};
c(10,1:2) = {'a12Hi', ub(4)};
c(11,1:2) = {'a21Low', lb(5)};
c(12,1:2) = {'a21Hi', ub(5)};
c(13,1:2) = {'a22Low', lb(6)};
c(14,1:2) = {'a22Hi', ub(6)};
c(15,1:2) = {'PopSize', POSpopsize};
c(16,1:2) = {'PopMaxIters', POSmaxIters};
if bRound
  c(17,1:2) = {'Rounded?', "True"};
  c(18,1:2) = {'Rounded', numDigits};
else
  c(17,1:2) = {'Rounded?', "False"};
  c(18,1:2) = {'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 fof file Reg997Gen1.m.

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

  M = 2^64 - 253;
  k = 17;
  kp1 = k + 1;
  maxElems = gmaxElems;
  numDigits = 10;
  x=zeros(maxElems,1);
  ix1 = zeros(kp1,1);
  ix2 = zeros(kp1,1);
  rng('shuffle','twister');
  for i=1:kp1
    ix1(i) = 1 + fix(rand*c(1));
    ix2(i) = 1 + fix(rand*c(2));
  end
  a11 = c(3);
  a12 = c(4);
```

```
  a21 = c(5);
  a22 = c(6);
  for iter=1:maxElems
    for i=2:k
      ip1 = i + 1;
      ix1(ip1) = a11*ix1(ip1) + a12*ix1(i);
      ix2(ip1) = a21*ix2(ip1) + a22*ix2(i);
      if ix2(ip1) > M
        ix2(ip1) = mod(ix2(ip1), M);
        ix1(ip1) = ix1(ip1) + 1;
      end
      if ix1(ip1) > M, ix1(ip1) = mod(ix1(ip1),M); end
    end
    x(iter) = mod(ix1(kp1) + ix2(kp1), M) / M;
    ix1(1:k) = ix1(2:kp1);
    ix2(1:k) = ix2(2:kp1);
  end
  if bRound, 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 2.3. The listing of file rng997Gen1.m.

Table 2.1 shows the results of the wide-trust region optimization.

	res1	res2	res3	res4	res5
	Wide Range	Wide Range	Wide Range	Wide Range	Wide Range
Mean	112.084969	112.4674059	112.3573103	112.4059805	109.7182066
Sdev	2.721490195	2.116759834	2.577599572	2.204744271	7.841496295
Min	102.545895	106.1583851	105.6441433	105.8957352	82.40590661
Max	116.7845719	116.1867448	117.3161296	117.132473	117.3952387
Range	14.23867688	10.02835977	11.67198632	11.23673782	34.98933212
Count	100	100	100	100	100
Conf	0.609995555	0.474451127	0.577743871	0.494171982	1.757595118
CI Upper	112.6949645	112.941857	112.9350541	112.9001525	111.4758017
CI Lower	111.4749734	111.9929547	111.7795664	111.9118085	107.9606114
Max Elements	10000	10000	10000	10000	10000
Max Iters	100	100	100	100	100
iX1Low	1024	1024	1024	1024	1024
iX1Hi	2048	2048	2048	2048	2048
iX2Low	1024	1024	1024	1024	1024
iX2Hi	2048	2048	2048	2048	2048
a11Low	1	1	1	1	1
a11Hi	3	50	100	150	200
a12Low	1	1	1	1	1
a12Hi	3	50	100	150	200
a21Low	1	1	1	1	1
a21Hi	3	50	100	150	200
a22Low	1	1	1	1	1
a22Hi	3	50	100	150	200
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 2.1. The results of the wide-trust region optimization.

Table 2.2 shows the results of the narrow-trust region optimization.

	res1b	res2b	res3b	res4b	res5b
	Narrow Range	Narrow Range	Narrow Range	Narrow Range	Narrow Range
Mean	112.1744293	112.1818402	112.2949679	112.3205516	111.9358306
Sdev	2.609844207	2.439007008	2.460986637	2.254017315	2.6241158

Min	103.4693391	103.1308155	105.3119149	104.3264513	105.3772466
Max	116.1418153	116.6687427	116.966336	116.0519256	117.1880697
Range	12.67247619	13.53792723	11.65442109	11.72547437	11.8108231
Count	100	100	100	100	100
Conf	0.584971192	0.546679696	0.551606216	0.505216056	0.588170031
CI Upper	112.7594005	112.7285199	112.8465741	112.8257676	112.5240006
CI Lower	111.5894581	111.6351605	111.7433617	111.8153355	111.3476606
Max Elements	10000	10000	10000	10000	10000
Max Iters	100	100	100	100	100
iX1Low	870	1247	1243	1245	1281
iX1Hi	1178	1687	1681	1685	1734
iX2Low	1583	1741	1249	870	1376
iX2Hi	2142	2355	1690	1178	1862
a11Low	1	37	84	102	0
a11Hi	5	50	114	138	3
a12Low	1	31	61	27	170
a12Hi	5	42	82	36	230
a21Low	1	32	14	5	170
a21Hi	2	43	19	7	230
a22Low	1	8	3	117	170
a22Hi	5	10	4	158	230
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 2.2. 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 = [1002.25479,1620.202733,4.746494933,2.323887575,1.056714686,4.231838479];
sFilename='res1c.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
bRound=true;
nDigits = 10;
doAll2(maxElems,maxIters,c,sFilename,sSheetName,nDigits)


maxElems=10000;
maxIters=1000000;
c =
[1529.174193,1976.660417,45.52641792,33.68297548,33.22348174,8.390088965];
sFilename='res2c.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
```

```
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,sSheetName,nDigits)

maxElems=10000;
maxIters=1000000;
c = [1674.373178,1437.570785,114,76.05910963,17.45387204,3.528187015];
sFilename='res3c.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,sSheetName,nDigits)

maxElems=10000;
maxIters=1000000;
c = [1252.11116,1101.890998,102,35.61770003,6.155585798,153.5956953];
sFilename='res4c.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,sSheetName,nDigits)

maxElems=10000;
maxIters=1000000;
c = [1281,1862,1.712170986,207.4305176,201.420982,186.8829569];
sFilename='res5c.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,sSheetName,nDigits)

system('shutdown /s')
```

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

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

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

gmaxElems = maxElems;
numDigits = nDigits;

bResetResMat = true;
% output file exists?
if isfile(sFilename)
  resMat = readmatrix(sFilename,"Sheet", sSheetName,"Range","A2:G101");
  [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,7);
  nStart = 1;
end

for i=nStart:maxIters
  factor = rng997Gen2(c);
  resMat(i,1) = factor;
  resMat(i,2:7) = c;

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

  resMat = sortrows(resMat,1);

  T1 = array2table(resMat);
  T1.Properties.VariableNames(1:7) = {'Factor', 'iX1','iX2', 'A11' ,'A12',
'A21' ,'A22'};
  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(1:7) = {'Factor', 'iX1','iX2', 'A11' ,'A12',
'A21' ,'A22'};
writetable(T1, sFilename, "Sheet", sSheetName);


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

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

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

```
function factor = rng997Gen2(c)
%UNTITLED2 Summary of this function goes here
  global gmaxElems
  global numDigits;
```

```
  M = 2^64 - 253;
  k = 17;
  kp1 = k + 1;
  maxElems = gmaxElems;
  numDigits = 10;
  x=zeros(maxElems,1);
  ix1 = zeros(kp1,1);
  ix2 = zeros(kp1,1);
  rng('shuffle','twister');
  for i=1:kp1
    ix1(i) = 1 + fix(rand*c(1));
    ix2(i) = 1 + fix(rand*c(2));
  end
  a11 = c(3);
  a12 = c(4);
  a21 = c(5);
  a22 = c(6);
  for iter=1:maxElems
    for i=2:k
      ip1 = i + 1;
      ix1(ip1) = a11*ix1(ip1) + a12*ix1(i);
      ix2(ip1) = a21*ix2(ip1) + a22*ix2(i);
      if ix2(ip1) > M
        ix2(ip1) = mod(ix2(ip1), M);
        ix1(ip1) = ix1(ip1) + 1;
      end
      if ix1(ip1) > M, ix1(ip1) = mod(ix1(ip1),M); end
    end
    x(iter) = mod(ix1(kp1) + ix2(kp1), M) / M;
    ix1(1:k) = ix1(2:kp1);
    ix2(1:k) = ix2(2:kp1);
  end
  x = round(x,numDigits);
  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.

Table 2.3 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	147.8244017	147.7586482	147.7676064	147.7430432	147.7873734
Sdev	11.77242451	11.77430184	11.77906855	11.76932798	11.7962554
Min	102.1892678	103.3723268	101.4726884	102.0396653	99.02810452
Max	232.5451879	222.7421611	222.8796873	232.9123491	240.8634152
Range	130.3559201	119.3698343	121.4069989	130.8726839	141.8353107
Count	1000000	1000000	1000000	1000000	1000000
Conf	0.026386744	0.026390952	0.026401636	0.026379804	0.026440159
CI Upper	147.8507885	147.7850391	147.7940081	147.769423	147.8138135
CI Lower	147.798015	147.7322572	147.7412048	147.7166634	147.7609332
Max Elements	10000	10000	10000	10000	10000
Max Iters	1000000	1000000	1000000	1000000	1000000

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

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

k = 17
M = 2^64 - 253

```
for i=1:k+1
  ix1(i) = 1 + fix(rand*1252.11116)
  ix2(i) = 1 + fix(rand*1101.890998)
end

for iter=1 to random numbers count
  for i=2 to k
    ip1 = i + 1
    ix1(ip1) = 102*ix1(ip1) + 35.61770003*ix1(i)
    ix2(ip1) = 6.155585798*ix2(ip1) + 153.5956953*ix2(i)
    if ix2(ip1) > M
      ix2(ip1) = ix2(ip1) mod M
      ix1(ip1) = ix1(ip1) + 1
    end
    if ix1(ip1) > M, ix1(ip1) = ix1(ip1) mod M)
  end
  x(iter) = (ix1(kp1) + ix2(kp1) mod M) / M
  ix1(1:k) = ix1(2:kp1)
  ix2(1:k) = ix2(2:kp1)
end                                                    (2.2)
```

The first loop initializes the arrays ix1 and ix2. 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/ ACORN Algorithms (Version ACCORN 3 PSO)

Equation 2.1 shows a general form of the original ACORN algorithm. In this section I present and extended version ACORN algorithm as defined in:

```
r = function Acorn(k,ix1(),ix2(), a11, a12, a21, a22,a13,a23)
% ix1() and ix2() are arrays with k+1 sizes and provide the function
% with an input of k elements each.
  k = 7
  M = 2^64 - 253
  for i=2:k
    ip1 = i + 1
    im1 = i - 1
```

```
      ix1(ip1) = a11*ix1(ip1) + a12*ix1(i) + a13*ix1(im1)
      ix2(ip1) = a21*ix2(ip1) + a22*ix2(i) + a23*ix2(im1)
      if ix2(ip1) > M
        ix2(ip1) = mod(ix2(ip1), M)
        ix1(ip1) = ix1(ip1) + 1
      end
      if ix1(ip1) > M, ix1(ip1) = ix1(ip1) mod M)
    end
    r= mod(ix1(kp1) + ix2(kp1), M) / M
    ix1(1:k) = ix1(2:kp1)
    ix2(1:k) = ix2(2:kp1)                                  (3.1)
end
```

This section looks at a better version of the ACORN algorithms that uses a11, a12, a13, a21, a22, a23, and arrays ix1 and ix2 that appear in equation 3.1. The calculations are all based on the value of k = 17 and M = 2^64 - 253. The variable r is the sought uniform random value in the range (0, 1].

The approache that estimates the best coefficients for a ACORN 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 array ix1 and ix2. 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 = 2^10;
xM2 = 2^11;
```

```
maxElems=10000;
maxIters=100;
lb=[xM1 xM1 1 1 1 1 1 1];
ub=[xM2 xM2 3 3 3 3 3 3];
sFilename='res1.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
bRound=true;
nDigits = 10;
% try
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,bRo
und,nDigits)
% catch ME
%     jjj=1;
% end

maxElems=10000;
maxIters=100;
lb=[xM1 xM1 1 1 1 1 1 1];
ub=[xM2 xM2 50 50 50 50 50 50];
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=[xM1 xM1 1 1 1 1 1 1];
ub=[xM2 xM2 100 100 100 100 100 100];
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=[xM1 xM1 1 1 1 1 1 1];
ub=[xM2 xM2 150 150 150 150 150 150];
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=[xM1 xM1 1 1 1 1 1 1];
ub=[xM2 xM2 200 200 200 200 200 200];
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)

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

maxElems=10000;
maxIters=100;
x =
[1291.133976,1063.643659,1.207326864,1.653642987,2.333190616,2.586482435,2.94
0401912,1.81695708];
r = 0.15;
lb = (1-r)*x;
ub = (1+r)*x;
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 =
[1041.32554,1663.722768,47.03196877,10.67649157,7.481713073,1.039904008,29.41
544209,36.26414739];
r = 0.15;
lb = (1-r)*x;
ub = (1+r)*x;
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 =
[1328.158662,1853.972603,36.09374619,69.99021043,63.07434469,46.18987718,16.9
1432496,57.84417329];
r = 0.15;
lb = (1-r)*x;
ub = (1+r)*x;
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 =
[1950.252432,1245.132756,119.0305414,92.69647791,104.1550266,84.99439168,150,
146.3767716];
r = 0.15;
lb = (1-r)*x;
ub = (1+r)*x;
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 =
[1281.341019,1385.692708,188.117186,182.8184018,154.313403,1,1,113.7504115];
r = 0.15;
lb = (1-r)*x;
ub = (1+r)*x;
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 3.2 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,bIs
Round,nDigits)
global gmaxElems
global bestFactor
global bRound
global numDigits

gmaxElems = maxElems;
bRound = bIsRound;
numDigits = nDigits;

bResetResMat = true;
% output file exists?
if isfile(sFilename)
  resMat = readmatrix(sFilename,"Sheet", sSheetName,"Range","A2:I101");
  [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,9);
  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 , numDigits);
  resMat(i,1) = bestFactor;
  resMat(i,2:9) = x;

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

  resMat = sortrows(resMat,1);

  T1 = array2table(resMat);
  T1.Properties.VariableNames(1:9) = {'Factor', 'iX1','iX2', 'A11' ,'A12',
'A13', 'A21' ,'A22', 'A23'};
  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(1:9) = {'Factor', 'iX1','iX2', 'A11' ,'A12',
'A13', 'A21' ,'A22', 'A23'};
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) = {'iX1Low', lb(1)};
c(4,1:2) = {'iX1Hi', ub(1)};
c(5,1:2) = {'iX2Low', lb(2)};
c(6,1:2) = {'iX2Hi', ub(2)};
c(7,1:2) = {'a11Low', lb(3)};
c(8,1:2) = {'a11Hi', ub(3)};
c(9,1:2) = {'a12Low', lb(4)};
c(10,1:2) = {'a12Hi', ub(4)};
c(11,1:2) = {'a13Low', lb(5)};
c(12,1:2) = {'a13Hii', ub(5)};
c(13,1:2) = {'a21Low', lb(6)};
c(14,1:2) = {'a21Hi', ub(6)};
c(15,1:2) = {'a22Low', lb(7)};
c(16,1:2) = {'a22Hi', ub(7)};
c(17,1:2) = {'a23Low', lb(8)};
c(18,1:2) = {'a23Hi', ub(8)};
c(19,1:2) = {'PopSize', POSpopsize};
c(20,1:2) = {'PopMaxIters', POSmaxIters};
```

```
if bRound
  c(21,1:2) = {'Rounded?', "True"};
  c(22,1:2) = {'Rounded', numDigits};
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 for Reg997Gen1.m.

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

  M = 2^64 - 253;
  k = 17;
  kp1 = k + 1;
  maxElems = gmaxElems;
  numDigits = 10;
  x=zeros(maxElems,1);
  ix1 = zeros(kp1,1);
  ix2 = zeros(kp1,1);
  rng('shuffle','twister');
  for i=1:kp1
    ix1(i) = 1 + fix(rand*c(1));
    ix2(i) = 1 + fix(rand*c(2));
  end
  a11 = c(3);
  a12 = c(4);
  a13 = c(5);
  a21 = c(6);
  a22 = c(7);
  a23 = c(8);
  for iter=1:maxElems
    for i=2:k
      ip1 = i + 1;
      im1 = i - 1;
      ix1(ip1) = a11*ix1(ip1) + a12*ix1(i) + a13*ix1(im1);
      ix2(ip1) = a21*ix2(ip1) + a22*ix2(i) + a23*ix2(im1);
      if ix2(ip1) > M
        ix2(ip1) = mod(ix2(ip1), M);
        ix1(ip1) = ix1(ip1) + 1;
      end
      if ix1(ip1) > M, ix1(ip1) = mod(ix1(ip1),M); end
    end
    x(iter) = mod(ix1(kp1) + ix2(kp1), M) / M;
    ix1(1:k) = ix1(2:kp1);
```

```
    ix2(1:k) = ix2(2:kp1);
  end
  if bRound, 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 of the wide-trust region optimization.

	res1	res2	res3	res4	res5
	Wide Range	Wide Range	Wide Range	Wide Range	Wide Range
Mean	112.5327951	112.0593511	111.957722	112.1574813	112.0479908
Sdev	2.474556302	2.496994531	2.403546176	2.276820972	2.754376626
Min	104.9533536	101.9562724	104.0005099	104.9899564	103.6960196
Max	117.8066623	116.7753578	117.1048602	116.1280264	117.1903193
Range	12.85330871	14.8190854	13.10435035	11.13806996	13.49429966
Count	100	100	100	100	100
Conf	0.554647725	0.559677035	0.538731495	0.510327274	0.617366728
CI Upper	113.0874428	112.6190281	112.4964535	112.6678086	112.6653575
CI Lower	111.9781473	111.4996741	111.4189905	111.647154	111.430624
Factor	104.9533536	101.9562724	104.0005099	104.9899564	103.6960196
iX1	1291.133976	1041.32554	1328.158662	1950.252432	1281.341019
iX2	1063.643659	1663.722768	1853.972603	1245.132756	1385.692708
A11	1.207326864	47.03196877	36.09374619	119.0305414	188.117186
A12	1.653642987	10.67649157	69.99021043	92.69647791	182.8184018
A13	2.333190616	7.481713073	63.07434469	104.1550266	154.313403
A21	2.586482435	1.039904008	46.18987718	84.99439168	1
A22	2.940401912	29.41544209	16.91432496	150	1
A23	1.81695708	36.26414739	57.84417329	146.3767716	113.7504115
Max Elements	10000	10000	10000	10000	10000
Max Iters	100	100	100	100	100
iX1Low	1024	1024	1024	1024	1024
iX1Hi	2048	2048	2048	2048	2048
iX2Low	1024	1024	1024	1024	1024
iX2Hi	2048	2048	2048	2048	2048
a11Low	1	1	1	1	1
a11Hi	3	50	100	150	200
a12Low	1	1	1	1	1
a12Hi	3	50	100	150	200
a13Low	1	1	1	1	1
a13Hii	3	50	100	150	200
a21Low	1	1	1	1	1
a21Hi	3	50	100	150	200
a22Low	1	1	1	1	1
a22Hi	3	50	100	150	200
a23Low	1	1	1	1	1
a23Hi	3	50	100	150	200
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.

1. Table 3.2 shows the results of the narrow-trust region optimization.

	res1b	res2b	res3b	res4b	res5b
	Narrow Range	Narrow Range	Narrow Range	Narrow Range	Narrow Range

Mean	112.5771242	112.2314062	112.5573937	112.0444324	112.0255625
Sdev	1.895672625	2.458018384	2.253845258	2.45380593	2.727024713
Min	107.4873943	103.1093083	107.2971326	104.7718344	103.1997973
Max	116.5539166	116.9055181	117.7237544	116.8939084	117.2123006
Range	9.066522366	13.79620987	10.42662171	12.12207404	14.01250325
Count	100	100	100	100	100
Conf	0.424896579	0.550940911	0.505177491	0.549996731	0.611236063
CI Upper	113.0020208	112.7823471	113.0625712	112.5944291	112.6367986
CI Lower	112.1522276	111.6804653	112.0522162	111.4944356	111.4143264
Factor	107.4873943	103.1093083	107.2971326	104.7718344	103.1997973
iX1	1484.804072	1143.540153	1345.137766	1768.652854	1373.416401
iX2	1050.050141	1834.116287	1809.690441	1200.359875	1363.818439
A11	1.151189883	46.5485523	41.03398297	136.8799258	184.1918715
A12	1.851241288	10.21334458	69.64298734	81.23973335	210.2411621
A13	1.983212024	8.603970034	67.62000762	93.43921363	166.512196
A21	2.9744548	1.195889609	50.15030409	91.17178892	1.091077403
A22	2.499341625	31.96161157	17.28213738	172.131416	0.984877347
A23	2.089500642	37.80192622	66.13189991	124.4892246	120.9571991
Max Elements	10000	10000	10000	10000	10000
Max Iters	100	100	100	100	100
iX1Low	1097.4639	885.1267	1128.9349	1657.7146	1089.1399
iX1Hi	1484.8041	1197.5244	1527.3825	2242.7903	1473.5422
iX2Low	904.0971	1414.1644	1575.8767	1058.3628	1177.8388
iX2Hi	1223.1902	1913.2812	2132.0685	1431.9027	1593.5466
a11Low	1.0262	39.9772	30.6797	101.176	159.8996
a11Hi	1.3884	54.0868	41.5078	136.8851	216.3348
a12Low	1.4056	9.075	59.4917	78.792	155.3956
a12Hi	1.9017	12.278	80.4887	106.6009	210.2412
a13Low	1.9832	6.3595	53.6132	88.5318	131.1664
a13Hii	2.6832	8.604	72.5355	119.7783	177.4604
a21Low	2.1985	0.88392	39.2614	72.2452	0.85
a21Hi	2.9745	1.1959	53.1184	97.7436	1.15
a22Low	2.4993	25.0031	14.3772	127.5	0.85
a22Hi	3.3815	33.8278	19.4515	172.5	1.15
a23Low	1.5444	30.8245	49.1675	124.4203	96.6878
a23Hi	2.0895	41.7038	66.5208	168.3333	130.813
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-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=[1484.804072,1050.050141,1.151189883,1.851241288,1.983212024,2.9744548,2.49
9341625,2.089500642];
```



```
sFilename='res1c.xlsb';
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,bRound,nDigits)


maxElems=10000;
maxIters=1000000;
c=[1143.540153,1834.116287,46.5485523,10.21334458,8.603970034,1.195889609,31.
96161157,37.80192622];
sFilename='res2c.xlsb';
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c=[1345.137766,1809.690441,41.03398297,69.64298734,67.62000762,50.15030409,17
.28213738,66.13189991];
sFilename='res3c.xlsb';
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c=[1768.652854,1200.359875,136.8799258,81.23973335,93.43921363,91.17178892,17
2.131416,124.4892246];
sFilename='res4c.xlsb';
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c=[1373.416401,1363.818439,184.1918715,210.2411621,166.512196,1.091077403,0.9
84877347,120.9571991];
sFilename='res5c.xlsb';
bRound=true;
nDigits = 10;
doAll2(maxElems,maxIters,c,sFilename,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,bIsRound,nDigits)
global gmaxElems
global bRound
global numDigits

gmaxElems = maxElems;
bRound = bIsRound;
numDigits = nDigits;
```

```
bResetResMat = true;
% output file exists?
% if isfile(sFilename)
%   resMat = readmatrix(sFilename,"Sheet", sSheetName,"Range","A2:I101");
%   [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,9);
  nStart = 1;
end

for i=nStart:maxIters
  factor = rng997Gen2(c);
  resMat(i,1) = factor;
  resMat(i,2:9) = c;

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

%   resMat = sortrows(resMat,1);
%
%   T1 = array2table(resMat);
%   T1.Properties.VariableNames(1:9) = {'Factor', 'iX1','iX2', 'A11' ,'A12',
'A13', 'A21' ,'A22', 'A23'};
%   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(1:9) = {'Factor', 'iX1','iX2', 'A11' ,'A12',
'A13', 'A21' ,'A22', 'A23'};
writetable(T1, sFilename, "Sheet", "Sheet1");
end
```

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

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

```
function factor = rng997Gen2(c)
```

```
%UNTITLED2 Summary of this function goes here
  global gmaxElems
  global numDigits;
  global bRound


  M = 2^64 - 253;
  k = 17;
  kp1 = k + 1;
  maxElems = gmaxElems;
  numDigits = 10;
  x=zeros(maxElems,1);
  ix1 = zeros(kp1,1);
  ix2 = zeros(kp1,1);
  rng('shuffle','twister');
  for i=1:kp1
    ix1(i) = 1 + fix(rand*c(1));
    ix2(i) = 1 + fix(rand*c(2));
  end
  a11 = c(3);
  a12 = c(4);
  a13 = c(5);
  a21 = c(6);
  a22 = c(7);
  a23 = c(8);
  for iter=1:maxElems
    for i=2:k
      ip1 = i + 1;
      im1 = i - 1;
      ix1(ip1) = a11*ix1(ip1) + a12*ix1(i) + a13*ix1(im1);
      ix2(ip1) = a21*ix2(ip1) + a22*ix2(i) + a23*ix2(im1);
      if ix2(ip1) > M
        ix2(ip1) = mod(ix2(ip1), M);
        ix1(ip1) = ix1(ip1) + 1;
      end
      if ix1(ip1) > M, ix1(ip1) = mod(ix1(ip1),M); end
    end
    x(iter) = mod(ix1(kp1) + ix2(kp1), M) / M;
    ix1(1:k) = ix1(2:kp1);
    ix2(1:k) = ix2(2:kp1);
  end
  if bRound, 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	147.8682416	147.7602238	147.7724376	147.7446966	147.7162231
Sdev	11.79155536	11.77244922	11.78283433	11.7588075	11.76810684
Min	103.7447095	106.0859385	104.4878863	104.3500725	105.6830748
Max	232.0632299	225.528088	227.6615201	236.0026115	224.3526256
Range	128.3185204	119.4421496	123.1736338	131.652539	118.6695508
Count	1000000	1000000	1000000	1000000	1000000
Conf	0.026429624	0.0263868	0.026410077	0.026356223	0.026377067
CI Upper	147.8946713	147.7866106	147.7988477	147.7710528	147.7426001
CI Lower	147.841812	147.733837	147.7460275	147.7183404	147.689846
Max Elements	103.7447095	106.0859385	104.4878863	104.3500725	105.6830748
Max Iters	1484.804072	1143.540153	1345.137766	1768.652854	1373.416401

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

The column titled res5c in Table 3.3 has the lowest upper mean value. The best modified ACORN equation is:

```
k = 17
M = 2^64 - 253
for i=1:k+1
  ix1(i) = 1 + fix(rand*1373.416401)
  ix2(i) = 1 + fix(rand*1363.818439)
end

for iter=1 to random numbers count
  for i=2 to k
    ip1 = i + 1
    im1 = i -0 1
    ix1(ip1) = 184.1918715*ix1(ip1) + 210.2411621*ix1(i) + 166.512196*ix(im1)
    ix2(ip1) = 1.091077403*ix2(ip1) + 0.984877347*ix2(i)
                 + 120.9571991 * ix(im1)
    if ix2(ip1) > M
      ix2(ip1) = ix2(ip1) mod M
      ix1(ip1) = ix1(ip1) + 1
    end
    if ix1(ip1) > M, ix1(ip1) = ix1(ip1) mod M)
  end
  x(iter) = (ix1(kp1) + ix2(kp1) mod M) / M
  ix1(1:k) = ix1(2:kp1)
  ix2(1:k) = ix2(2:kp1)
end                                                              (3.2)
```

The first loop initializes the arrays ix1 and ix2. 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.

