Project 997 PRNGs Part 2WH
Wichmann-Hill Variant Algorithms
By
Namir C. Shammas

1/ Introduction

The Wichmann-Hill PRNG algorithm is a good method to generate uniform random numbers. The algorithm uses multiple equations to first calculate random integers. Then, the last step uses these random integers to calculate a uniform random number. This study explores variants of the Wichmann-Hill PRNG algorithm.

2/ Wichmann-Hill Algorithms (Version WH 2 PSO)

The Wichmann-Hill (WH) algorithm is defined (for 16-bit integers) as:

[r, r1, r2, r3] = function WH(*r1*, *r2*, *r3*)
  % r1, r2, r3 should be random from 1 to 30,000.
  $r1 = 171 \times \text{mod}(r1, 177) - 2 \times \text{floor}(r1 / 177)$
  $r2 = 172 \times \text{mod}(r2, 176) - 35 \times \text{floor}(r2 / 176)$
  $r3 = 170 \times \text{mod}(r3, 178) - 63 \times \text{floor}(r3 / 178)$
  $r = \text{mod}(r1/30269 + r2/30307 + r3/30323, 1)$ (2.1)
end

This study looks at another variantof the WH algorithms that uses optimum values for the integer constants r1, r2, r3, and r4.

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

1. Optimize the penalty factor (see Appendix of Project 997 PRNGs Part 1) using particle swarm optimization algorithms. This optimization starts with a wide trust region and narrows it down.

2. Optimize the penalty using the narrowed optimum regions obtained in step 1. This step yields refined trust regions.
3. Perform a large run of generating random numbers using the refined trust regions from step 2. The calculations yield the statistics for the penalty factor. The upper confidence values for the mean penlty factor are the values we look at to determine the fitness of the algorithm.

The two optimization phases involve using the following set of equations:

iw=round(rand*iw,0)
ix=round(rand*ix,0)
iy=round(rand*iy,0)
iz=round(rand*iz,0) (2.2)

iw = c1+a1*iw mod Mv1
ix = c2+a2*ix mod Mv2
iy = c3+a3*iy mod Mv3
iz = c4+a4*iz mod Mv4
r = iy/Mv1 + ix/Mv2 + iy/Mv3 + iz/Mv4 mod 1 (2.3)

The process finds the optimum values for iw, ix, iy, iz, c1, c2, c3, c4, a1, a2, a3, and a4. The value for r is the uniform random number in (0,1]. The values of Mv1 through Mv4 are selected from the the following values. Each set of calculations in calculations use a different set of four constanst out of the following values.

```
M1 = 16718909
M2 = 16725061
M3 = 16731047
M4 = 16737079
M5 = 16743091
M6 = 16749377
M7 = 16755383
M8 = 16761491
M9 = 16767827
```

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

```
M1 = 16718909;
M2 = 16725061;
M3 = 16731047;
M4 = 16737079;
M5 = 16743091;
M6 = 16749377;
M7 = 16755383;
```



```
M8 = 16761491;
M9 = 16767827;

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

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

maxElems=10000;
maxIters=250;
lb=[100 11 111 100 11 111 100 11 111 100 11 111];
ub=[100000 1000000 1000000000 100000 1000000 1000000000 100000 1000000
1000000000 100000 1000000 1000000000];
sFilename='res1.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=[M1,M2,M3,M4];
bRound=true;
nDigits = 10;
doAll(maxElems,
maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,bRound,nDigits)

maxElems=10000;
maxIters=250;
lb=[100 11 111 100 11 111 100 11 111 100 11 111];
ub=[100000 1000000 1000000000 100000 1000000 1000000000 100000 1000000
1000000000 100000 1000000 1000000000];
sFilename='res2.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=[M2,M3,M4,M5];
bRound=true;
nDigits = 10;
% doAll(maxElems,
maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,bRound,nDigits)

maxElems=10000;
maxIters=250;
lb=[100 11 111 100 11 111 100 11 111 100 11 111];
ub=[100000 1000000 1000000000 100000 1000000 1000000000 100000 1000000
1000000000 100000 1000000 1000000000];
sFilename='res3.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=[M3,M4,M5,M6];
bRound=true;
nDigits = 10;
% doAll(maxElems,
maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,bRound,nDigits)

maxElems=10000;
maxIters=250;
lb=[100 11 111 100 11 111 100 11 111 100 11 111];
```



```
ub=[100000 1000000 1000000000 100000 1000000 1000000000 100000 1000000
1000000000 100000 1000000 1000000000];
sFilename='res4.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=[M4,M5,M6,M7];
bRound=true;
nDigits = 10;
% doAll(maxElems,
maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,bRound,nDigits)

maxElems=10000;
maxIters=250;
lb=[100 11 111 100 11 111 100 11 111 100 11 111];
ub=[100000 1000000 1000000000 100000 1000000 1000000000 100000 1000000
1000000000 100000 1000000 1000000000];
sFilename='res5.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=[M5,M6,M7,M8];
bRound=true;
nDigits = 10;
% doAll(maxElems,
maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,bRound,nDigits)

maxElems=10000;
maxIters=250;
lb=[100 11 111 100 11 111 100 11 111 100 11 111];
ub=[100000 1000000 1000000000 100000 1000000 1000000000 100000 1000000
1000000000 100000 1000000 1000000000];
sFilename='res6.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=[M6,M7,M8,M9];
bRound=true;
nDigits = 10;
% doAll(maxElems,
maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,bRound,nDigits)

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

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

maxElems=10000;
maxIters=250;
x =
[100000,270644,575654316,1542,580787,97510638,39473,11,1000000000,40825,31911
0,511474550];
r = 0.15;
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
sFilename='res1b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
```



```
POSmaxIters=350;
xM=[M1,M2,M3,M4];
bRound=true;
nDigits = 10;

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

maxElems=10000;
maxIters=250;
x =
[16223,331985,231047694,94746,376928,26288003,81901,9548,340612855,100000,160
594,789074549];
r = 0.15;
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
sFilename='res2b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=[M2,M3,M4,M5];
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=250;
x =
[86793,355382,919751143,19164,481725,35478589,8468,327605,327033758,10462,591
7,798151647];
r = 0.15;
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
sFilename='res3b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=[M3,M4,M5,M6];
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=250;
x =
[100000,289456,37036485,12193,965295,1000000000,21473,502772,774671012,100000
,1000000,353610382];
r = 0.15;
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
sFilename='res4b.xlsb';
sSheetName='Sheet1';
```



```
POSpopsize=250;
POSmaxIters=350;
xM=[M4,M5,M6,M7];
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=250;
x =
[87825,205890,259595645,24451,144602,173092469,34642,823924,481385876,49839,7
31875,974681316];
r = 0.15;
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
sFilename='res5b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=[M5,M6,M7,M8];
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=250;
x =
[57514,974897,308866306,44183,815411,634787532,48479,666497,823428474,70743,6
31830,813572607];
r = 0.15;
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
sFilename='res6b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=[M6,M7,M8,M9];
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, xMarr, bRound, nDigits)
global gmaxElems
global bestFactor
global M1
global M2
global M3
global M4
global doRounding
global numDigits

gmaxElems = maxElems;
M1 = xMarr(1);
M2 = xMarr(2);
M3 = xMarr(3);
M4 = xMarr(4);
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,13);
n = fix(log10(maxIters));
m = fix(maxIters/20);
for i=1:maxIters
  if mod(i,m)==0, fprintf("."); end
  bestFactor = 1e+99;
%   xrnd = round(i/(maxIters+1),n);
  x = particleswarm(@rng997Gen1,length(lb),lb,ub,options);
  resMat(i,1) = bestFactor;
  resMat(i,2:13) = round(x,0);
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(1:13) = {'Factor', 'IW', 'C1','A1', 'IX', 'C2'
,'A2', 'IY', 'C3','A3', 'IZ', 'C4' ,'A4'};
writetable(T1, sFilename, "Sheet", sSheetName);
pause(10);
c = cell(34,2);
c(1,1:2) = {'Max Elements', maxElems};
c(2,1:2) = {'Max Iters', maxIters};
c(3,1:2) = {'IWlow', lb(1)};
c(4,1:2) = {'IWhi', ub(1)};
c(5,1:2) = {'C1low', lb(2)};
```



```
c(6,1:2) = {'C1hi', ub(2)};
c(7,1:2) = {'A1low', lb(3)};
c(8,1:2) = {'A1hi', ub(3)};
c(9,1:2) = {'IXlow', lb(4)};
c(10,1:2) = {'IXhi', ub(4)};
c(11,1:2) = {'C2low', lb(5)};
c(12,1:2) = {'C2hi', ub(5)};
c(13,1:2) = {'A2low', lb(6)};
c(14,1:2) = {'A2hi', ub(6)};
c(15,1:2) = {'IYlow', lb(7)};
c(16,1:2) = {'IYhi', ub(7)};
c(17,1:2) = {'C3low', lb(8)};
c(18,1:2) = {'C3hi', ub(8)};
c(19,1:2) = {'A3low', lb(9)};
c(20,1:2) = {'A3hi', ub(9)};
c(21,1:2) = {'IZlow', lb(10)};
c(22,1:2) = {'IZhi', ub(10)};
c(23,1:2) = {'C4low', lb(11)};
c(24,1:2) = {'C4hi', ub(11)};
c(25,1:2) = {'A4low', lb(12)};
c(26,1:2) = {'A4hi', ub(12)};
c(27,1:2) = {'PopSize', POSpopsize};
c(28,1:2) = {'PopMaxIters', POSmaxIters};
c(29,1:2) = {'M1', M1};
c(30,1:2) = {'M2', M2};
c(31,1:2) = {'M3', M3};
c(32,1:2) = {'M4', M4};
if doRounding
  c(33,1:2) = {'Rounded?', "True"};
  c(34,1:2) = {'Rounded', nDigits};
else
  c(33,1:2) = {'Rounded?', "False"};
  c(34,1:2) = {'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 partial listing for Reg997Gen1.m.

```
function factor = rng997Gen1(c)
%UNTITLED2 Summary of this function goes here
  global gmaxElems
  global bestFactor
  global bestX
  global M1
  global M2
  global M3
  global M4
  global doRounding
  global numDigits

  maxElems = gmaxElems;
  c = round(c, 0);
```



```
  iw = c(1);
  c1 = c(2);
  a1 = c(3);
  ix = c(4);
  c2 = c(5);
  a2 = c(6);
  iy = c(7);
  c3 = c(8);
  a3 = c(9);
  iz = c(10);
  c4 = c(11);
  a4 = c(12);

  x=zeros(maxElems,1);
  for i=1:maxElems
    iw = mod(c1+a1*iw,M1);
    ix = mod(c2+a2*ix,M2);
    iy = mod(c3+a3*iy,M3);
    iz = mod(c4+a4*iz,M4);
    x(i) = mod(iy/M1 + ix/M2 + iy/M3 + iz/M4,1);
  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 partial listing of file rng997Gen1.m.

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

	res1	res2	res3
	Wide Range	Wide Range	Wide Range
Mean	111.871996	112.155761	112.270018
Sdev	2.61335628	2.36717582	2.1521915
Min	99.9684136	104.219907	105.622884
Max	118.165557	116.530985	117.448297
Range	18.1971439	12.3110779	11.8254132
Count	250	250	250
Conf	0.37046613	0.3355679	0.30509199
CI Upper	112.242462	112.491329	112.57511
CI Lower	111.50153	111.820193	111.964926
IW	100000	16223	86793
C1	270644	331985	355382
A1	575654316	231047694	919751143
IX	1542	94746	19164
C2	580787	376928	481725
A2	97510638	26288003	35478589
IY	39473	81901	8468
C3	11	9548	327605
A3	1000000000	340612855	327033758
IZ	40825	100000	10462
C4	319110	160594	5917
A4	511474550	789074549	798151647
Max Elements	10000	10000	10000
Max Iters	250	250	250
IWlow	100	100	100
IWhi	100000	100000	100000
C1low	11	11	11
C1hi	1000000	1000000	1000000

A1low	111	111	111
A1hi	1000000000	1000000000	1000000000
IXlow	100	100	100
IXhi	100000	100000	100000
C2low	11	11	11
C2hi	1000000	1000000	1000000
A2low	111	111	111
A2hi	1000000000	1000000000	1000000000
IYlow	100	100	100
IYhi	100000	100000	100000
C3low	11	11	11
C3hi	1000000	1000000	1000000
A3low	111	111	111
A3hi	1000000000	1000000000	1000000000
IZlow	100	100	100
IZhi	100000	100000	100000
C4low	11	11	11
C4hi	1000000	1000000	1000000
A4low	111	111	111
A4hi	1000000000	1000000000	1000000000
PopSize	250	250	250
PopMaxIters	350	350	350
M1	16718909	16725061	16731047
M2	16725061	16731047	16737079
M3	16731047	16737079	16743091
M4	16737079	16743091	16749377
Rounded?	TRUE	TRUE	TRUE
Rounded	10	10	10

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

	res4	res5	res6
	Wide Range	Wide Range	Wide Range
Mean	112.352577	112.112868	112.194719
Sdev	2.46075778	2.38796016	2.4857467
Min	100.780283	105.054705	103.182739
Max	119.154019	116.425769	117.075435
Range	18.3737365	11.3710639	13.8926965
Count	250	250	250
Conf	0.34883396	0.33851426	0.35237636
CI Upper	112.701411	112.451383	112.547095
CI Lower	112.003743	111.774354	111.842343
IW	100000	87825	57514
C1	289456	205890	974897
A1	37036485	259595645	308866306
IX	12193	24451	44183
C2	965295	144602	815411
A2	1000000000	173092469	634787532
IY	21473	34642	48479
C3	502772	823924	666497
A3	774671012	481385876	823428474
IZ	100000	49839	70743
C4	1000000	731875	631830
A4	353610382	974681316	813572607

Max Elements	**10000**	**10000**	**10000**
Max Iters	**250**	**250**	**250**
IWlow	**100**	**100**	**100**
IWhi	**100000**	**100000**	**100000**
C1low	**11**	**11**	**11**
C1hi	**1000000**	**1000000**	**1000000**
A1low	**111**	**111**	**111**
A1hi	**1000000000**	**1000000000**	**1000000000**
IXlow	**100**	**100**	**100**
IXhi	**100000**	**100000**	**100000**
C2low	**11**	**11**	**11**
C2hi	**1000000**	**1000000**	**1000000**
A2low	**111**	**111**	**111**
A2hi	**1000000000**	**1000000000**	**1000000000**
IYlow	**100**	**100**	**100**
IYhi	**100000**	**100000**	**100000**
C3low	**11**	**11**	**11**
C3hi	**1000000**	**1000000**	**1000000**
A3low	**111**	**111**	**111**
A3hi	**1000000000**	**1000000000**	**1000000000**
IZlow	**100**	**100**	**100**
IZhi	**100000**	**100000**	**100000**
C4low	**11**	**11**	**11**
C4hi	**1000000**	**1000000**	**1000000**
A4low	**111**	**111**	**111**
A4hi	**1000000000**	**1000000000**	**1000000000**
PopSize	**250**	**250**	**250**
PopMaxIters	**350**	**350**	**350**
M1	**16737079**	**16743091**	**16749377**
M2	**16725061**	**16731047**	**16737079**
M3	**16731047**	**16737079**	**16743091**
M4	**16737079**	**16743091**	**16749377**
Rounded?	**TRUE**	**TRUE**	**TRUE**
Rounded	**10**	**10**	**10**

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

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

	res1b	**res2b**	**res3b**
	Narrow Range	**Narrow Range**	**Narrow Range**
Mean	**112.003939**	**112.208125**	**112.318607**
Sdev	**2.43823453**	**2.36643433**	**2.38987158**
Min	**100.736871**	**102.713236**	**104.884133**
Max	**117.123443**	**117.746222**	**117.02725**
Range	**16.3865715**	**15.0329861**	**12.1431167**
Count	**250**	**250**	**250**
Conf	**0.34564109**	**0.33546279**	**0.33878522**
CI Upper	**112.34958**	**112.543588**	**112.657392**
CI Lower	**111.658298**	**111.872663**	**111.979822**

IW	104578	18656	81317
C1	261612	317471	368223
A1	662002463	222394052	949722674
IX	1549	83451	16609
C2	530006	382723	528031
A2	102992215	28212821	34176041
IY	39825	73934	7764
C3	13	8625	375071
A3	1027009018	298192496	375383542
IZ	44815	102371	8893
C4	349084	136505	5754
A4	439793665	751660950	810080686
Max Elements	10000	10000	10000
Max Iters	250	250	250
IWlow	85000	13790	73774
IWhi	115000	18656	99812
C1low	230047	282187	302075
C1hi	311241	381783	408689
A1low	489306169	196390540	781788472
A1hi	662002463	265704848	1057713814
IXlow	1311	80534	16289
IXhi	1773	108958	22039
C2low	493669	320389	409466
C2hi	667905	433467	553984
A2low	82884042	22344803	30156801
A2hi	112137234	30231203	40800377
IYlow	33552	69616	7198
IYhi	45394	94186	9738
C3low	9	8116	278464
C3hi	13	10980	376746
A3low	850000000	289520927	277978694
A3hi	1150000000	391704783	376088822
IZlow	34701	85000	8893
IZhi	46949	115000	12031
C4low	271244	136505	5029
C4hi	366977	184683	6805
A4low	434753368	670713367	678428900
A4hi	588195733	907435731	917874394
PopSize	250	250	250
PopMaxIters	350	350	350
M1	16749377	16749377	16749377
M2	16755383	16755383	16755383
M3	16761491	16761491	16761491
M4	16767827	16767827	16767827
Rounded?	TRUE	TRUE	TRUE
Rounded	10	10	10

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

	res4b	res5b	res6b
	Narrow Range	Narrow Range	Narrow Range
Mean	112.0609276	112.125516	112.2231824

Sdev	2.564173392	2.374973528	2.414851429
Min	101.1340029	103.5326676	103.1227019
Max	117.1941398	116.9830325	118.2787056
Range	16.06013687	13.45036495	15.15600377
Count	250	250	250
Conf	0.363494029	0.336673292	0.342326333
CI Upper	112.4244216	112.4621893	112.5655087
CI Lower	111.6974335	111.7888427	111.880856
IW	96234	87322	59331
C1	263109	200307	940480
A1	38479426	298269534	269415958
IX	10611	25367	45354
C2	994371	164188	836101
A2	1051635700	150214840	572909402
IY	21258	34957	44372
C3	507091	755458	718675
A3	658470360	415650443	750954918
IZ	115000	52396	69619
C4	1056988	730263	598616
A4	360407176	1103691591	697060245
Max Elements	10000	10000	
Max Iters	250	250	
IWlow	85000	74651	
IWhi	115000	100999	
C1low	246038	175007	
C1hi	332874	236773	
A1low	31481012	220656298	
A1hi	42591958	298534992	
IXlow	10364	20783	
IXhi	14022	28119	
C2low	820501	122912	
C2hi	1110089	166292	
A2low	850000000	147128599	
A2hi	1150000000	199056339	
IYlow	18252	29446	
IYhi	24694	39838	
C3low	427356	700335	
C3hi	578188	947513	
A3low	658470360	409177995	
A3hi	890871664	553593757	
IZlow	85000	42363	
IZhi	115000	57315	
C4low	850000	622094	
C4hi	1150000	841656	
A4low	300568825	828479119	
A4hi	406651939	1120883513	
PopSize	250	250	
PopMaxIters	350	350	
M1	16749377	16749377	
M2	16755383	16755383	
M3	16761491	16761491	

M4	16767827	16767827	
Rounded?	TRUE	TRUE	
Rounded	10	10	

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

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

```
M1 = 16718909;
M2 = 16725061;
M3 = 16731047;
M4 = 16737079;
M5 = 16743091;
M6 = 16749377;
M7 = 16755383;
M8 = 16761491;
M9 = 16767827;

%
% Note: Uncomment one call to doAll2() at a time!
%

maxElems=10000;
maxIters=1000000;
c =
[104578,261612,662002463,1549,530006,102992215,39825,13,1027009018,44815,3490
84,439793665];
sFilename='res1c.xlsb';
sSheetName='Sheet1';
xM=[M1,M2,M3,M4];
bRound=true;
nDigits = 10;
doAll2(maxElems,maxIters,c,sFilename,sSheetName,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c =
[18656,317471,222394052,83451,382723,28212821,73934,8625,298192496,102371,136
505,751660950];
sFilename='res2c.xlsb';
sSheetName='Sheet1';
xM=[M2,M3,M4,M5];
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,sSheetName,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c =
[81317,368223,949722674,16609,528031,34176041,7764,375071,375383542,8893,5754
,810080686];
sFilename='res3c.xlsb';
sSheetName='Sheet1';
```



```
xM=[M3,M4,M5,M6];
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,sSheetName,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c =
[96234,263109,38479426,10611,994371,1051635700,21258,507091,658470360,115000,
1056988,360407176];
sFilename='res4c.xlsb';
sSheetName='Sheet1';
xM=[M4,M5,M6,M7];
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,sSheetName,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c =
[87322,200307,298269534,25367,164188,150214840,34957,755458,415650443,52396,7
30263,1103691591];
sFilename='res5c.xlsb';
sSheetName='Sheet1';
xM=[M5,M6,M7,M8];
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,c,sFilename,sSheetName,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c =
[59331,940480,269415958,45354,836101,572909402,44372,718675,750954918,69619,5
98616,697060245];
sFilename='res6c.xlsb';
sSheetName='Sheet1';
xM=[M6,M7,M8,M9];
bRound=true;
nDigits = 10;
% doAll2(maxElems,maxIters,lb,ub,sFilename,sSheetName,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, sSheetName, xMarr, bRound,
nDigits)
global gmaxElems
global M1
global M2
global M3
global M4
global doRounding
global numDigits
```



```
gmaxElems = maxElems;
M1 = xMarr(1);
M2 = xMarr(2);
M3 = xMarr(3);
M4 = xMarr(4);
doRounding = bRound;
numDigits = nDigits;
resMat=zeros(maxIters,17);
for i=1:maxIters
  [factor,WXYZrnd] = rng997Gen2(c);
  resMat(i,1) = factor;
  resMat(i,2:13) = round(c,0);
  resMat(i,14:17) = WXYZrnd;
  fprintf("iter: %i, Factor = %f\n", i, factor);
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(1:17) = {'Factor', 'IW', 'C1','A1', 'IX', 'C2'
,'A2', 'IY', 'C3','A3', 'IZ', 'C4' ,'A4', 'iWrnd','ixRnd','Yrnd','iZrnd'};
writetable(T1, sFilename, "Sheet", sSheetName);
fprintf("-------------------------------------------------------\n\n");
end
```

Listing 2.5. The source code file doAll2.m.

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

```
function [factor,WXYZrnd] = rng997Gen2(c)
%UNTITLED2 Summary of this function goes here
  global gmaxElems
  global bestFactor
  global bestX
  global M1
  global M2
  global M3
  global M4
  global doRounding
  global numDigits

  rng('shuffle','twister');
  maxElems = gmaxElems;
  c = round(c, 0);
  iw = c(1);
  c1 = c(2);
  a1 = c(3);
  ix = c(4);
  c2 = c(5);
```

```
  a2 = c(6);
  iy = c(7);
  c3 = c(8);
  a3 = c(9);
  iz = c(10);
  c4 = c(11);
  a4 = c(12);
  iw=round(rand*iw,0);
  ix=round(rand*ix,0);
  iy=round(rand*iy,0);
  iz=round(rand*iz,0);
  WXYZrnd = [iw,ix,iy,iz];
  x=zeros(maxElems,1);
  for i=1:maxElems
    iw = mod(c1+a1*iw,M1);
    ix = mod(c2+a2*ix,M2);
    iy = mod(c3+a3*iy,M3);
    iz = mod(c4+a4*iz,M4);
    x(i) = mod(iy/M1 + ix/M2 + iy/M3 + iz/M4,1);
  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 2.6. 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
	Random seed	Random seed	Random seed
Mean	147.7710767	147.7673404	147.7738236
Sdev	11.78978591	11.77083712	11.78002408
Min	105.4834697	102.8024557	106.3752766
Max	232.7578442	227.9964724	222.1119687
Range	127.2743745	125.1940167	115.7366921
Count	1000000	1000000	1000000
Conf	0.026425658	0.026383186	0.026403778
CI Upper	147.7975024	147.7937235	147.8002273
CI Lower	147.7446511	147.7409572	147.7474198
IW	104578	18656	81317
C1	261612	317471	368223
A1	662002463	222394052	949722674
IX	1549	83451	16609
C2	530006	382723	528031
A2	102992215	28212821	34176041
IY	39825	73934	7764
C3	13	8625	375071
A3	1027009018	298192496	375383542
IZ	44815	102371	8893
C4	349084	136505	5754
A4	439793665	751660950	810080686
Max Elements	10000	10000	10000
Max Iters	1000000	1000000	1000000

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

	res4c	res5c	res6c
	Random seed	Random seed	Random seed
Mean	147.787	147.764994	147.754773

Sdev	11.7854175	11.7910152	11.7872398
Min	105.73784	106.202883	104.600056
Max	226.115645	235.115588	224.971433
Range	120.377805	128.912705	120.371377
Count	1000000	1000000	1000000
Conf	0.02641587	0.02642841	0.02641995
CI Upper	147.813416	147.791423	147.781193
CI Lower	147.760584	147.738566	147.728353
IW	96234	87322	59331
C1	263109	200307	940480
A1	38479426	298269534	269415958
IX	10611	25367	45354
C2	994371	164188	836101
A2	1051635700	150214840	572909402
IY	21258	34957	44372
C3	507091	755458	718675
A3	658470360	415650443	750954918
IZ	115000	52396	69619
C4	1056988	730263	598616
A4	360407176	1103691591	697060245
Max Elements	10000	10000	10000
Max Iters	1000000	1000000	1000000

Table 2.3b. The rand-seed results for the penalty factor statistics (cont.).

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

iw=round(rand*59331,0)
ix=round(rand*45354,0)
iy=round(rand*44372,0)
iz=round(rand*69619,0) (2.5)

iw = 940480+269415958*iw mod 16743091
ix = 836101+572909402*ix mod16749377
iy = 718675+750954918*iy mod 16755383
iz = 598616+697060245*iz mod 16761491
r = iy/16743091 + ix/16749377 + iy/16755383 + iz/16761491 mod 1 (2.6)

The first set of equations initializes the parameters of iw, ix, iy, and iz. You can use these values in the second set of equations for as many iterations as you need. The value r is the uniform random number generated in the range of 0 to 1 (excluded).

3/ Wichmann-Hill Algorithms (Version WH I PSO)

The Wichmann-Hill (WH) algorithm is defined (for 16-bit integers) as:

[r, r1, r2, r3] = function WH(*r1*, *r2*, *r3*)
% r1, r2, r3 should be random from 1 to 30,000.
r1 = 171 × mod(*r1*, 177) − 2 × floor(r*1* / 177)
r2 = 172 × mod(*r2*, 176) − 35 × floor(*r2* / 176)
r3 = 170 × mod(*r3*, 178) − 63 × floor(*r3* / 178)
r = mod(r1/30269 + r2/30307 + r3/30323, 1) (3.1)
end

This study looks at a better version of the WH algorithms that uses optimum values for the integer constants r1, r2, r3, and r4.

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

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

The two optimization phases involve using the following set of equations:

iw = c1+a1*iw mod Mv1
ix = c2+a2*ix mod Mv2
iy = c3+a3*iy mod Mv3
iz = c4+a4*iz mod Mv4
r = iy/Mv1 + ix/Mv2 + iy/Mv3 + iz/Mv4 mod 1 (3.2)

The process finds the optimum values for iw, ix, iy, iz, c1, c2, c3, c4, a1, a2, a3, and a4. The value for r is the uniform random number in (0,1]. The values of Mv1 through Mv4 are selected from the the following values. Each set of calculations in calculations use a different set of four constanst out of the following values.


```
M1 = 16718909
M2 = 16733399
M3 = 16747883
M4 = 16762553
M5 = 16776971
```

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

```
M1 = 16718909;
M2 = 16733399;
M3 = 16747883;
M4 = 16762553;
M5 = 16776971;

maxElems=10000;
maxIters=250;
lb=[100 11 111 100 11 111 100 11 111 100 11 111];
ub=[100000 1000000 1000000000 100000 1000000 1000000000 100000 1000000
1000000000 100000 1000000 1000000000];
sFilename='res1.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=[M1,M2,M3,M4];
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=250;
lb=[100 11 111 100 11 111 100 11 111 100 11 111];
ub=[100000 1000000 1000000000 100000 1000000 1000000000 100000 1000000
1000000000 100000 1000000 1000000000];
sFilename='res2.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=[M2,M3,M4,M5];
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=250;
lb=[100 11 111 100 11 111 100 11 111 100 11 111];
ub=[100000 1000000 1000000000 100000 1000000 1000000000 100000 1000000
1000000000 100000 1000000 1000000000];
sFilename='res3.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
```



```
POSmaxIters=350;
xM=[M1,M2,M4,M5];
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=250;
lb=[100 11 111 100 11 111 100 11 111 100 11 111];
ub=[100000 1000000 1000000000 100000 1000000 1000000000 100000 1000000
1000000000 100000 1000000 1000000000];
sFilename='res4.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=[M1,M3,M4,M5];
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=250;
lb=[100 11 111 100 11 111 100 11 111 100 11 111];
ub=[100000 1000000 1000000000 100000 1000000 1000000000 100000 1000000
1000000000 100000 1000000 1000000000];
sFilename='res5.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=[M1,M2,M3,M5];
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

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

maxElems=10000;
maxIters=250;
x =
[100000,536998,327237597,72930,1000000,509588848,72547,616869,668765867,7533,
11,417407285];
r = 0.15;
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
sFilename='res1b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=[M1,M2,M3,M4];
bRound=true;
nDigits = 10;
```



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

maxElems=10000;
maxIters=250;
x = [];
r = 0.15;
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
sFilename='res2b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=[M2,M3,M4,M5];
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=250;
x = [14094,192045,787013850,308805,754969551];
r = 0.15;
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
sFilename='res3b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1=2^64-1;
xM2=2^48-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=250;
x = [36362,322301,376350456,126523,111];
r = 0.15;
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
sFilename='res4b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1=2^32-1;
xM2=2^16-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)
```



```
maxElems=10000;
maxIters=250;
x = [70245,96116,337947553,184184,41947176];
r = 0.15;
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
sFilename='res5b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1=2^48-1;
xM2=2^32-1;
bRound=true;
nDigits = 10;
%
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xM,
bRound,nDigits)

maxElems=10000;
maxIters=250;
x = [];
r = 0.15;
lb = round((1-r)*x,0);
ub = round((1+r)*x,0);
sFilename='res6b.xlsb';
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM1=2^48-1;
xM2=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.

```
function
doAll(maxElems,maxIters,lb,ub,sFilename,sSheetName,POSpopsize,POSmaxIters,xMa
rr,bRound,nDigits)
global gmaxElems
global bestFactor
global M1
global M2
global M3
global M4
global doRounding
```



```
global numDigits

gmaxElems = maxElems;
M1 = xMarr(1);
M2 = xMarr(2);
M3 = xMarr(3);
M4 = xMarr(4);
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,13);
n = fix(log10(maxIters));
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;
  resMat(i,2:13) = round(x,0);
  fprintf("Factor =  %f,", bestFactor);
  fprintf("%i, ", resMat(i,2:13));
  fprintf("\n");
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(1:13) = {'Factor', 'IW', 'C1','A1', 'IX', 'C2'
,'A2', 'IY', 'C3','A3', 'IZ', 'C4' ,'A4'};
writetable(T1, sFilename, "Sheet", sSheetName);
pause(10);
c = cell(34,2);
c(1,1:2) = {'Max Elements', maxElems};
c(2,1:2) = {'Max Iters', maxIters};
c(3,1:2) = {'IWlow', lb(1)};
c(4,1:2) = {'IWhi', ub(1)};
c(5,1:2) = {'C1low', lb(2)};
c(6,1:2) = {'C1hi', ub(2)};
c(7,1:2) = {'A1low', lb(3)};
c(8,1:2) = {'A1hi', ub(3)};
c(9,1:2) = {'IXlow', lb(4)};
c(10,1:2) = {'IXhi', ub(4)};
c(11,1:2) = {'C2low', lb(5)};
c(12,1:2) = {'C2hi', ub(5)};
c(13,1:2) = {'A2low', lb(6)};
c(14,1:2) = {'A2hi', ub(6)};
```



```
c(15,1:2) = {'IYlow', lb(7)};
c(16,1:2) = {'IYhi', ub(7)};
c(17,1:2) = {'C3low', lb(8)};
c(18,1:2) = {'C3hi', ub(8)};
c(19,1:2) = {'A3low', lb(9)};
c(20,1:2) = {'A3hi', ub(9)};
c(21,1:2) = {'IZlow', lb(10)};
c(22,1:2) = {'IZhi', ub(10)};
c(23,1:2) = {'C4low', lb(11)};
c(24,1:2) = {'C4hi', ub(11)};
c(25,1:2) = {'A4low', lb(12)};
c(26,1:2) = {'A4hi', ub(12)};
c(27,1:2) = {'PopSize', POSpopsize};
c(28,1:2) = {'PopMaxIters', POSmaxIters};
c(29,1:2) = {'M1', M1};
c(30,1:2) = {'M2', M2};
c(31,1:2) = {'M3', M3};
c(32,1:2) = {'M4', M4};
if doRounding
  c(33,1:2) = {'Rounded?', "True"};
  c(34,1:2) = {'Rounded', nDigits};
else
  c(33,1:2) = {'Rounded?', "False"};
  c(34,1:2) = {'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 partial listing for Reg997Gen1.m.

```
function factor = rng997Gen1(c)
%UNTITLED2 Summary of this function goes here
  global gmaxElems
  global bestFactor
  global bestX
  global M1
  global M2
  global M3
  global M4
  global doRounding
  global numDigits

  maxElems = gmaxElems;
  c = round(c, 0);
  iw = c(1);
  c1 = c(2);
  a1 = c(3);
  ix = c(4);
  c2 = c(5);
  a2 = c(6);
  iy = c(7);
  c3 = c(8);
  a3 = c(9);
```



```
  iz = c(10);
  c4 = c(11);
  a4 = c(12);

  x=zeros(maxElems,1);
  for i=1:maxElems
    iw = mod(c1+a1*iw,M1);
    ix = mod(c2+a2*ix,M2);
    iy = mod(c3+a3*iy,M3);
    iz = mod(c4+a4*iz,M4);
    x(i) = mod(iy/M1 + ix/M2 + iy/M3 + iz/M4,1);
  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 partial 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	112.3571889	112.4154774	112.3403936	112.1523218	112.2586375
Sdev	2.26701902	2.282853025	2.465471314	2.351045832	2.809078285
Min	104.7073844	104.5420733	102.9935462	104.1948702	100.4062348
Max	117.2449522	117.888978	117.1154675	118.0373727	119.1896423
Range	12.53756781	13.34690467	14.12192128	13.84250256	18.78340756
Count	250	250	250	250	250
Conf	0.321369795	0.323614404	0.349502145	0.333281331	0.398211442
CI Upper	112.6785587	112.7390918	112.6898958	112.4856031	112.6568489
CI Lower	112.0358191	112.0918629	111.9908915	111.8190404	111.860426
IW	100000	36249	66847	62981	57097
C1	536998	661254	648467	348622	454286
A1	327237597	349778272	192741784	213082551	105228798
IX	72930	24528	83072	88230	63859
C2	1000000	404866	149591	100962	148370
A2	509588848	681124142	453132065	211530223	216211511
IY	72547	91943	5689	100000	71649
C3	616869	388790	260727	227321	384301
A3	668765867	626574693	743982179	111	105206230
IZ	7533	100	56687	29318	36691
C4	11	190799	61207	478363	441375
A4	417407285	1000000000	946005613	470547794	760580564
Max Elements	10000	10000	10000	10000	10000
Max Iters	250	250	250	250	250
IWlow	100	100	100	100	100
IWhi	100000	100000	100000	100000	100000
C1low	11	11	11	11	11
C1hi	1000000	1000000	1000000	1000000	1000000
A1low	111	111	111	111	111
A1hi	1000000000	1000000000	1000000000	1000000000	1000000000
IXlow	100	100	100	100	100
IXhi	100000	100000	100000	100000	100000
C2low	11	11	11	11	11
C2hi	1000000	1000000	1000000	1000000	1000000
A2low	111	111	111	111	111
A2hi	1000000000	1000000000	1000000000	1000000000	1000000000
IYlow	100	100	100	100	100

IYhi	100000	100000	100000	100000	100000
C3low	11	11	11	11	11
C3hi	1000000	1000000	1000000	1000000	1000000
A3low	111	111	111	111	111
A3hi	1000000000	1000000000	1000000000	1000000000	1000000000
IZlow	100	100	100	100	100
IZhi	100000	100000	100000	100000	100000
C4low	11	11	11	11	11
C4hi	1000000	1000000	1000000	1000000	1000000
A4low	111	111	111	111	111
A4hi	1000000000	1000000000	1000000000	1000000000	1000000000
PopSize	250	250	250	250	250
PopMaxIters	350	350	350	350	350
M1	16718909	16733399	16718909	16718909	16718909
M2	16733399	16747883	16733399	16747883	16733399
M3	16747883	16762553	16762553	16762553	16747883
M4	16762553	16776971	16776971	16776971	16776971
Rounded?	TRUE	TRUE	TRUE	TRUE	TRUE
Rounded	10	10	10	10	10

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

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

	res1b	res2b	res3b	res4b	res5b
	Narrow Range	Narrow Range	Narrow Range	Narrow Range	Narrow Range
Mean	112.2045789	112.2084845	112.0685278	112.031605	112.1961281
Sdev	2.687327179	2.514375518	2.482298946	2.464064893	2.332473392
Min	104.9478262	102.8551402	102.4906688	103.6682985	102.355284
Max	117.9752242	117.506092	118.5402002	116.8612159	117.06025
Range	13.02739801	14.65095177	16.0495314	13.19291736	14.70496599
Count	250	250	250	250	250
Conf	0.380952156	0.356434744	0.351887609	0.349302773	0.330648525
CI Upper	112.585531	112.5649193	112.4204154	112.3809078	112.5267766
CI Lower	111.8236267	111.8520498	111.7166402	111.6823022	111.8654796
IW	107979	41686	66039	68240	61091
C1	577299	623914	585386	330464	506256
A1	329338405	335061272	168723170	216472596	112246494
IX	65024	25560	83136	95732	73438
C2	886516	430065	151875	102590	129036
A2	442838110	700802204	405529740	199379830	234015116
IY	64484	94799	4841	91269	76220
C3	574755	388966	257214	216756	326656
A3	656589874	720560897	855579506	94	105271768
IZ	7188	85	57484	26235	39481
C4	10	162179	56712	538031	440953
A4	443956272	850000000	955379239	499081728	661103184
Max Elements	10000	10000	10000	10000	10000
Max Iters	250	250	250	250	250
IWlow	85000	30812	56820	53534	48532

IWhi	115000	41686	76874	72428	65662
C1low	456448	562066	551197	296329	386143
C1hi	617548	760442	745737	400915	522429
A1low	278151957	297311531	163830516	181120168	89444478
A1hi	376323237	402245013	221653052	245044934	121013118
IXlow	61991	20849	70611	74996	54280
IXhi	83870	28207	95533	101464	73438
C2low	850000	344136	127152	85818	126115
C2hi	1150000	465596	172030	116106	170626
A2low	433150521	578955521	385162255	179800690	183779784
A2hi	586027175	783292763	521101875	243259756	248643238
IYlow	61665	78152	4836	85000	60902
IYhi	83429	105734	6542	115000	82396
C3low	524339	330472	221618	193223	326656
C3hi	709399	447108	299836	261419	441946
A3low	568450987	532588489	632384852	94	89425296
A3hi	769080747	720560897	855579506	128	120987164
IZlow	6403	85	48184	24920	31187
IZhi	8663	115	65190	33716	42195
C4low	9	162179	52026	406609	375169
C4hi	13	219419	70388	550117	507581
A4low	354796192	850000000	804104771	399965625	646493479
A4hi	480018378	1150000000	1087906455	541129963	874667649
PopSize	250	250	250	250	250
PopMaxIters	350	350	350	350	350
M1	16718909	16733399	16733399	16733399	16733399
M2	16733399	16747883	16747883	16747883	16747883
M3	16747883	16762553	16762553	16762553	16762553
M4	16762553	16776971	16776971	16776971	16776971
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. This file performs the random-seed generation of one million sets of 10,.000 random numbers for each tested version of the algorithm.

```
M1 = 16718909;
M2 = 16733399;
M3 = 16747883;
M4 = 16762553;
M5 = 16776971;

maxElems=10000;
maxIters=1000000;
c=
[107979,577299,329338405,65024,886516,442838110,64484,574755,656589874,7188,1
0,443956272];
c(10) = [];
c(7) = [];
c(4) = [];
c(1) = [];
sFilename='res1c.xlsb';
```



```
sSheetName='Sheet1';
POSpopsize=250;
POSmaxIters=350;
xM=[M1,M2,M3,M4];
bRound=true;
nDigits = 10;
doAll2(maxElems,maxIters,c,sFilename,sSheetName,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c=
[41686,623914,335061272,25560,430065,700802204,94799,388966,720560897,85,1621
79,850000000];
c(10) = [];
c(7) = [];
c(4) = [];
c(1) = [];
sFilename='res2c.xlsb';
sSheetName='Sheet1';
xM=[M2,M3,M4,M5];
bRound=true;
nDigits = 10;
doAll2(maxElems,maxIters,c,sFilename,sSheetName,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c=[66039,585386,168723170,83136,151875,405529740,4841,257214,855579506,57484,
56712,955379239];
c(10) = [];
c(7) = [];
c(4) = [];
c(1) = [];
sFilename='res3c.xlsb';
sSheetName='Sheet1';
xM=[M1,M2,M4,M5];
bRound=true;
nDigits = 10;
doAll2(maxElems,maxIters,c,sFilename,sSheetName,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
c=
[68240,330464,216472596,95732,102590,199379830,91269,216756,94,26235,538031,4
99081728];
c(10) = [];
c(7) = [];
c(4) = [];
c(1) = [];
sFilename='res4c.xlsb';
sSheetName='Sheet1';
xM=[M1,M3,M4,M5];
bRound=true;
nDigits = 10;
doAll2(maxElems,maxIters,c,sFilename,sSheetName,xM,bRound,nDigits)

maxElems=10000;
maxIters=1000000;
```



```
c=
[61091,506256,112246494,73438,129036,234015116,76220,326656,105271768,39481,4
40953,661103184];
c(10) = [];
c(7) = [];
c(4) = [];
c(1) = [];
sFilename='res5c.xlsb';
sSheetName='Sheet1';
xM=[M1,M2,M3,M5];
bRound=true;
nDigits = 10;
doAll2(maxElems,maxIters,c,sFilename,sSheetName,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,sSheetName,xMarr,bRound,nDigits)
global gmaxElems
global bestFactor
global M1
global M2
global M3
global M4
global doRounding
global numDigits

gmaxElems = maxElems;
M1 = xMarr(1);
M2 = xMarr(2);
M3 = xMarr(3);
M4 = xMarr(4);
doRounding = bRound;
numDigits = nDigits;
resMat=zeros(maxIters,13);
konst = 32358;
i = 0;
for iiw=konst:konst:maxIters
  for iix=konst:konst:maxIters
    for iiy=konst:konst:maxIters
      for iiz=konst:konst:maxIters
        iw = iiw - fix(rand*konst);
        ix = iix - fix(rand*konst);
        iy = iiy - fix(rand*konst);
        iz = iiz - fix(rand*konst);
        i = i + 1;
        factor = rng997Gen2(c,[iw,ix,iy,iz]);
        resMat(i,1) = factor;
        resMat(i,2) = iw;
        resMat(i,3) = c(1);
        resMat(i,4) = c(2);
        resMat(i,5) = ix;
        resMat(i,6) = c(3);
```



```
        resMat(i,7) = c(4);
        resMat(i,8) = iy;
        resMat(i,9) = c(5);
        resMat(i,10) = c(6);
        resMat(i,11) = iz;
        resMat(i,12) = c(7);
        resMat(i,13) = c(8);
        fprintf("Factor =  %f, X=[", factor);
        fprintf("%i, ", resMat(i,2:13));
        fprintf("]\n");
      end
    end
  end
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(1:13) = {'Factor', 'IW', 'C1','A1', 'IX', 'C2'
,'A2', 'IY', 'C3','A3', 'IZ', 'C4' ,'A4'};
writetable(T1, sFilename, "Sheet", sSheetName);

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

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

Listing 3.6 shows the source code of file rng977Gen2.m.

```
function factor = rng997Gen2(c, WXYZrnd)
%UNTITLED2 Summary of this function goes here
  global gmaxElems
  global bestFactor
  global bestX
  global M1
  global M2
  global M3
  global M4
  global doRounding
  global numDigits

  maxElems = gmaxElems;
  WXYZrnd = round(WXYZrnd, 0);
  c = round(c, 0);
  iw = WXYZrnd(1);
  c1 = c(1);
  a1 = c(2);
  ix = WXYZrnd(2);
  c2 = c(3);
```



```
  a2 = c(4);
  iy = WXYZrnd(3);
  c3 = c(5);
  a3 = c(6);
  iz = WXYZrnd(4);
  c4 = c(7);
  a4 = c(8);

  x=zeros(maxElems,1);
  for i=1:maxElems
    iw = mod(c1+a1*iw,M1);
    ix = mod(c2+a2*ix,M2);
    iy = mod(c3+a3*iy,M3);
    iz = mod(c4+a4*iz,M4);
    x(i) = mod(iy/M1 + ix/M2 + iy/M3 + iz/M4,1);
  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 3.6. 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 Init	Random Init	Random Init	Random Init	Random Init
Mean	147.7594927	147.7729728	147.7416605	147.7437161	147.7659187
Sdev	11.77083342	11.77119166	11.78039516	11.77348876	11.78015392
Min	104.4474955	105.0900988	105.0527588	100.9593134	105.7310653
Max	224.509828	233.5804061	221.8014568	223.5147722	229.014083
Range	120.0623325	128.4903074	116.748698	122.5554587	123.2830177
Count	810000	810000	810000	810000	810000
Conf	0.029314642	0.029315535	0.029338455	0.029321255	0.029337855
CI Upper	147.7888073	147.8022883	147.7709989	147.7730374	147.7952566
CI Lower	147.7301781	147.7436573	147.712322	147.7143949	147.7365809
IW	367724	740450	794653	322938	217616
C1	577299	623914	585386	330464	506256
A1	329338405	335061272	168723170	216472596	112246494
IX	517939	874092	202495	780908	351310
C2	886516	430065	151875	102590	129036
A2	442838110	700802204	405529740	199379830	234015116
IY	179626	837094	841226	600658	817683
C3	574755	388966	257214	216756	326656
A3	656589874	720560897	855579506	94	105271768
IZ	491174	351895	207163	95033	422675
C4	10	162179	56712	538031	440953
A4	443956272	850000000	955379239	499081728	661103184
Max Elements	10000	10000	10000	10000	10000
Max Iters	810000	810000	810000	810000	810000

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

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

iw = 585386+168723170*iw mod 16743091

ix = 151875+405529740*ix mod 16749377
iy = 257214+855579506*iy mod 16755383
iz = 56712+955379239*iz mod 16761491
r = iy/16743091 + ix/16749377 + iy/16755383 + iz/16761491 mod 1 (3.3)

The variables iw, ix, iy, and iz are initialized in special loops. See Listing 3.6 on how the code initializes these variables.The value r is the uniform random number generated in the range of 0 to 1 (excluded).

DOCUMENT HISTORY

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

