Quantum Shammas Polynomials
Part 1D of the Study

By
Namir Shammas

Contents

Introduction....2
The Quantum Shammas Polynomial Function....3
The Quantum Padé Shammas Polynomial Function....4
The Quantum Shammas Fourier Series Function....5
The PSO Function....5
The Random Search Function....8
The Halton Quasi Random Search Function....10
The Sobol Quasi Random Search Function....12
Testing Quantum Shammas Polynomials....14
Testing Bessel Function Fit with PSO-Run1....14
Testing Bessel Function Fit with PSO-Run2....18
Testing Bessel Function Fit with Random Search Optimization-Run1....22
Testing Bessel Function Fit with Random Search Optimization-Run2....26
Testing Bessel Function Fit with Halton Random Search Optimization-Run1....30
Testing Bessel Function Fit with Halton Random Search Optimization-Run2....34
Testing Bessel Function Fit with Sobol Random Search Optimization-Run1....38
Testing Bessel Function Fit with Sobol Random Search Optimization-Run2....42
Conclusion for Bessel Function Fitting....46
Testing ln(x) Function Fit with PSO....47
Testing ln(x) Function Fit with Random Search Optimization....50
Testing ln(x) Function Fit with Halton Random Search Optimization....54
Testing ln(x) Function Fit with Sobol Random Search Optimization....58

 Version 1.0.0

Conclusion for fitting the ln(x) Function 62
Testing the Right-Side Gauss-Bell Function Fit with PSO 63
Testing the Right-Side Gauss-Bell Function Fit with Random Search Optimization 66
Testing the Right-Side Gauss-Bell Function Fit with Halton Random Search Optimization..70
Testing the Right-Side Gauss-Bell Function Fit with Sobol Random Search Optimization..74
Conclusion for fitting the Right-Side Normal Gaussian Function 78
Conclusion for Part 1D 79
Next is Part 1E 79
Document History 79

Introduction

Part 1 of this study introduced you to Quantum Shammas Polynomials. Parts 1B and 1C showed you examples of using power ranges that are wider and narrower, respectively, that the one in Part 1. In this part, I present a version of the Quantum Shammas Polynomials that have power ranges that alternate between wide and narrow ranges. The equation for Quantum Shammas Polynomials is:

$$y(x) = a_0 + a_1*x^{r1} + a_2*x^{r2} + \ldots + a_n*x^{rn} \qquad \text{for } x>=0 \qquad (1)$$

In this part, the odd-indexed ranges (r_1, r_3, etc) have certain values that are bigger than those of even-indexed ranges (r_2, r_4, etc). The values of these ranges, which I chose arbitrarily, for the first four polynomial terms are:

1) The range for r_1 is (0.5, 1.5) with a span of 1 within the range.
2) The range for r_1 is (1.7, 2.3) with a span of 0.6 within the range and a gap of 0.2 with ranges r_1 and r_3.
3) The range for r_3 is (2.5, 3.5) with a span of 1 within the range and a gap of 0.2 with ranges r_2 and r_4.
4) The range for r_4 is (3.7, 4.3) with a span of 0.6 within the range and a gap of 0.2 with ranges r_3.

The limits of these ranges never overlap and have a gap of 0.2 between them. This gap ensures that no two random powers have the same exact value. The values of the random powers (r_i) are chosen to minimize the sum of errors squared between some observed values of y(x) and the ones calculated using equation (1). This minimization process involves optimization using either an optimization algorithm or random search. The latter method is feasible in the case of Quantum Shammas Polynomials because the ranges for the random powers are relatively small. This study shows using an evolutionary optimization algorithm, randoms search optimization, and quasi-random sequence search optimization (using the Holton and Sobol sequences).

The Quantum Shammas Polynomial Function

The Quantum Shammas Polynomial function in MATLAB is:

```
function SSE = quantShammasPoly(pwr)
  global xData yData yCalc glbRsqr QSPcoeff

  n = length(xData);
  order = length(pwr);
  SSE = 0;
  X = [1+zeros(n,1)];
  for j=1:order
    X = [X xData.^pwr(j)];
  end
  [QSPcoeff] = regress(yData,X);
  SSE = 0;
  SStot = 0;
  ymean = mean(yData);
  SStot = sum((yData - ymean).^2);
  yCalc = zeros(n,1);
  for i=1:n
    yCalc(i) = QSPcoeff(1);
    for j=1:order
      yCalc(i) = yCalc(i) + QSPcoeff(j+1)*xData(i)^pwr(j);
    end
    SSE = SSE + (yCalc(i) - yData(i))^2;
  end
  glbRsqr = 1 - SSE / SStot;
end
```

The above function takes one input parameter, the array of random powers pwr. The function returns the sum of errors squared. The function builds the regression matrix and calls function regress() to obtain the regression coefficients. The function then calculates the projected y values and uses them to calculate the result. The function also calculates the total sum of squared differences between the observed values and their mean value. Finally, the function calculates the coefficient of determination and stores it in the global variable glbRsqr. The function also uses global variables to access the x and y data, return the calculated values of y, and return the coefficients of the fitted Quantum Shammas Polynomial.

The Quantum Padé Shammas Polynomial Function

The Quantum Shammas Padé Polynomial function in MATLAB is:

```
function SSE = quantShammasPadéPoly(pwr)
  global xData yData yCalc glbRsqr QSPcoeff
  global orderP orderQ

  n = length(xData);
  order = length(pwr);
  SSE = 0;
  X = [1+zeros(n,1)];
  for j=1:orderP
    X = [X xData.^pwr(j)];
  end
  for j=1:orderQ
     k = orderP + j;
     X = [X -yData.*xData.^pwr(k)];
  end
  [QSPcoeff] = regress(yData,X);
  SSE = 0;
  SStot = 0;
  ymean = mean(yData);
  SStot = sum((yData - ymean).^2);
  yCalc = zeros(n,1);
  for i=1:n
    sumP = QSPcoeff(1);
    for j=1:orderP
      sumP = sumP + QSPcoeff(j+1)*xData(i)^pwr(j);
    end
    sumQ = 1;
    for j=1:orderQ
      k = orderP + j;
      sumQ = sumQ - QSPcoeff(k+1)*yData(i)*xData(i)^pwr(k);
    end
    yCalc(i) = sumP / sumQ;
    SSE = SSE + (yCalc(i) - yData(i))^2;
  end
  glbRsqr = 1 - SSE / SStot;
```



```
end
```

The above function resembles the quantShammasPoly() except it performs a Padé polynomial fit and calculations for the projected y data. The function returns the sum of errors squared. The function also calculates the coefficient of determination and stores it in the global variable glbRsqr. The function also uses global variables to access the x and y data, return the calculated values of y, and return the coefficients of the fitted Quantum Shammas Polynomial.

The Quantum Shammas Fourier Series Function

The Quantum Shammas Fourier Series (Gen 1) function in MATLAB is:

```
function SSE = quantShammasFourierPoly(pwr)
  global xData yData yCalc glbRsqr QSPcoeff
  n = length(xData);
  order = length(pwr);
  X = [1+zeros(n,1)];
  for j=1:2:order
    X = [X sin(pwr(j)*xData) cos(pwr(j+1)*xData)];
  end
  [QSPcoeff] = regress(yData,X);
  SSE = 0;
  ymean = mean(yData);
  SStot = sum((yData - ymean).^2);
  yCalc = zeros(n,1);
  for i=1:n
    yCalc(i) = QSPcoeff(1);
    for j=2:2:order
      yCalc(i) = yCalc(i) + QSPcoeff(j)*sin(pwr(j-1)*xData(i)) + ...
                            QSPcoeff(j+1)*cos(pwr(j)*xData(i));
    end
    SSE = SSE + (yCalc(i) - yData(i))^2;
  end
  glbRsqr = 1 - SSE / SStot;
end
```

The above function resembles the quantShammasPoly() except it performs a Fourier series fit (with sine and cosine terms) and calculations for the projected y data. The function returns the sum of errors squared. The function also calculates the coefficient of determination and stores it in the global variable glbRsqr.

The PSO Function

The next function implements the Particle Swarm Optimization (PSO) algorithm:


```
function [bestX,bestFx] = psox(fx,Lb,Ub,MaxPop,MaxIters,bShow)
% PSOX implements particle swarm optimization.
%
%
% INPUT
% ======
% fx - handle of optimized function.
% Lb - array of low bound values.
% Ub - array of upper bound values.
% MaxPop - maximum population of swarm.
% MaxIters - maximum number of iterations
% bShow - Boolean flag to request viewing intermediate results.
%
% OUTPUT
% ======
% bestX - array of best solutions.
% bestFx - best optimized function value.
%
% Example
% =======
%
% >>
%
  if nargin < 6, bShow = false; end
  n = length(Lb);
  m = n + 1;
  pop = 1e+99+zeros(MaxPop,m);
  pop2 = pop;
  aPop = zeros(1,n);
  vel = zeros(MaxPop,n);

  % Initizialize population
  for i=1:MaxPop
    pop(i,1:n) = Lb + (Ub - Lb) .* rand(1,n);
    vel(i,1:n) = (Ub - Lb) / 10 .* (2*rand(1,n)-1);
    pop(i,m) = fx(pop(i,1:n));
    pop2(i,:) = pop(i,:);
    aPop(1:n) = Lb + (Ub - Lb) .* rand(1,n);
    f0 = fx(aPop);
    if f0 < pop2(i,m)
      pop2(i,1:n) = aPop(1:n);
      pop2(i,m) = f0;
    end
  end

  pop = sortrows(pop,m);
  pop2 = pop;

  if bShow
    fprintf('Best X =');
    fprintf(' %f,', pop(1,1:n));
    fprintf('Best Fx = %e\n', pop(1,m));
```

```
  end
  bestFx = pop(1,m);

  % pso loop
  for iter = 1:MaxIters

    IterFactor = sqrt((iter - 1)/(MaxIters - 1));
    w = 1 - 0.3 * IterFactor;
    c1 = 2 - 1.9 * IterFactor;
    c2 = 2 - 1.9 * IterFactor;

    for i=2:MaxPop
      for j=1:n
        vel(i,j) = w*vel(i,j) + c1*rand*(pop(1,j) - pop(i,j)) + ...
          c2*rand*(pop2(i,j) - pop(i,j));
        p = pop(i,j) + vel(i,j);

        if p < Lb(j) || p > Ub(j)
          pop(i,j) = Lb(j) + (Ub(j) - Lb(j))*rand;
        else
          pop(i,j) = p;
        end
      end

      pop(i,m) = fx(pop(i,1:n));

      % find new global best?
      if pop(1,m) > pop(i,m)
        pop(1,:) = pop(i,:);
        % find new local best?
      elseif pop(i,m) < pop2(i,m)
        pop2(i,:) = pop(i,:);
      end
    end

    [pop,Idx] = sortrows(pop,m);
    pop2 = sortrows(pop2,m);
    vel = vel(Idx,:);

    if bestFx > pop(1,m)
      if bShow
        fprintf('%i: Best X = %i', iter);
        fprintf(' %f,', pop(1,1:n));
        fprintf('Best Fx = %e\n', pop(1,m));
      end
      bestFx = pop(1,m);
    end
  end
  bestFx = pop(1,m);
  bestX = pop(1,1:n);
end
```

The function has the following input parameters:

- The parameter fx is the handle of the optimized function.
- The parameter Lb is the row array of low bound values.
- The parameter Ub is the row array of upper bound values.
- The parameter MaxPop is the maximum population of swarm.
- The parameter MaxIters is the maximum number of iterations
- The parameter bShow is the Boolean flag to request viewing intermediate results.

The output parameters are:

- The parameter bestX is the array of best solutions.
- The parameter bestFx is the best optimized function value.

The Random Search Function

The next function performs a random search optimization:

```
function [bestX,bestFx] = randomSearch(fx,Lb,Ub,MaxIters)
% RANDOMSEARCH performs random search optimization.
%
%
% INPUT
% ======
% fx - handle of optimized function.
% Lb - array of low bound values.
% Ub - array of upper bound values.
% MaxIters - maximum number of iterations
%
% OUTPUT
% ======
% bestX - array of best solutions.
% bestFx - best optimized function value.

  bestFx = 1e99;
  n = length(Lb);
  bestX = 1e+99+zeros(n,1);
  for irun=1:2
    for iter = 1:MaxIters
      X = Lb + (Ub - Lb).*rand(1,n);
      f = fx(X);
      if f < bestFx
```

```
        bestFx = f;
        bestX = X;
        k = iter + (irun-1) *MaxIters;
        fprintf("%7i: Fx = %e, X=[", k, bestFx);
        fprintf("%f, ", X)
        fprintf("]\n");
      end
    end

    delta = 0.15;
    deltaMin = 0.05;
    bExit = false;
    bChanged = true;
    while delta >= deltaMin && bChanged
      for i=1:n
        if bestX(i) > 0
          Lb(i) = (1-delta)*bestX(i);
          Ub(i) = (1+delta)*bestX(i);
        else
          Lb(i) = (1+delta)*bestX(i);
          Ub(i) = (1-delta)*bestX(i);
        end
      end
      % check if neighboring bounds are too close
      bChanged = false;
      for i=1:n-1
        d = round(Lb(i+1),0)- round(Ub(i),0);
        if d == 0
          delta = delta - deltaMin;
          bChanged = true;
          break;
        end
      end
      if delta == 0
        bChanged = false;
        bExit = true;
      end
    end

    if bExit, break; end
    Lb
    Ub
  end
end
```

The function has the following input parameters:

- The parameter fx is the handle of the optimized function.
- The parameter Lb is the row array of low bound values.
- The parameter Ub is the row array of upper bound values.
- The parameter MaxIters is the maximum number of iterations

The output parameters are:

- The parameter bestX is the array of best solutions.
- The parameter bestFx is the best optimized function value.

The above function is easy to code and works well with Quantum Shammas Polynomials since the range of each power is relatively small (<2). The above improvement performs two passes for the random search. The first pass uses the lower and upper ranges (in parameters Lb and Ub) that are supplied to the function. The second pass narrows the values of arrays Lb and Ub to closely bracket the best values of X obtained at the end of the first pass.

The Halton Quasi Random Search Function

The next function performs random-search optimization using the Halton quasi-random sequences:

```
function [bestX,bestFx] = haltonRandomSearch(fx,Lb,Ub,MaxIters)
% HALTONRANDOMSEARCH performs optimization using the Halton
quasi-random sequence.
%
%
% INPUT
% ======
% fx - handle of optimized function.
% Lb - array of low bound values.
% Ub - array of upper bound values.
% MaxIters - maximum number of iterations
%
% OUTPUT
% ======
% bestX - array of best solutions.
% bestFx - best optimized function value.

  bestFx = 1e99;
  n = length(Lb);
  bestX = 1e+99+zeros(n,1);
```

```
% set up halton sequences
p = haltonset(n,'Skip',1e3,'Leap',1e2);
p = scramble(p,'RR2');
rando = net(p,MaxIters);
for irun=1:2
  for iter = 1:MaxIters
    for i=1:n
      X(i) = Lb(i) + (Ub(i) - Lb(i))*rando(iter,i);
    end
    f = fx(X);
    if f < bestFx
      bestFx = f;
      bestX = X;
      k = iter + (irun-1) *MaxIters;
      fprintf("%7i: Fx = %e, X=[", k, bestFx);
      fprintf("%f, ", X)
      fprintf("]\n");
    end
  end

  delta = 0.15;
  deltaMin = 0.05;
  bExit = false;
  bChanged = true;
  while delta >= deltaMin && bChanged
    for i=1:n
      if bestX(i) > 0
        Lb(i) = (1-delta)*bestX(i);
        Ub(i) = (1+delta)*bestX(i);
      else
        Lb(i) = (1+delta)*bestX(i);
        Ub(i) = (1-delta)*bestX(i);
      end
    end
    % check if neighboring bounds are too close
    bChanged = false;
    for i=1:n-1
      d = round(Lb(i+1),0)- round(Ub(i),0);
      if d == 0
        delta = delta - deltaMin;
        bChanged = true;
        break;
      end
    end
    if delta == 0
      bChanged = false;
      bExit = true;
```



```
      end
    end

    if bExit, break; end
    Lb
    Ub
  end
end
```

The above function has the same input and output parameters as the randomSearch() function. The above code shows lines in red that highlight the statements that generate multiple columns of the Halton sequence and stores them in the matrix rando. The function accesses the various elements of matrix rando as pseudo-random numbers are needed.

The Sobol Quasi Random Search Function

The next function performs random-search optimization using the Sobol quasi-random sequences:

```
function [bestX,bestFx] = sobolRandomSearch(fx,Lb,Ub,MaxIters)
% SOBOLRANDOMSEARCH performs optimization using the Sobol quasi-
random sequence.
%
%
% INPUT
% ======
% fx - handle of optimized function.
% Lb - array of low bound values.
% Ub - array of upper bound values.
% MaxIters - maximum number of iterations
%
% OUTPUT
% ======
% bestX - array of best solutions.
% bestFx - best optimized function value.

  bestFx = 1e99;
  n = length(Lb);
  bestX = 1e+99+zeros(n,1);

  % set up Sobol sequences
  p = sobolset(n,'Skip',1e3,'Leap',1e2);
  p = scramble(p,'MatousekAffineOwen');
  rando = net(p,MaxIters);
  for irun=1:2
```

```
for iter = 1:MaxIters
  for i=1:n
    X(i) = Lb(i) + (Ub(i) - Lb(i))*rando(iter,i);
  end
  f = fx(X);
  if f < bestFx
    bestFx = f;
    bestX = X;
    k = iter + (irun-1) *MaxIters;
    fprintf("%7i: Fx = %e, X=[", k, bestFx);
    fprintf("%f, ", X)
    fprintf("]\n");
  end
end

delta = 0.15;
deltaMin = 0.05;
bExit = false;
bChanged = true;
while delta >= deltaMin && bChanged
  for i=1:n
    if bestX(i) > 0
      Lb(i) = (1-delta)*bestX(i);
      Ub(i) = (1+delta)*bestX(i);
    else
      Lb(i) = (1+delta)*bestX(i);
      Ub(i) = (1-delta)*bestX(i);
    end
  end
  % check if neighboring bounds are too close
  bChanged = false;
  for i=1:n-1
    d = round(Lb(i+1),0)- round(Ub(i),0);
    if d == 0
      delta = delta - deltaMin;
      bChanged = true;
      break;
    end
  end
  if delta == 0
    bChanged = false;
    bExit = true;
  end
end

if bExit, break; end
Lb
```

```
    Ub
  end
end
```

The above function has the same input and output parameters as the randomSearch() function. The above code shows lines in red that highlight the statements that generate multiple columns of the Sobol sequence and store them in the matrix rando. The function accesses the various elements of matrix rando as pseudo-random numbers are needed.

Testing Quantum Shammas Polynomials

The next sections show examples of using the Quantum Shammas Polynomials to fit a selection of arbitrary functions. The results of the Quantum Shammas Polynomials are compared with those of classical polynomials. The adjusted coefficient of determinations are good indicators of how the two types of polynomial stack up against each other.

Testing Bessel Function Fit with PSO-Run1

The next MATLAB script (found in file testBessel1pso.m) tests fitting Bessel J(0, x) for x in the range (0, 5) and samples at 0.1 steps. The curve fits use a fourth order Quantum Shammas Polynomial and a fourth order classical polynomial.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_run1";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "besselj(0,x)";
fprintf(sEqn);
fprintf("x=0:0.1:5\n")
xData= 0:0.1:5;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.5, 1.7, 2.3);
```

```
[bestX,bestFx] = psox(@quantShammasPoly,Lb,Ub,1000,5000,true);

SSE = quantShammasPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Polynomial Powers\n");
bestX
fprintf("Quantum Shammas Polynomial Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular polynomial fit\n");
c = polyfit(xData,yData,order)
yPoly = polyval(c,xData);
r = rsqr(yData,yPoly);
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A1");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A4");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A7");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr1, maxPwr1, minPwr2,
maxPwr2)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  delta1 = maxPwr1 - minPwr1;
```



```
  delta2 = maxPwr2 - minPwr2;
  gap = minPwr2 - maxPwr1;
  Lb(1) = minPwr1;
  Ub(1) = maxPwr1;
  for i=2:order
    if mod(i,2)>0
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta1;
    else
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta2;
    end
  end
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

The above code copies the console output to a diary text file. It also writes the summary results to an Excel table, shown below:

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	
0.766113533	2.299726913	3.498583136	4.294254393	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
1.001265312	-0.020919321	-0.276014328	0.071689049	-0.00988616
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.980927341	0.138170634	-0.457980428	0.113695746	-0.007357698
r_sqr1	r_sqr2			
0.999997844	0.999803041			

Table 1. Summary of the results appearing in file besselj_0_x_run1.xlsx.

The second row shows the powers for the fitted Quantum Shammas Polynomial. The fifth row shows the intercept (below QSPcoeff1) and to its right the coefficients for the other coefficients of the Quantum Shammas Polynomial. The eighth row shows the intercept and coefficients for the classical polynomial. The cell under r_sqr1 shows the adjusted coefficient of determination for the fitted Quantum Shammas Polynomial. The cell under r_sqr2 shows the adjusted coefficient of determination for the fitted classical polynomial. The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is higher than the one for the classical polynomial. This condition indicates that the Quantum Shammas Polynomial performs a better fit for the above example.

Here is the graph (from file besselj_0_x_run1.jpg) for the Bessel function and the two fitted polynomials:

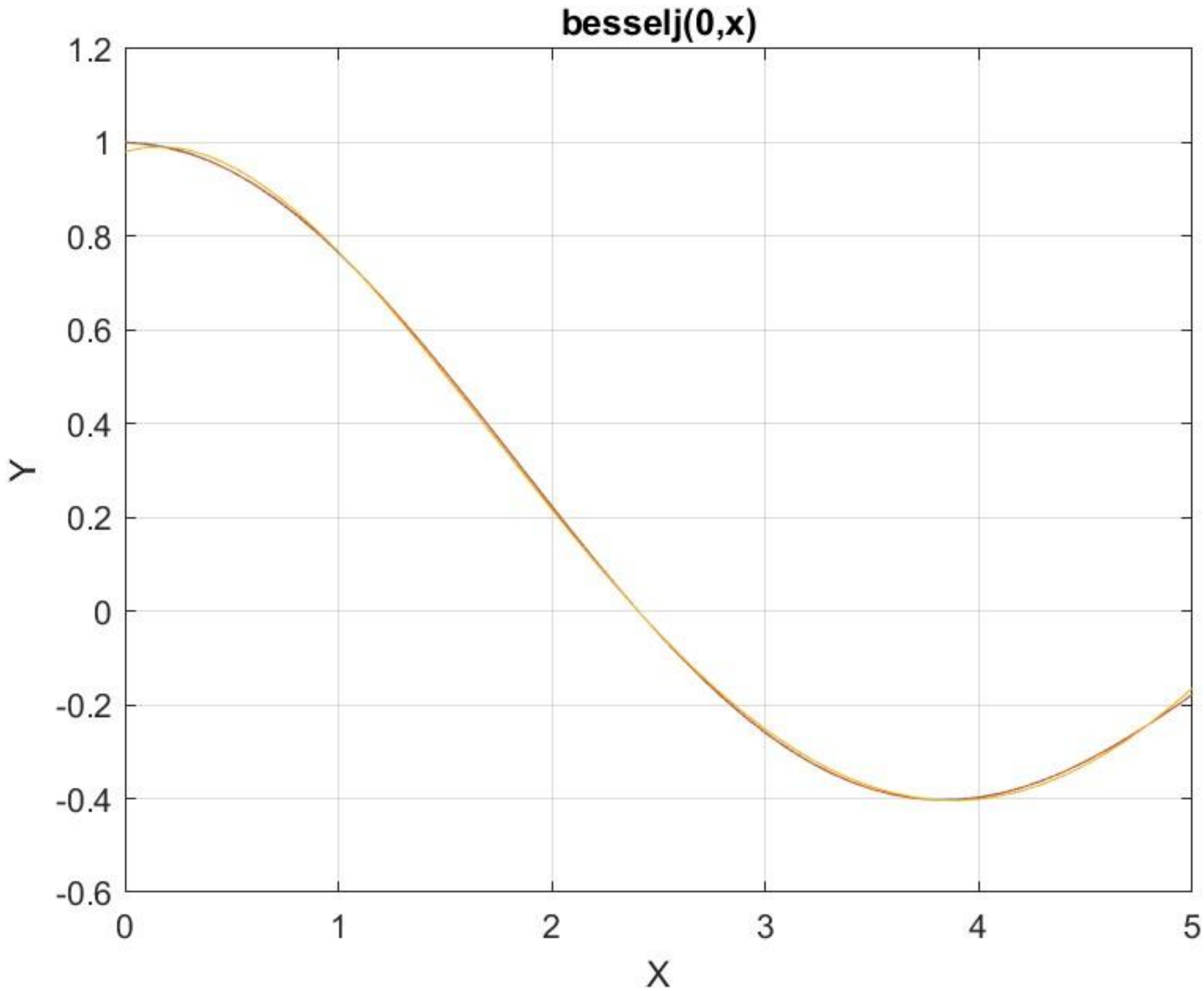

Figure 1. The graph from file besselj_0_x_run1.jpg.

The above graph shows a reasonably good fit for both polynomials. Keep in mind that the Quantum Shammas Polynomial is slightly better than the one for the classical polynomial

Testing Bessel Function Fit with PSO-Run2

The next MATLAB script (found in file testBessel2pso.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial and a sixth order classical polynomial.


```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_run2";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "besselj(0,x)";
fprintf(sEqn);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(order, 0.5, 1.5, 1.7, 2.3);

[bestX,bestFx] = psox(@quantShammasPoly,Lb,Ub,1000,5000,true);

SSE = quantShammasPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Polynomial Powers\n");
bestX
fprintf("Quantum Shammas Polynomial Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular polynomial fit\n");
c = polyfit(xData,yData,order)
yPoly = polyval(c,xData);
r = rsqr(yData,yPoly);
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);
```

```
QSPpwr = bestX;
Coeff = flip(c);
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A1");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A4");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A7");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr1, maxPwr1, minPwr2, 
maxPwr2)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  delta1 = maxPwr1 - minPwr1;
  delta2 = maxPwr2 - minPwr2;
  gap = minPwr2 - maxPwr1;
  Lb(1) = minPwr1;
  Ub(1) = maxPwr1;
  for i=2:order
    if mod(i,2)>0
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta1;
    else
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta2;
    end
  end
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

In the above code, each calls to function psox()) performs a PSO search using a population size of 1000 and 5000 maximum iterations. The above code copies the console output to a diary text file. It also writes the summary results to an Excel table, shown below:

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
1.496676211	2.296721037	3.499666852	4.2368365	5.464643311	6.278845477	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
0.967697087	0.222450939	-0.543360839	0.181328362	-0.043681228	0.001401985	-6.76896E-05
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.998806149	0.996718149					

Table 2. Summary of the results appearing in file besselj_0_x_run2.xlsx.

The second row shows the powers for the fitted Quantum Shammas Polynomial. The fifth row shows the intercept (below QSPcoeff1) and to its right the coefficients for the rest of the coefficients of the Quantum Shammas Polynomial. The eighth row shows the intercept and coefficients for the classical polynomial. The cell under r_sqr1 shows the adjusted coefficient of determination for the fitted Quantum Shammas Polynomial. The cell under r_sqr2 shows the adjusted coefficient of determination for the fitted classical polynomial. The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is slightly higher than the one for the classical polynomial. This condition indicates that the Quantum Shammas Polynomial performs a better fit for the above example.

Note that the adjusted coefficient of determination for the fitted Quantum Shammas Polynomial in Table 2 is slightly higher than the one in Table 1. Since both methods used involve random numbers, I do not consider the difference as significant. It does show that the random search method surprisingly performs as well as the PSO method!

Here is the graph (from file besselj_0_x_run2.jpg) for the Bessel function and the two fitted polynomials:

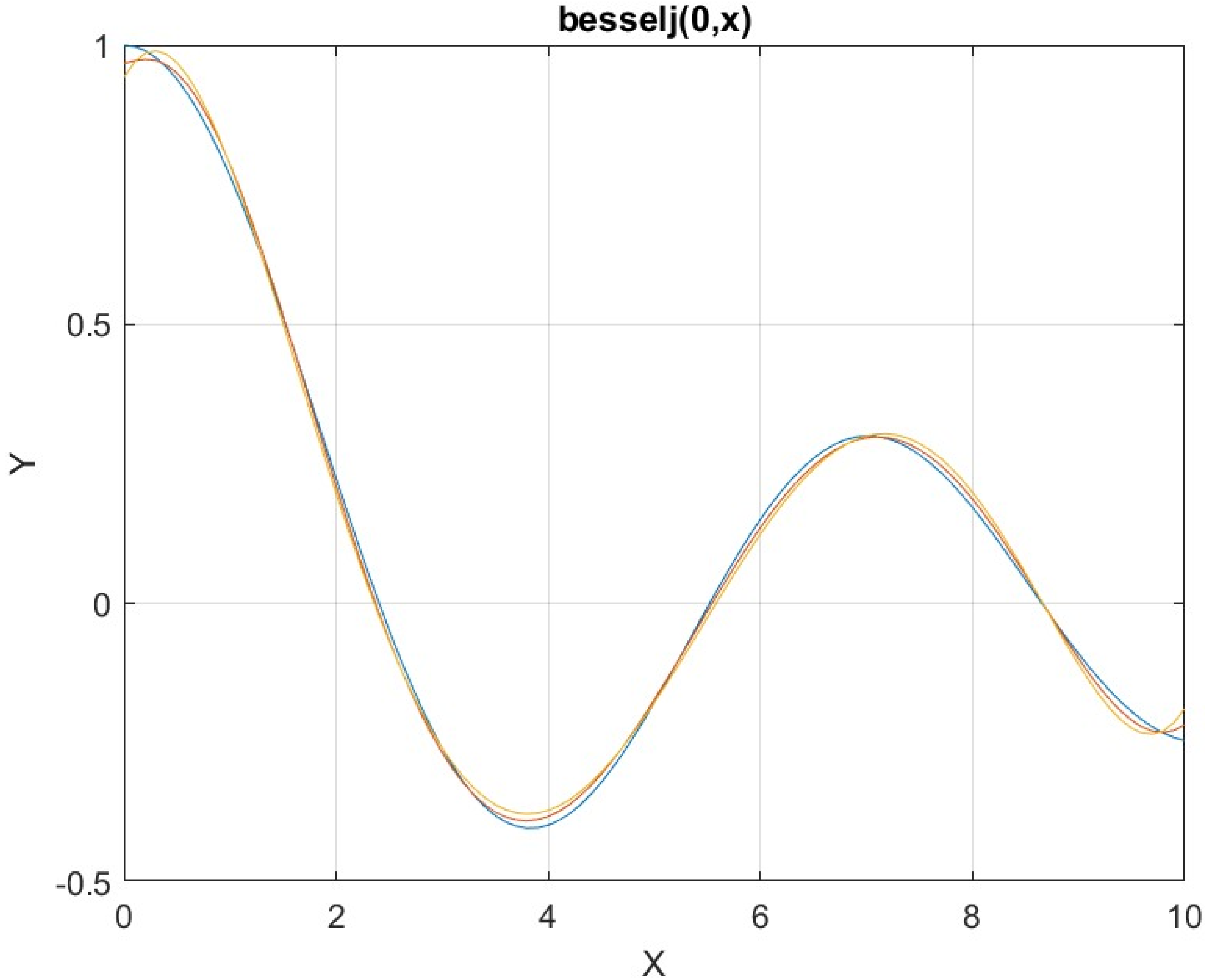

Figure 2. The graph from file besselj_0_x_run2.jpg .

The above graphs let you detect some slight deviations between the Bessel function and the two fitted polynomials. This is not unexpected since I have doubled the upper limit of the range of x from 5 to 10.

Testing Bessel Function Fit with Random Search Optimization-Run1

The next MATLAB script (found in file testBessel1Random.m) tests fitting Bessel J(0, x) for x in the range (0, 5) and samples at 0.1 steps. The curve fits use a fourth order Quantum Shammas Polynomial and a fourth order classical polynomial.

```
clc
clear
close all
```

```
global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_random_run1";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "besselj(0,x)";
fprintf(sEqn);
fprintf("x=0:0.1:5\n")
xData= 0:0.1:5;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.5, 1.7, 2.3);

[bestX,bestFx] = randomSearch(@quantShammasPoly,Lb,Ub,1000000);

SSE = quantShammasPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Polynomial Powers\n");
bestX
fprintf("Quantum Shammas Polynomial Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular polynomial fit\n");
c = polyfit(xData,yData,order)
yPoly = polyval(c,xData);
r = rsqr(yData,yPoly);
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
T1 = array2table(QSPpwr);
```



```
writetable(T1,xlFile,"Sheet","Sheet1","Range","A1");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A4");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A7");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr1, maxPwr1, minPwr2,
maxPwr2)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  delta1 = maxPwr1 - minPwr1;
  delta2 = maxPwr2 - minPwr2;
  gap = minPwr2 - maxPwr1;
  Lb(1) = minPwr1;
  Ub(1) = maxPwr1;
  for i=2:order
    if mod(i,2)>0
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta1;
    else
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta2;
    end
  end
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

The above script uses random search optimization by calling function randomSearch() and requests a million random searches. The above code copies the console output to a diary text file. It also writes the summary results to an Excel table, shown below:

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	
0.91709331	2.240315179	3.81358442	4.121911307	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
1.001100624	-0.022892391	-0.255085298	0.081079599	-0.038437534
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.980927341	0.138170634	-0.457980428	0.113695746	-0.007357698
r_sqr1	r_sqr2			
0.999998535	0.999803041			

Table 3. Summary of the results appearing in file besselj_0_x_random_run1.xlsx.

The above table shows similar types of results as the ones in Table 1 and Table 2. Again, the adjusted coefficient of determination for the Quantum Shammas Polynomial is higher than that for the classical polynomial. Both are good values.

Here is the graph (from file besselj_0_x_random_run1.jpg) for the Bessel function and the two fitted polynomials:

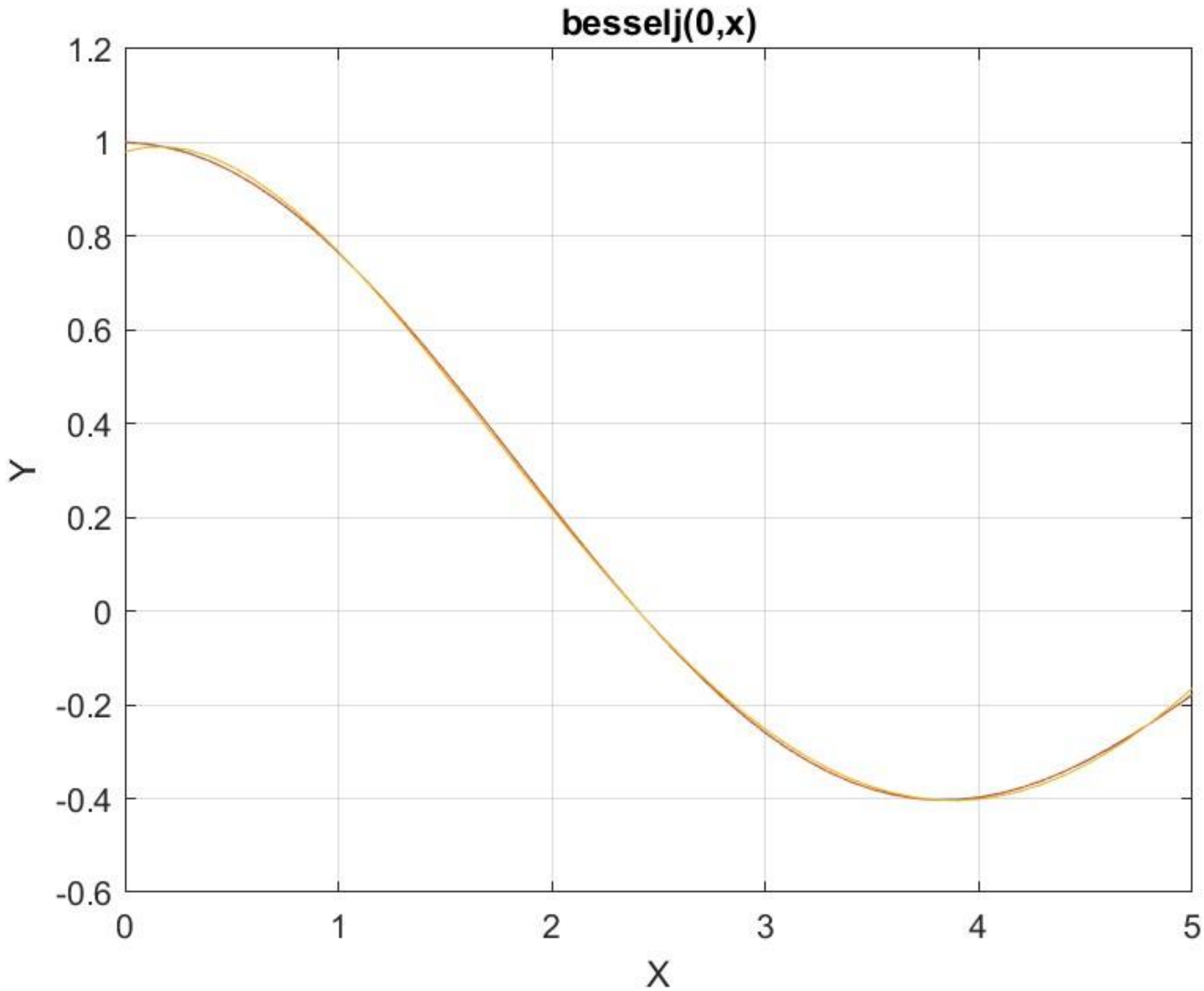

Figure 3. The graph from file besselj_0_x_random_run1.jpg.

The figure shows that both types of polynomials fit the Bessel function well.

Testing Bessel Function Fit with Random Search Optimization-Run2

The next MATLAB script (found in file testBessel2Random.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial and a sixth order classical polynomial.

```
clc
clear
close all
```

```
global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_random_run2";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "besselj(0,x)";
fprintf("%s\n", sEqn);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(order, 0.5, 1.5, 1.7, 2.3);

[bestX,bestFx] = randomSearch(@quantShammasPoly,Lb,Ub,1000000);

SSE = quantShammasPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Polynomial Powers\n");
bestX
fprintf("Quantum Shammas Polynomial Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular polynomial fit\n");
c = polyfit(xData,yData,order)
yPoly = polyval(c,xData);
r = rsqr(yData,yPoly);
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
T1 = array2table(QSPpwr);
```

```
writetable(T1,xlFile,"Sheet","Sheet1","Range","A1");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A4");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A7");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr1, maxPwr1, minPwr2,
maxPwr2)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  delta1 = maxPwr1 - minPwr1;
  delta2 = maxPwr2 - minPwr2;
  gap = minPwr2 - maxPwr1;
  Lb(1) = minPwr1;
  Ub(1) = maxPwr1;
  for i=2:order
    if mod(i,2)>0
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta1;
    else
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta2;
    end
  end
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

The above script uses random search optimization by calling function randomSearch() and requests a million random searches. The above code is very similar to the one before it. The differences are in the names of the output files and the range of x. The above code copies the console output to a diary text file. It also writes the summary results to an Excel table, shown below:

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
1.471129989	2.490811225	3.789273019	4.670677265	5.975886814	6.783246831	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
0.985939304	-0.001890649	-0.278459568	0.08785271	-0.017568416	0.000630699	-3.7781E-05
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.999738603	0.996718149					

Table 4. Summary of the results appearing in file besselj_0_x_random_run2.xlsx.

The above table shows similar types of results as the ones in Table 1 and Table 2. Again, the adjusted coefficient of determination for the Quantum Shammas Polynomial is slightly higher than that for the classical polynomial.

Here is the graph (from file besselj_0_x_random_run2.jpg) for the Bessel function and the two fitted polynomials:

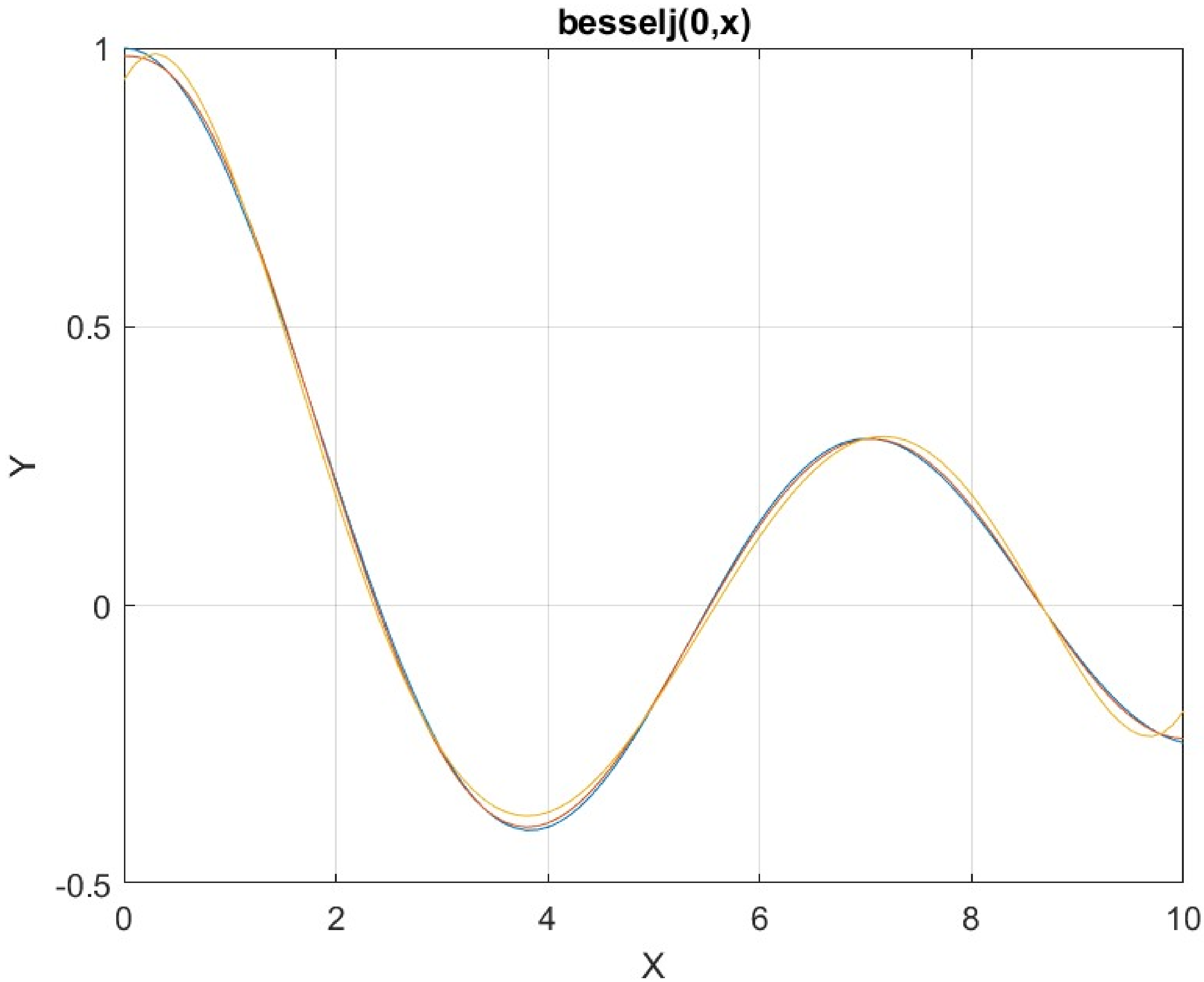

Figure 4. The graph from file besselj_0_x_random_run2.jpg.

The above graphs let you detect some slight deviations between the Bessel function and the two fitted polynomials. This is not unexpected since I have doubled the upper limit of the range of x from 5 to 10.

Testing Bessel Function Fit with Halton Random Search Optimization-Run1

The next MATLAB script (found in file testBessel1Halton.m) tests fitting Bessel J(0, x) for x in the range (0, 5) and samples at 0.1 steps. The curve fits use a fourth order Quantum Shammas Polynomial and a fourth order classical polynomial.

```
clc
clear
close all
```

```
global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_halton_random_run1";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "besselj(0,x)";
fprintf("%s\n",sEqn);
fprintf("x=0:0.1:5\n")
xData= 0:0.1:5;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.5, 1.7, 2.3);

[bestX,bestFx] =
haltonRandomSearch(@quantShammasPoly,Lb,Ub,1000000);

SSE = quantShammasPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Polynomial Powers\n");
bestX
fprintf("Quantum Shammas Polynomial Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular polynomial fit\n");
c = polyfit(xData,yData,order)
yPoly = polyval(c,xData);
r = rsqr(yData,yPoly);
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);
QSPpwr = bestX;
Coeff = flip(c);
```

```
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A1");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A4");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A7");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr1, maxPwr1, minPwr2, 
maxPwr2)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  delta1 = maxPwr1 - minPwr1;
  delta2 = maxPwr2 - minPwr2;
  gap = minPwr2 - maxPwr1;
  Lb(1) = minPwr1;
  Ub(1) = maxPwr1;
  for i=2:order
    if mod(i,2)>0
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta1;
    else
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta2;
    end
  end
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

The above script uses random search optimization by calling function haltonRandomSearch() and requests a million random searches. The above code is like the one in first random search optimization program. The main difference is that the above code uses functions that involve the Halton quasi-random sequence. Running the above code produces the following Excel table:

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	
0.838502519	2.223110895	3.814380804	4.146430872	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
1.001161815	-0.019630073	-0.256923923	0.074714453	-0.033552986
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.980927341	0.138170634	-0.457980428	0.113695746	-0.007357698
r_sqr1	r_sqr2			
0.999998466	0.999803041			

Table 5. Summary of the results appearing in file besselj_0_x_halton_random_run1.xlsx.

The above table shows similar types of results as the ones in Table 1 and Table 2. Again, the adjusted coefficient of determination for the Quantum Shammas Polynomial is higher than that for the classical polynomial. Both are good values. Using the Halton sequence gives surprisingly good results. I suspect using one million iterations has something to do with it.

Here is the graph (from file besselj_0_x_halton_random_run1.jpg) for the Bessel function and the two fitted polynomials:

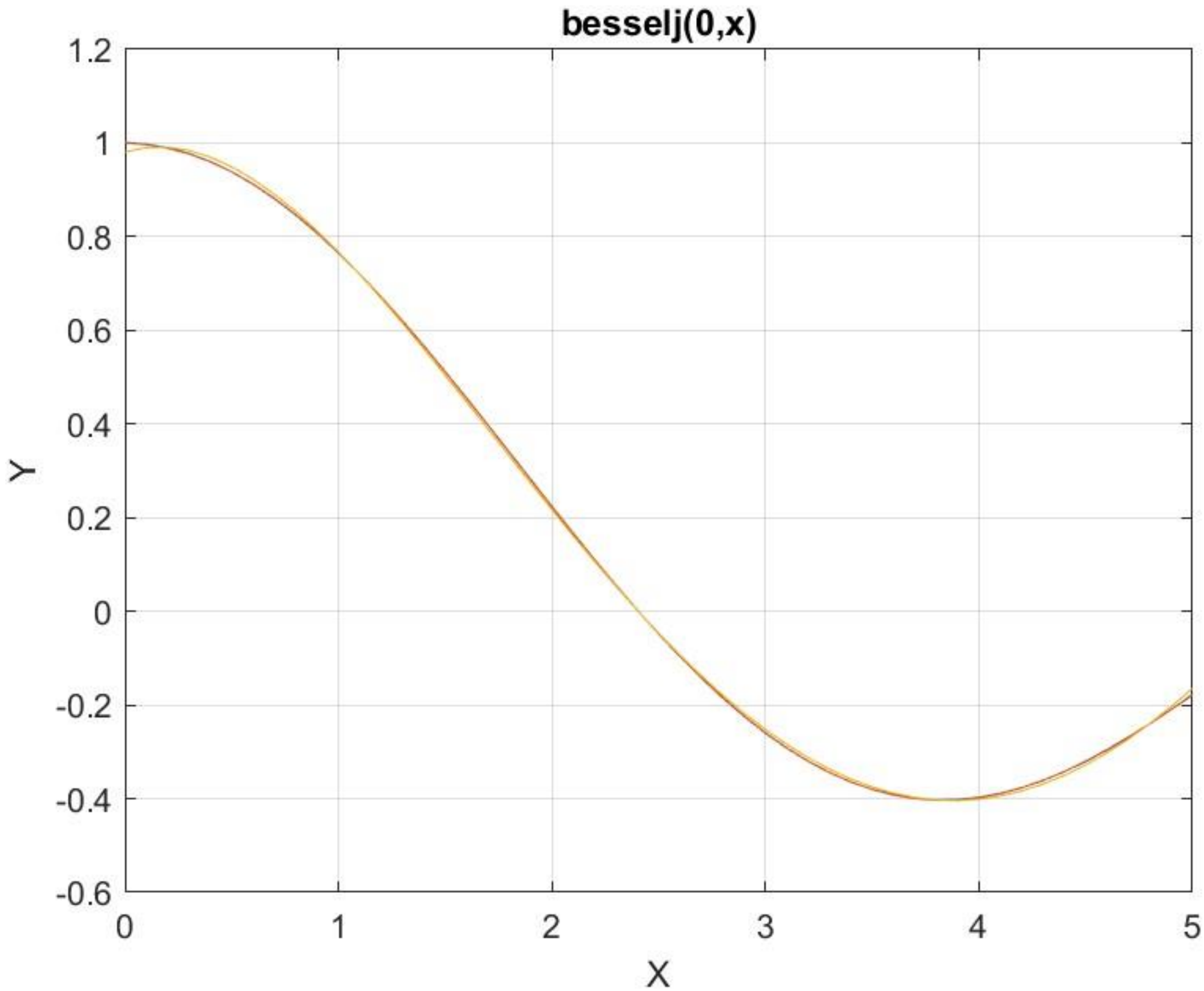

Figure 5. The graph from file besselj_0_x_halton_random_run1.jpg.

The figure shows that both types of polynomials fit the Bessel function well.

Testing Bessel Function Fit with Halton Random Search Optimization-Run2

The next MATLAB script (found in file testBessel2Halton.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial and a sixth order classical polynomial.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
```



```
zFilename = "besselj_0_x_halton_random_run2";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "besselj(0,x)";
fprintf("%s\n",sEqn);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(order, 0.5, 1.5, 1.7, 2.3);

[bestX,bestFx] =
haltonRandomSearch(@quantShammasPoly,Lb,Ub,1000000);

SSE = quantShammasPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Polynomial Powers\n");
bestX
fprintf("Quantum Shammas Polynomial Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular polynomial fit\n");
c = polyfit(xData,yData,order)
yPoly = polyval(c,xData);
r = rsqr(yData,yPoly);
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
T1 = array2table(QSPpwr);
```

```
writetable(T1,xlFile,"Sheet","Sheet1","Range","A1");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A4");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A7");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr1, maxPwr1, minPwr2,
maxPwr2)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  delta1 = maxPwr1 - minPwr1;
  delta2 = maxPwr2 - minPwr2;
  gap = minPwr2 - maxPwr1;
  Lb(1) = minPwr1;
  Ub(1) = maxPwr1;
  for i=2:order
    if mod(i,2)>0
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta1;
    else
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta2;
    end
  end
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

The above script uses random search optimization by calling function haltonRandomSearch() and requests a million random searches. The above code is very similar to the one before it. The differences are the names of the files and the range for x. The above code produces the following Excel table:

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
1.345272192	2.515237649	3.805536898	4.662782744	5.99095649	6.703197674	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
0.987239028	-0.010214761	-0.272434219	0.089617199	-0.018782304	0.000674102	-5.55858E-05
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.999747391	0.996718149					

Table 6. Summary of the results appearing in file besselj_0_x_halton_random_run2.xlsx.

The above table shows similar types of results as the ones in Table 1 and Table 2. Again, the adjusted coefficient of determination for the Quantum Shammas Polynomial is slightly higher than that for the classical polynomial.

Here is the graph (from file besselj_0_x_halton_random_run2.jpg) for the Bessel function and the two fitted polynomials:

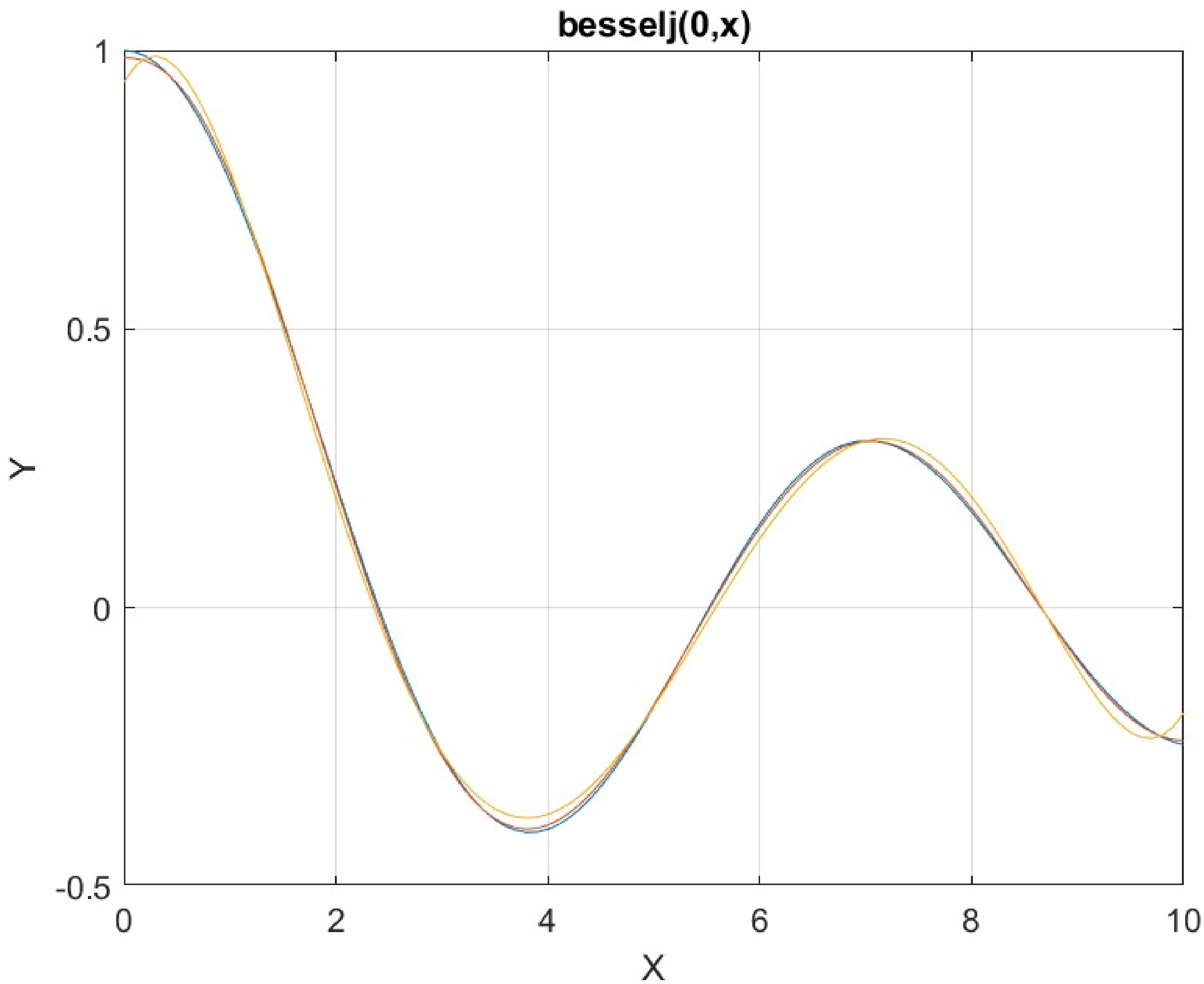

Figure 6. The graph from file besselj_0_x_halton_random_run2.jpg.

The curves in the above figure shows some deviations between the two polynomials and the curve for the Bessel function.

Testing Bessel Function Fit with Sobol Random Search Optimization-Run1

The next MATLAB script (found in file testBessel1Sobol.m) tests fitting Bessel J(0, x) for x in the range (0, 5) and samples at 0.1 steps. The curve fits use a fourth order Quantum Shammas Polynomial and a fourth order classical polynomial.

```
clc
clear
close all
```

```
global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_sobol_random_run1";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "besselj(0,x)";
fprintf("%s\n",sEqn);
fprintf("x=0:0.1:5\n")
xData= 0:0.1:5;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.5, 1.7, 2.3);

[bestX,bestFx] =
sobolRandomSearch(@quantShammasPoly,Lb,Ub,1000000);

SSE = quantShammasPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Polynomial Powers\n");
bestX
fprintf("Quantum Shammas Polynomial Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular polynomial fit\n");
c = polyfit(xData,yData,order)
yPoly = polyval(c,xData);
r = rsqr(yData,yPoly);
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
```

```
Coeff = flip(c);
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A1");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A4");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A7");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr1, maxPwr1, minPwr2,
maxPwr2)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  delta1 = maxPwr1 - minPwr1;
  delta2 = maxPwr2 - minPwr2;
  gap = minPwr2 - maxPwr1;
  Lb(1) = minPwr1;
  Ub(1) = maxPwr1;
  for i=2:order
    if mod(i,2)>0
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta1;
    else
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta2;
    end
  end
end
function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

The above script uses random search optimization by calling function sobolRandomSearch() and requests a million random searches. The above code is like the one in first random search optimization program. The main difference is that the above code uses functions that involve the Sobol quasi-random sequence. Running the above code produces the following Excel table:

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	
0.796215397	2.213924021	3.80804616	4.167893	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
1.001161421	-0.017916299	-0.257919168	0.069826481	-0.029381273
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.980927341	0.138170634	-0.457980428	0.113695746	-0.007357698
r_sqr1	r_sqr2			
0.999998426	0.999803041			

Table 7. Summary of the results appearing in file besselj_0_x_sobol_random_run1.xlsx.

The above table shows similar types of results as the ones in Table 1 and Table 2. Again, the adjusted coefficient of determination for the Quantum Shammas Polynomial is higher than that for the classical polynomial. Both are good values. Using the Sobol sequence gives surprisingly good results. I also suspect using one million iterations has something to do with it.

Here is the graph (from file besselj_0_x_sobol_random_run1.jpg) for the Bessel function and the two fitted polynomials:

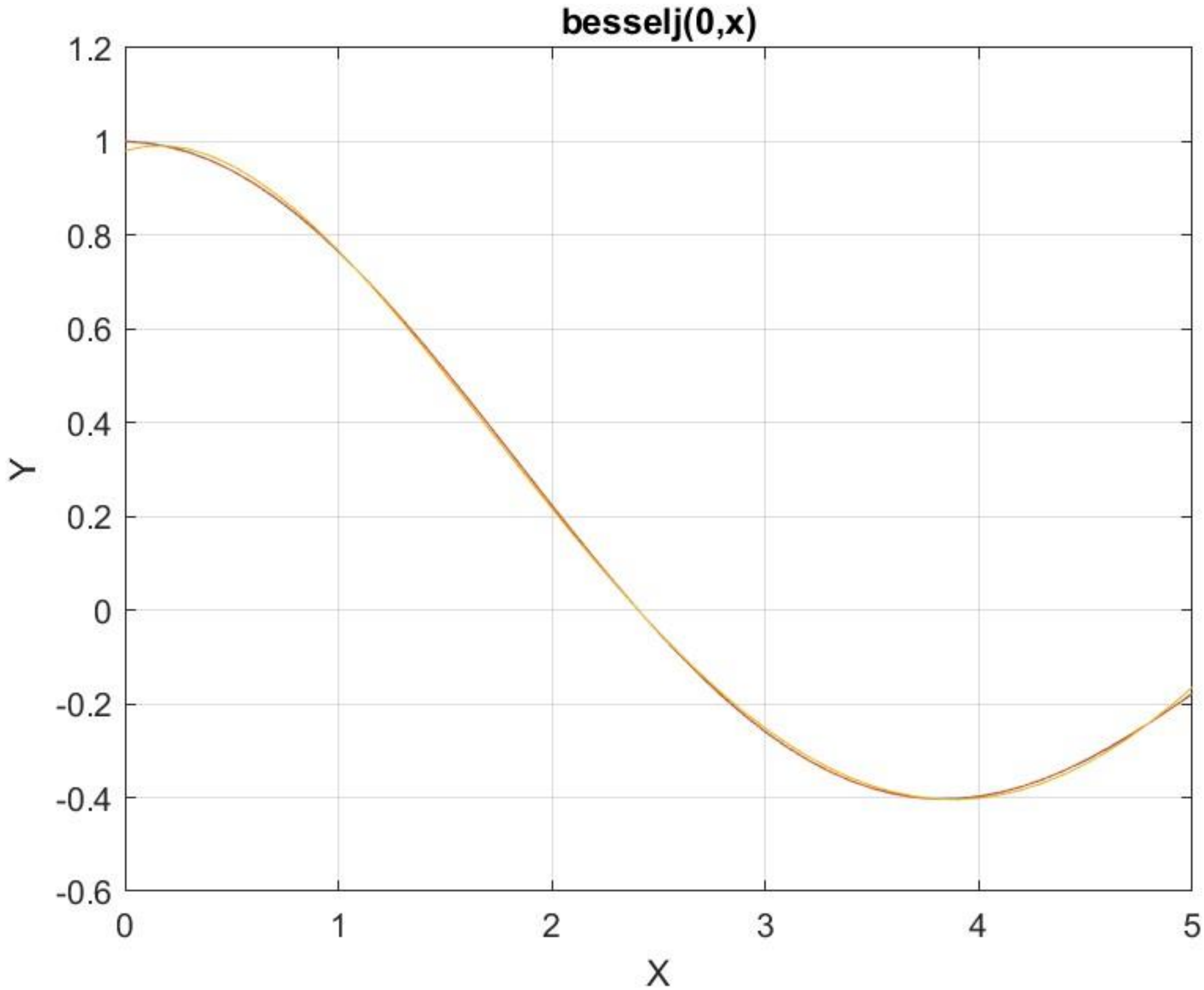

Figure 7. The graph from file besselj_0_x_sobol_random_run1.jpg.

The figure shows that both types of polynomials fit the Bessel function well.

Testing Bessel Function Fit with Sobol Random Search Optimization-Run2

The next MATLAB script (found in file testBessel1Sobo2.m) tests fitting Bessel J(0, x) for x in the range (0, 5) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial and a sixth order classical polynomial.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
```



```
zFilename = "besselj_0_x_sobol_random_run2";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "besselj(0,x)";
fprintf("%s\n",sEqn);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(order, 0.5, 1.5, 1.7, 2.3);

[bestX,bestFx] =
sobolRandomSearch(@quantShammasPoly,Lb,Ub,1000000);

SSE = quantShammasPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Polynomial Powers\n");
bestX
fprintf("Quantum Shammas Polynomial Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular polynomial fit\n");
c = polyfit(xData,yData,order)
yPoly = polyval(c,xData);
r = rsqr(yData,yPoly);
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r);

figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
```

```
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A1");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A4");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A7");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr1, maxPwr1, minPwr2,
maxPwr2)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  delta1 = maxPwr1 - minPwr1;
  delta2 = maxPwr2 - minPwr2;
  gap = minPwr2 - maxPwr1;
  Lb(1) = minPwr1;
  Ub(1) = maxPwr1;
  for i=2:order
    if mod(i,2)>0
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta1;
    else
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta2;
    end
  end
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

The above script uses random search optimization by calling function sobolRandomSearch() and requests a million random searches. The above code is very similar to the Halton version. The difference is in the filenames and the use of the Sobol-version of the random search optimization function. The above code generates the following Excel table.

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
1.581084402	2.504505821	3.762496947	4.63588809	5.953390774	6.74527333	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
0.984583706	0.004758179	-0.28940728	0.09570202	-0.019051264	0.000658717	-4.14767E-05
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.999707577	0.996718149					

Table 8. Summary of the results appearing in file besselj_0_x_sobol_random_run2.xlsx.

As expected, the adjusted coefficient of determination for the Quantum Shammas Polynomial is slightly higher than the one for classical polynomials.

Here is the graph (from file besselj_0_x_sobol_random_run2.jpg) for the Bessel function and the two fitted polynomials:

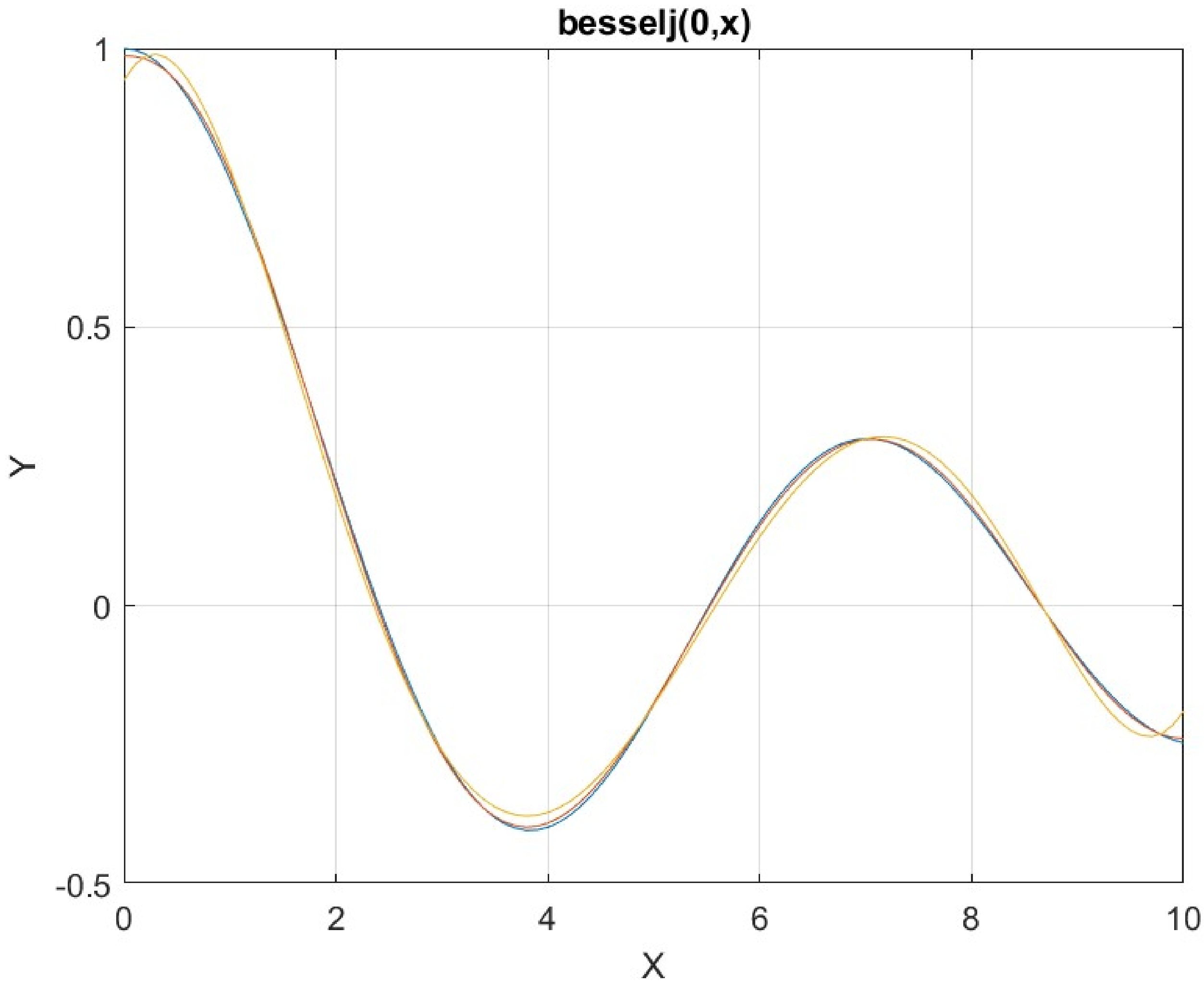

Figure 8. The graph from file besselj_0_x_sobol_random_run2.jpg

Again, the above curves show some deviations between the two types of fitted polynomials and the curve for the Bessel function.

Conclusion for Bessel Function Fitting

The results for the Bessel curve fitting show that all the applied methods yield better fittings than the classical polynomials.

The next four subsections look at the curve fitting of ln(x) with values of (x-1) in the range of (1, 7).

Testing ln(x) Function Fit with PSO

The next MATLAB script (found in file testLog1pso.m) tests fitting ln(x) vs (x-1) for x in the range (1, 7) and samples at 0.1 steps, and using the PSO method. The curve fits use a fourth order Quantum Shammas Polynomial and a fourth order classical polynomial.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "Ln_x";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "ln(x)";
fprintf(sEqn);
fprintf("x=1:0.1:7\n")
xData0= 1:0.1:7;
xData0 = xData0';
n = length(xData0);
yData = log(xData0);
xData = xData0 - 1;
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.5, 1.7, 2.3);

[bestX,bestFx] = psox(@quantShammasPoly,Lb,Ub,1000,5000,true);

SSE = quantShammasPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Polynomial Powers\n");
bestX
fprintf("Quantum Shammas Polynomial Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular polynomial fit\n");
c = polyfit(xData,yData,order)
yPoly = polyval(c,xData);
r = rsqr(yData,yPoly);
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r);
```

```
figure(1)
plot(xData0,yData,xData0,yCalc,xData0,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A1");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A4");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A7");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr1, maxPwr1, minPwr2,
maxPwr2)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  delta1 = maxPwr1 - minPwr1;
  delta2 = maxPwr2 - minPwr2;
  gap = minPwr2 - maxPwr1;
  Lb(1) = minPwr1;
  Ub(1) = maxPwr1;
  for i=2:order
    if mod(i,2)>0
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta1;
    else
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta2;
    end
  end
end

function r = rsqr(y,ycalc)
  n = length(y);
```

```
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

In the above code, each calls to function psox() performs a PSO search using a population size of 50 and 500 maximum iterations. The above code is very similar to the previous versions. The difference is in the filenames and the fitted function ln(x) vs (x-1). The above code generates the following Excel table.

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	
0.939289561	1.700234372	2.500929602	3.709431112	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
-0.002964267	0.929047739	-0.271126431	0.037786471	-0.000890851
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.019526836	0.850579688	-0.210226108	0.031956872	-0.001944825
r_sqr1	r_sqr2			
0.99999546	0.99989954			

Table 9. Summary of the results appearing in file Ln_x.xlsx.

The adjusted coefficient of determination for the Quantum Shammas Polynomial is higher than the one for classical polynomials.

Here is the graph (from file ln_x.jpg) for the ln(x) function and the two fitted polynomials:

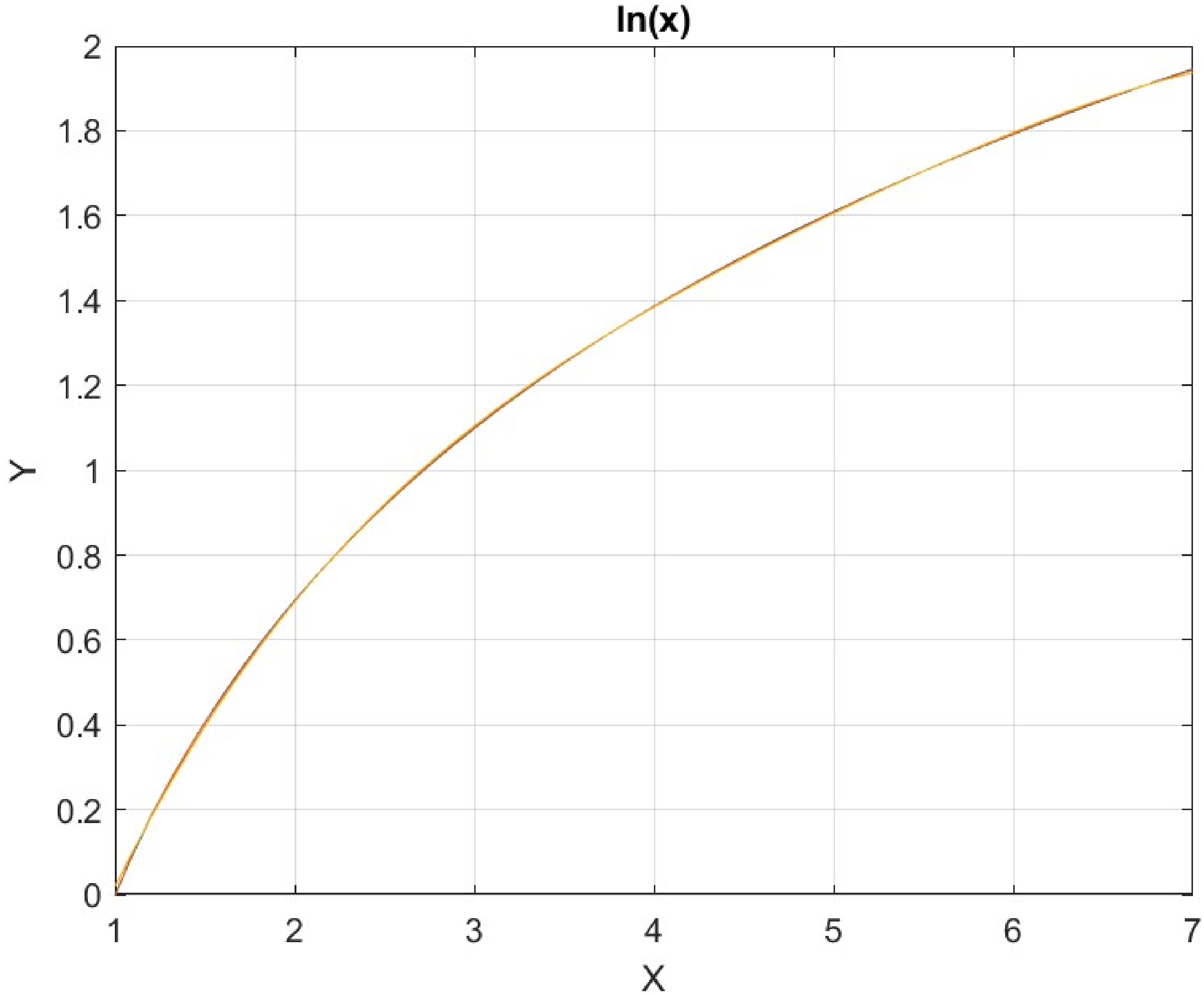

Figure 9. The graph from file ln_x.jpg

The above graph shows that the two types of polynomials fit the ln(x) function well.

Testing ln(x) Function Fit with Random Search Optimization

The next MATLAB script (found in file testLog1Random.m) tests fitting ln(x) vs (x-1) for x in the range (1, 7) and samples at 0.1 steps, and using the random search optimization. The curve fits use a fourth order Quantum Shammas Polynomial and a fourth order classical polynomial.

```
clc
clear
close all
```

```
global xData yData yCalc glbRsqr QSPcoeff
zFilename = "Ln_x_rand";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "ln(x)";
fprintf(sEqn);
fprintf("x=1:0.1:7\n")
xData0= 1:0.1:7;
xData0 = xData0';
n = length(xData0);
yData = log(xData0);
xData = xData0 - 1;
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.5, 1.7, 2.3);

[bestX,bestFx] = randomSearch(@quantShammasPoly,Lb,Ub,1000000);

SSE = quantShammasPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Polynomial Powers\n");
bestX
fprintf("Quantum Shammas Polynomial Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular polynomial fit\n");
c = polyfit(xData,yData,order)
yPoly = polyval(c,xData);
r = rsqr(yData,yPoly);
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData0,yData,xData0,yCalc,xData0,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
```

```
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A1");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A4");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A7");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr1, maxPwr1, minPwr2,
maxPwr2)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  delta1 = maxPwr1 - minPwr1;
  delta2 = maxPwr2 - minPwr2;
  gap = minPwr2 - maxPwr1;
  Lb(1) = minPwr1;
  Ub(1) = maxPwr1;
  for i=2:order
    if mod(i,2)>0
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta1;
    else
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta2;
    end
  end
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

The above script uses random search optimization by calling function randomSearch() and requests a million random searches. The above code is similar to ln_x,m except it uses different output filenames and calls the randomSearch() function for the curve fit optimization. The above code generates the following summary Excel table:

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	
1.004420082	1.53678095	2.271238281	3.347183885	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
-0.001660656	1.143567181	-0.512039957	0.064208762	-0.001713871
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.019526836	0.850579688	-0.210226108	0.031956872	-0.001944825
r_sqr1	r_sqr2			
0.999998347	0.99989954			

Table 10. Summary of the results appearing in file Ln_x_rand.xlsx.

The adjusted coefficient of determination for the Quantum Shammas Polynomial is higher than the one for classical polynomials. Interestingly, the adjusted coefficient of determination for the random search is also slightly higher than that of the PSO method!

Here is the graph (from file ln_x_rand.jpg) for the Bessel function and the two fitted polynomials:

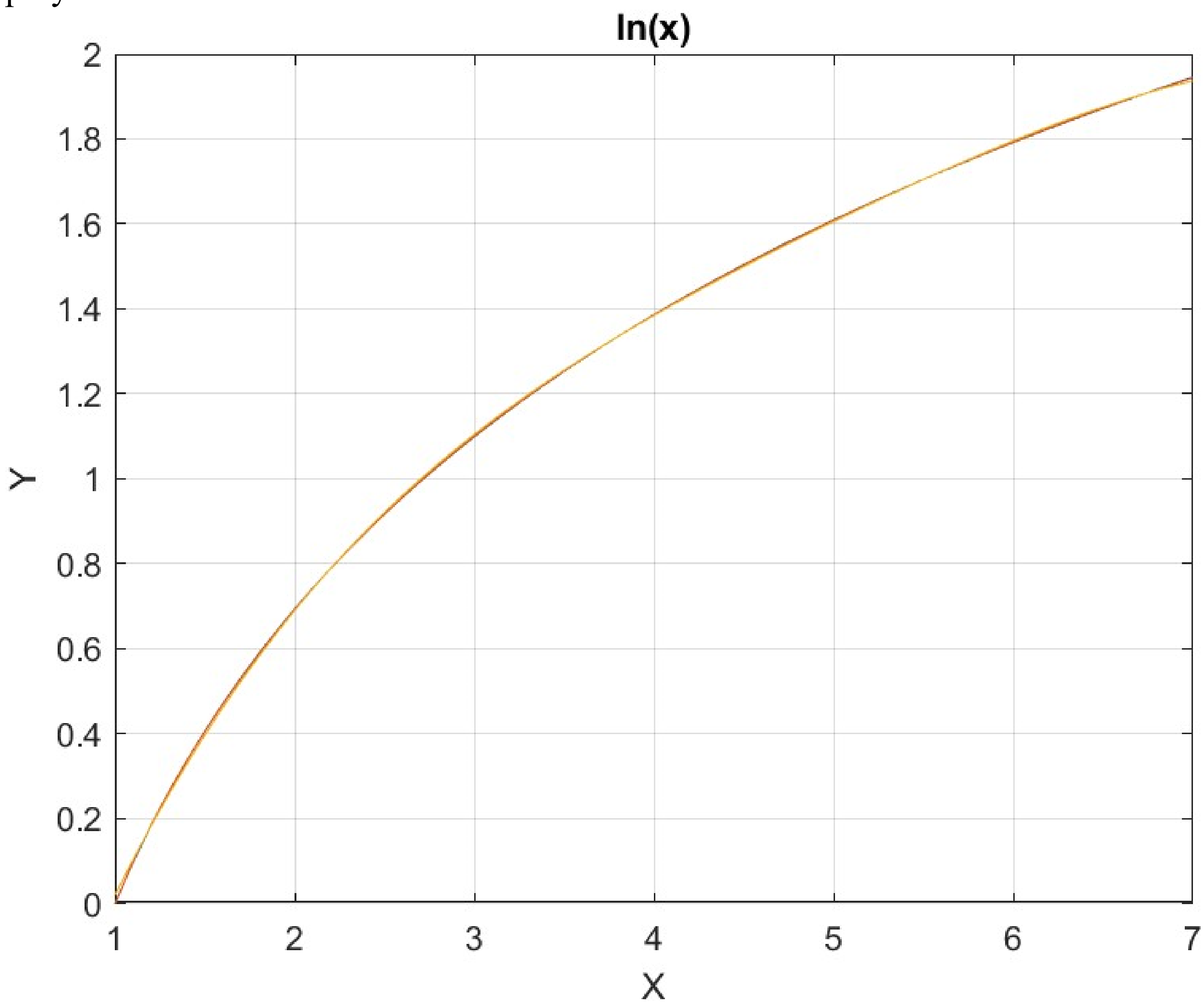

Figure 10. The graph from file ln_x_rand.jpg

The above graph shows that the two types of polynomials fit the ln(x) function well.

Testing ln(x) Function Fit with Halton Random Search Optimization

The next MATLAB script (found in file testLog1Halton.m) tests fitting ln(x) vs (x-1) for x in the range (1, 7) and samples at 0.1 steps, and using the Halton quasi-random search optimization. The curve fits use a fourth order Quantum Shammas Polynomial and a fourth order classical polynomial.

```
clc
clear
close all
```

```
global xData yData yCalc glbRsqr QSPcoeff
zFilename = "Ln_x_halton_rand";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "ln(x)";
fprintf(sEqn);
fprintf("x=1:0.1:7\n")
xData0= 1:0.1:7;
xData0 = xData0';
n = length(xData0);
yData = log(xData0);
xData = xData0 - 1;
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.5, 1.7, 2.3);

[bestX,bestFx] =
haltonRandomSearch(@quantShammasPoly,Lb,Ub,1000000);

SSE = quantShammasPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Polynomial Powers\n");
bestX
fprintf("Quantum Shammas Polynomial Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular polynomial fit\n");
c = polyfit(xData,yData,order)
yPoly = polyval(c,xData);
r = rsqr(yData,yPoly);
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData0,yData,xData0,yCalc,xData0,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);
```

```
QSPpwr = bestX;
Coeff = flip(c);
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A1");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A4");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A7");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr1, maxPwr1, minPwr2,
maxPwr2)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  delta1 = maxPwr1 - minPwr1;
  delta2 = maxPwr2 - minPwr2;
  gap = minPwr2 - maxPwr1;
  Lb(1) = minPwr1;
  Ub(1) = maxPwr1;
  for i=2:order
    if mod(i,2)>0
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta1;
    else
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta2;
    end
  end
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

The above script uses random search optimization by calling function haltonRandomSearch() and requests a million random searches. The above file generates the following Excel table summary.

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	
1.009828148	1.545275769	2.262748001	3.34245549	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
-0.001021003	1.150679812	-0.525074732	0.069667134	-0.001824848
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.019526836	0.850579688	-0.210226108	0.031956872	-0.001944825
r_sqr1	r_sqr2			
0.99999833	0.99989954			

Table 11. Summary of the results appearing in file Ln_x_halton_rand.xlsx.

The adjusted coefficient of determination for the Quantum Shammas Polynomial is higher than the one for classical polynomials. Interestingly, the adjusted coefficient of determination for the random search is also slightly higher than that of the PSO method! This is a bit surprinting, given that the Halton sequence is a quasi-random sequence!

Here is the graph (from file ln_x_halton_rand.jpg) for the Bessel function and the two fitted polynomials:

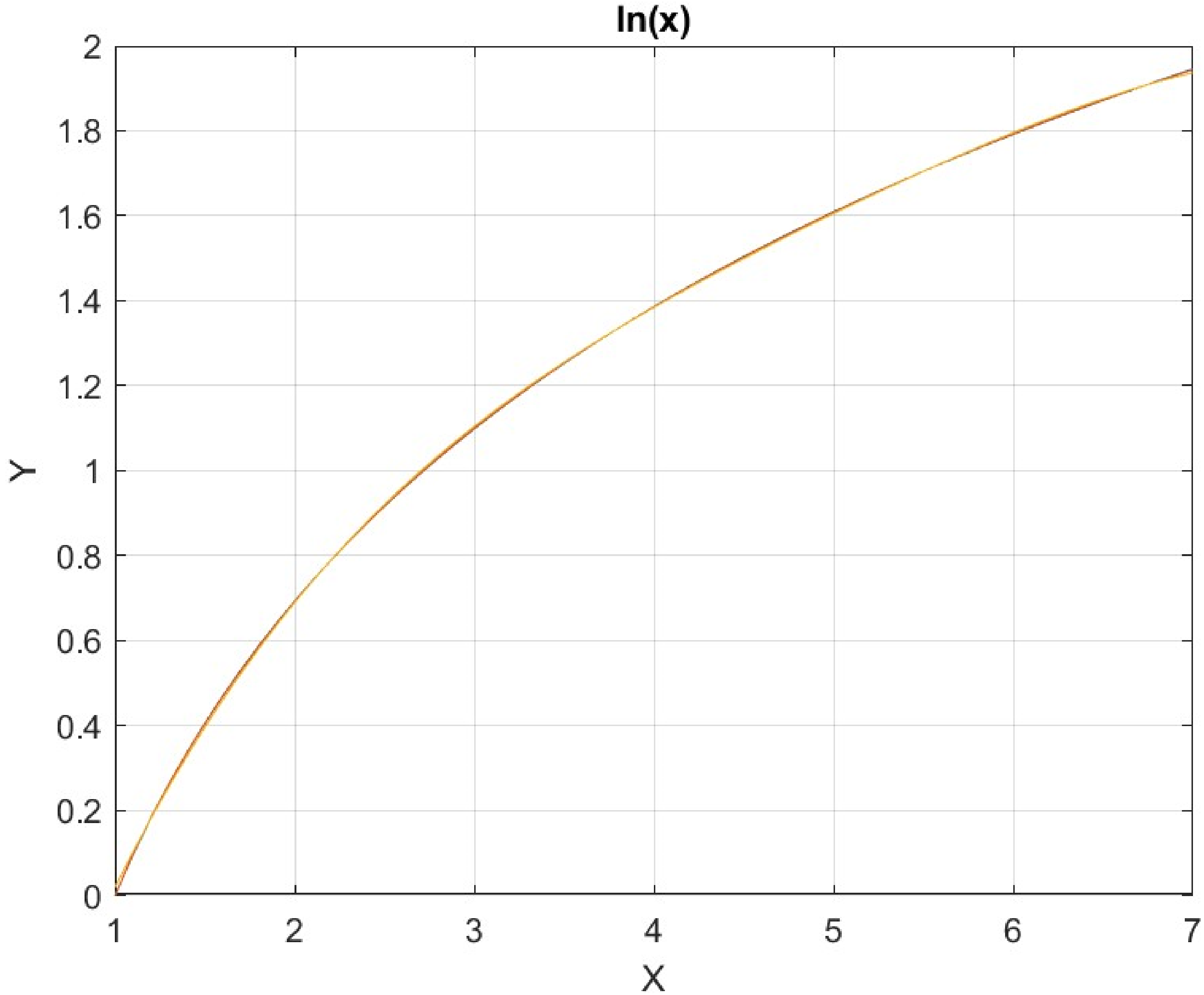

Figure 11. The graph from file ln_x_halton_rand.jpg

The above graph shows that the two types of polynomials fit the ln(x) function well.

Testing ln(x) Function Fit with Sobol Random Search Optimization

The next MATLAB script (found in file testLog1Sobol.m) tests fitting ln(x) vs (x-1) for x in the range (1, 7) and samples at 0.1 steps, and using the Sobol quasi-random search optimization. The curve fits use a fourth order Quantum Shammas Polynomial and a fourth order classical polynomial..

```
clc
clear
close all
```

```
global xData yData yCalc glbRsqr QSPcoeff
zFilename = "Ln_x_sobol_rand";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "ln(x)";
fprintf(sEqn);
fprintf("x=1:0.1:7\n")
xData0= 1:0.1:7;
xData0 = xData0';
n = length(xData0);
yData = log(xData0);
xData = xData0 - 1;
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.5, 1.7, 2.3);

[bestX,bestFx] =
sobolRandomSearch(@quantShammasPoly,Lb,Ub,1000000);

SSE = quantShammasPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Polynomial Powers\n");
bestX
fprintf("Quantum Shammas Polynomial Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular polynomial fit\n");
c = polyfit(xData,yData,order)
yPoly = polyval(c,xData);
r = rsqr(yData,yPoly);
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r);

figure(1)
plot(xData0,yData,xData0,yCalc,xData0,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);
```

```
QSPpwr = bestX;
Coeff = flip(c);
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A1");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A4");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A7");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr1, maxPwr1, minPwr2, 
maxPwr2)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  delta1 = maxPwr1 - minPwr1;
  delta2 = maxPwr2 - minPwr2;
  gap = minPwr2 - maxPwr1;
  Lb(1) = minPwr1;
  Ub(1) = maxPwr1;
  for i=2:order
    if mod(i,2)>0
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta1;
    else
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta2;
    end
  end
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

The above script uses random search optimization by calling function sobolRandomSearch() and requests a million random searches. The above file generates the following Excel table summary.

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	
1.006807557	1.53255124	2.280341574	3.399303283	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
-0.001362983	1.149561517	-0.515853033	0.061519146	-0.001492128
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.019526836	0.850579688	-0.210226108	0.031956872	-0.001944825
r_sqr1	r_sqr2			
0.99999827	0.99989954			

Table 12. Summary of the results appearing in file Ln_x_soboln_rand.xlsx.

The adjusted coefficient of determination for the Quantum Shammas Polynomial is higher than the one for classical polynomials. Interestingly, the adjusted coefficient of determination for the random search is also slightly higher than that of the PSO method! This is a bit surprinting, given that the Sobol sequence is a quasi-random sequence!

Here is the graph (from file ln_x_sobol_rand.jpg) for the Bessel function and the two fitted polynomials:

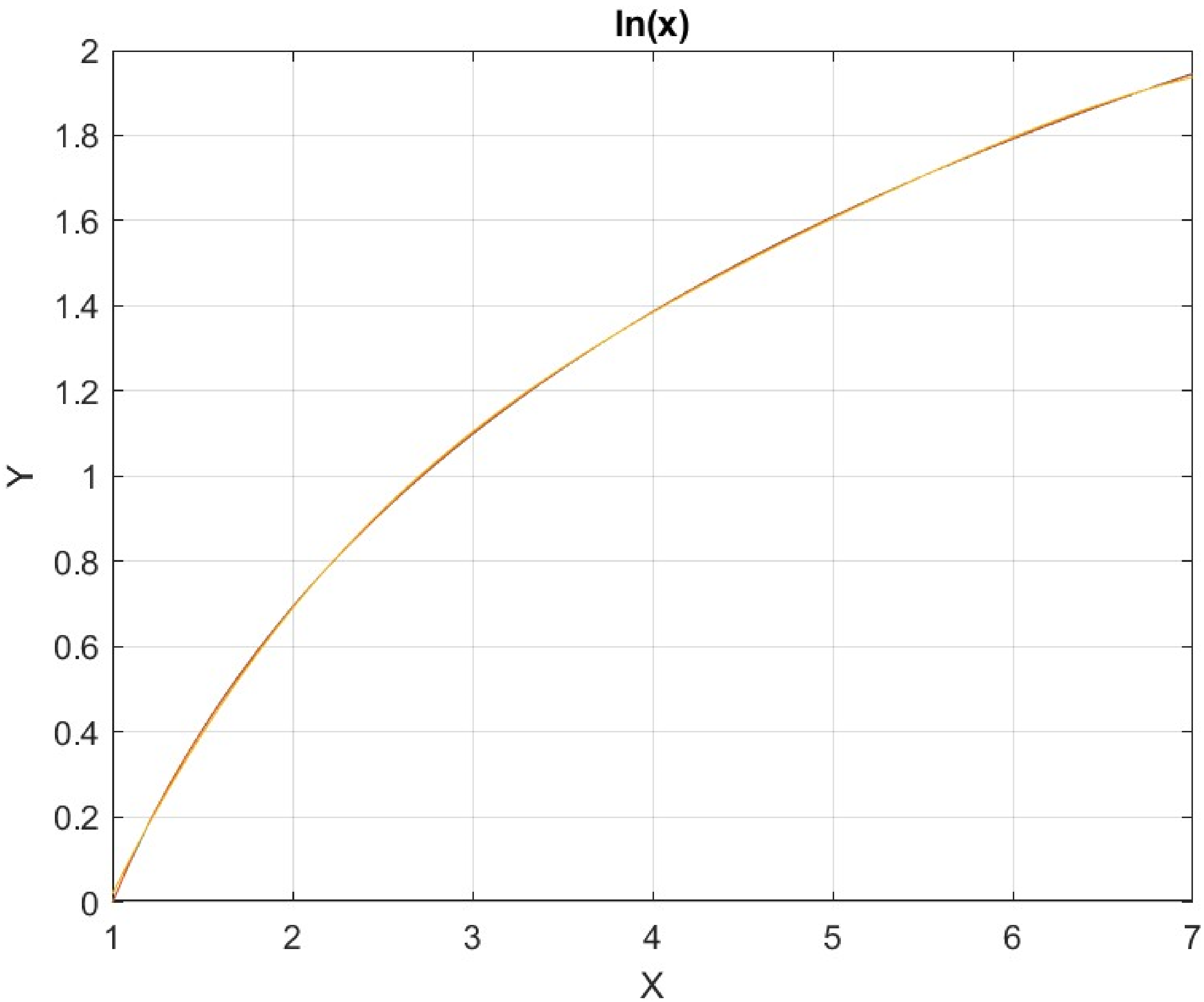

Figure 12. The graph from file ln_x_sobol_rand.jpg

The above graph shows that the two types of polynomials fit the ln(x) function well.

Conclusion for fitting the ln(x) Function

The above four subsections show that fitting the ln(x) vs (x-1) for the range of (1, 7) using the Quantum Shammas Polynomial is a success. These polynomials yield adjusted coefficients of determination that are higher than the corresponding classical polynomials.

The next four subsections in Part 1 look at fitting the right side of the standard Gaussian bell, where x>= 0. To calculate values for x<0, use the symmetry of y(x) = y(-x).

Testing the Right-Side Gauss-Bell Function Fit with PSO

The next MATLAB script (found in file testGauss1pso.m) tests fitting normal N(0, 1) for x in the range (0, 3) and samples at 0.1 steps, and using the PSO method. The curve fits use a fourth order Quantum Shammas Polynomial and a fourth order classical polynomial.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "Right_GaussBell_x";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "exp(-x^2/2)/sqrt(2*pi)";
fprintf(sEqn);
fprintf("x=0:0.1:3\n")
xData= 0:0.1:3;
xData = xData';
n = length(xData);
yData = exp(-xData.^2/2)/sqrt(2*pi);
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.5, 1.7, 2.3);

[bestX,bestFx] = psox(@quantShammasPoly,Lb,Ub,1000,5000,true);

SSE = quantShammasPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Polynomial Powers\n");
bestX
fprintf("Quantum Shammas Polynomial Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular polynomial fit\n");
c = polyfit(xData,yData,order)
yPoly = polyval(c,xData);
```

```
r = rsqr(yData,yPoly);
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A1");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A4");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A7");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr1, maxPwr1, minPwr2,
maxPwr2)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  delta1 = maxPwr1 - minPwr1;
  delta2 = maxPwr2 - minPwr2;
  gap = minPwr2 - maxPwr1;
  Lb(1) = minPwr1;
  Ub(1) = maxPwr1;
  for i=2:order
    if mod(i,2)>0
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta1;
    else
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta2;
    end
  end
end
```

```
function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

In the above code, each calls to function psox() performs a PSO search using a population size of 50 and 500 maximum iterations. The above code is very similar to the previous versions. The difference is in the filenames and the fitted normal Gaussian function. The above code generates the following Excel table.

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	
1.499340193	2.299416753	2.69245979	3.700593842	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.39810409	0.044258867	-0.693425929	0.529710131	-0.036961318
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.397644494	0.028633101	-0.306018517	0.139989216	-0.018592075
r_sqr1	r_sqr2			
0.999978519	0.999967249			

Table 13. Summary of the results appearing in file Right_GaussBell_x.xlsx.

The adjusted coefficient of determination for the Quantum Shammas Polynomial is higher (by a proverbial hair) than the one for classical polynomials. Since the PSO method uses random numbers, I consider the difference between the two results as statistically insignificant.

Here is the graph (from file Right_GaussBell_x.jpg) for the right normal Gauss function and the two fitted polynomials:

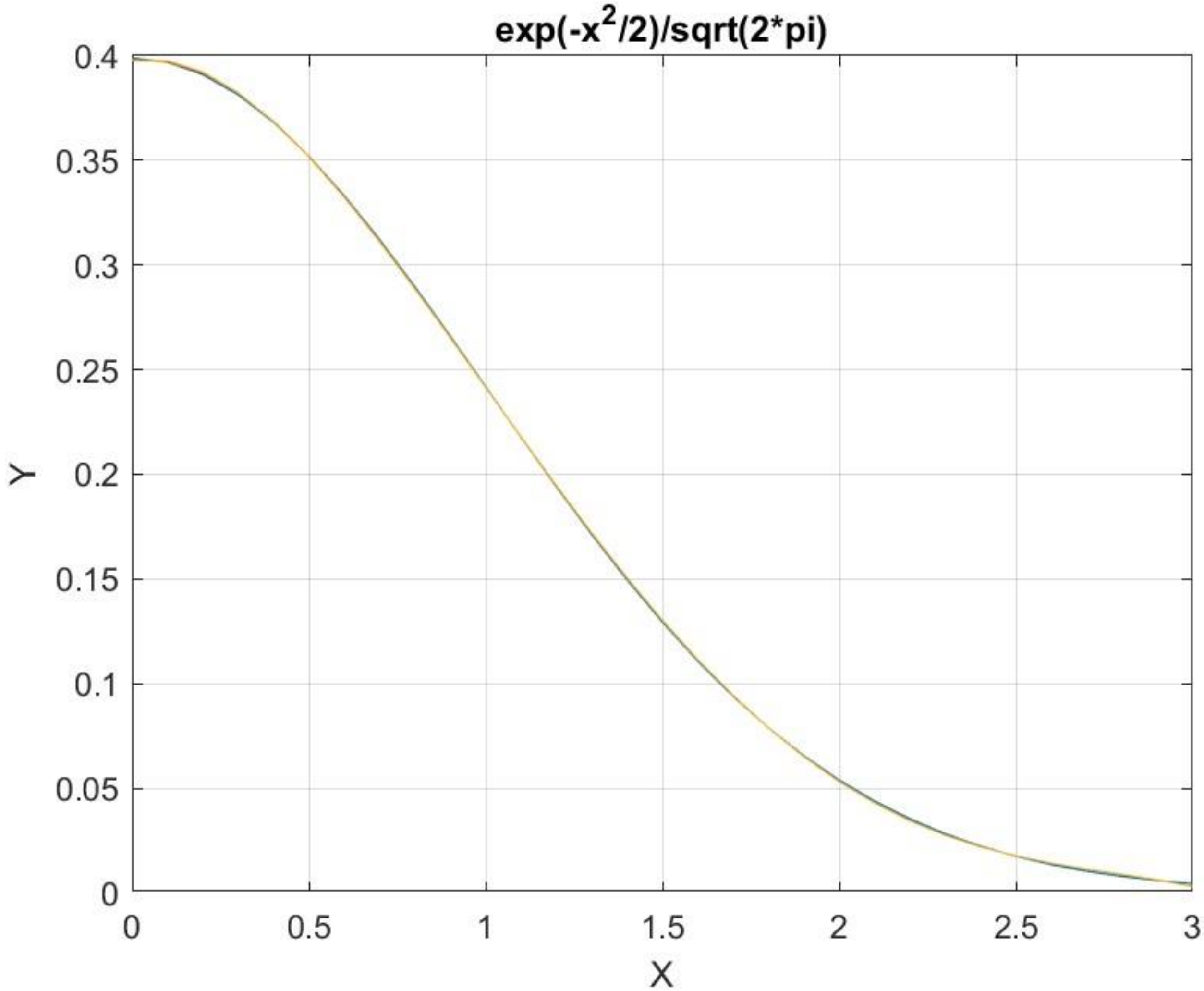

Figure 13. The graph from file Right_GaussBell_x.jpg.

The above graph shows that the two types of polynomials fit the right normal Gauss function well.

Testing the Right-Side Gauss-Bell Function Fit with Random Search Optimization

The next MATLAB script (found in file testGauss1Random.m) tests fitting normal N(0, 1) for x in the range (0, 3) and samples at 0.1 steps, and using the random search optimization. The curve fits use a fourth order Quantum Shammas Polynomial and a fourth order classical polynomial.

```
clc
```

```
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "Right_GaussBell_x_random";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "exp(-x^2/2)/sqrt(2*pi)";
fprintf(sEqn);
fprintf("x=0:0.1:3\n")
xData= 0:0.1:3;
xData = xData';
n = length(xData);
yData = exp(-xData.^2/2)/sqrt(2*pi);
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.5, 1.7, 2.3);

[bestX,bestFx] = randomSearch(@quantShammasPoly,Lb,Ub,1000000);

SSE = quantShammasPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Polynomial Powers\n");
bestX
fprintf("Quantum Shammas Polynomial Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular polynomial fit\n");
c = polyfit(xData,yData,order)
yPoly = polyval(c,xData);
r = rsqr(yData,yPoly);
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);
```

```
QSPpwr = bestX;
Coeff = flip(c);
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A1");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A4");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A7");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr1, maxPwr1, minPwr2,
maxPwr2)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  delta1 = maxPwr1 - minPwr1;
  delta2 = maxPwr2 - minPwr2;
  gap = minPwr2 - maxPwr1;
  Lb(1) = minPwr1;
  Ub(1) = maxPwr1;
  for i=2:order
    if mod(i,2)>0
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta1;
    else
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta2;
    end
  end
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

The above script uses random search optimization by calling function randomSearch() and requests a million random searches. The above code generates the following summary Excel table:

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	
1.632767848	2.517817764	2.693684321	3.335701501	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.398161785	0.03480119	-1.813454929	1.776987199	-0.154834019
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.397644494	0.028633101	-0.306018517	0.139989216	-0.018592075
r_sqr1	r_sqr2			
0.999981566	0.999967249			

Table 14. Summary of the results appearing in file Right_GaussBell_x_random.xlsx.

The adjusted coefficient of determination for the Quantum Shammas Polynomial is higher (by a proverbial hair) than the one for classical polynomials. Since the random search method uses random numbers, I consider the difference between the two results as statistically insignificant.

Here is the graph (from file Right_GaussBell_x _random.jpg) for the right normal Gauss function and the two fitted polynomials:

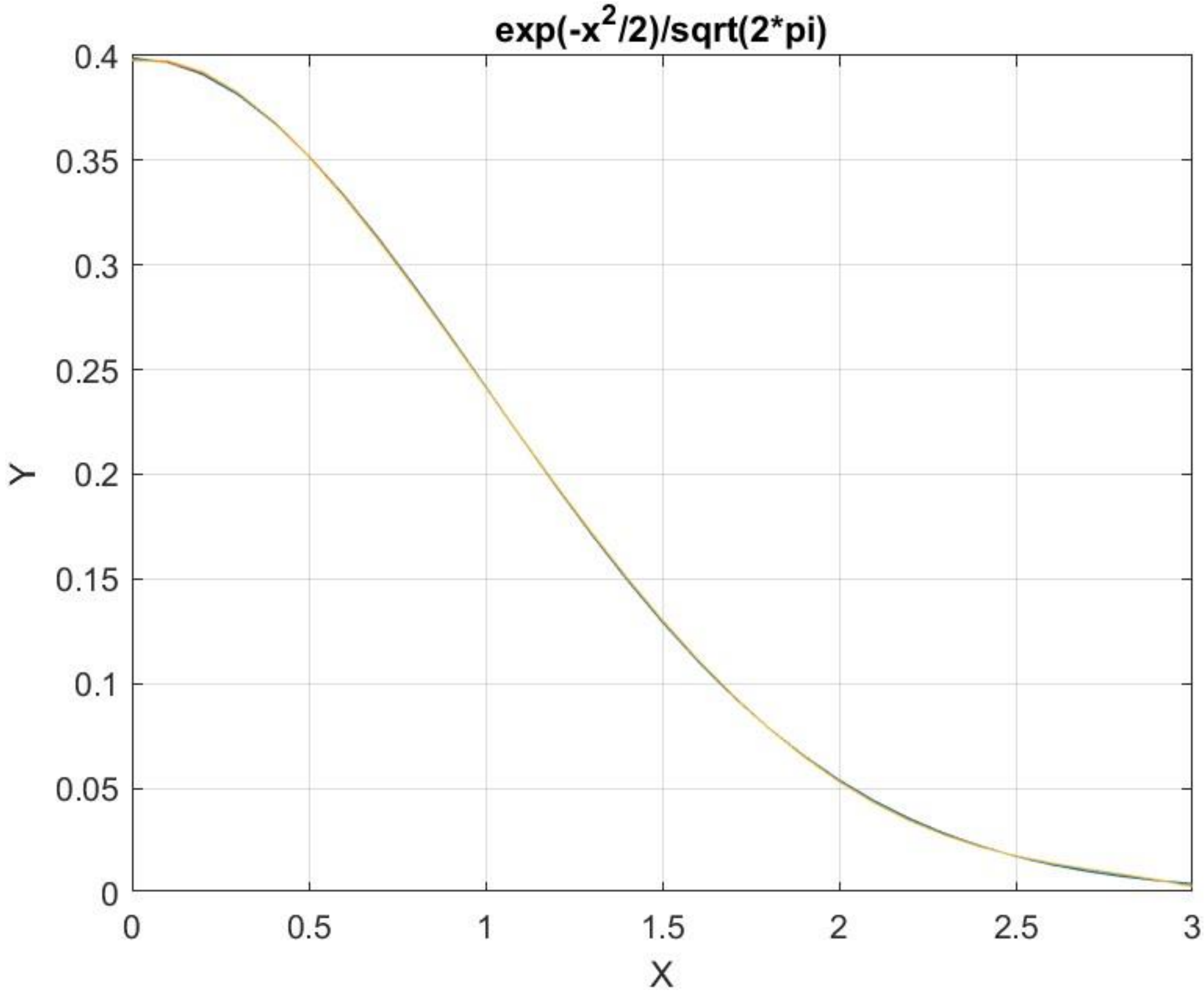

Figure 14. The graph from file Right_GaussBell_x_random.jpg.

The above graph shows that the two types of polynomials fit the right normal Gauss function well.

Testing the Right-Side Gauss-Bell Function Fit with Halton Random Search Optimization

The next MATLAB script (found in file testGauss1Halton.m) tests fitting normal N(0, 1) for x in the range (0, 3) and samples at 0.1 steps, and using the Halton quasi-random search optimization. The curve fits use a fourth order Quantum Shammas Polynomial and a fourth order classical polynomial.

```
clc
```

```
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "Right_GaussBell_x_halton_random";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "exp(-x^2/2)/sqrt(2*pi)";
fprintf(sEqn);
fprintf("x=0:0.1:3\n")
xData= 0:0.1:3;
xData = xData';
n = length(xData);
yData = exp(-xData.^2/2)/sqrt(2*pi);
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.5, 1.7, 2.3);

[bestX,bestFx] =
haltonRandomSearch(@quantShammasPoly,Lb,Ub,1000000);

SSE = quantShammasPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Polynomial Powers\n");
bestX
fprintf("Quantum Shammas Polynomial Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular polynomial fit\n");
c = polyfit(xData,yData,order)
yPoly = polyval(c,xData);
r = rsqr(yData,yPoly);
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);
```

```
QSPpwr = bestX;
Coeff = flip(c);
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A1");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A4");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A7");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr1, maxPwr1, minPwr2,
maxPwr2)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  delta1 = maxPwr1 - minPwr1;
  delta2 = maxPwr2 - minPwr2;
  gap = minPwr2 - maxPwr1;
  Lb(1) = minPwr1;
  Ub(1) = maxPwr1;
  for i=2:order
    if mod(i,2)>0
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta1;
    else
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta2;
    end
  end
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```


The above script uses random search optimization by calling function haltonRandomSearch() and requests a million random searches. The above code generates the following summary Excel table:

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	
1.636774847	2.526358417	2.690384538	3.335998859	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.398177902	0.033590214	-1.938398451	1.903880975	-0.155583034
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.397644494	0.028633101	-0.306018517	0.139989216	-0.018592075
r_sqr1	r_sqr2			
0.999981583	0.999967249			

Table 15. Summary of the results appearing in file Right_GaussBell_x_halton_random.xlsx.

The adjusted coefficient of determination for the Quantum Shammas Polynomial is higher (by a proverbial hair) than the one for classical polynomials. I consider the difference between the two results as statistically insignificant.

Here is the graph (from file Right_GaussBell_x_halton_random.jpg) for the right normal Gauss function and the two fitted polynomials:

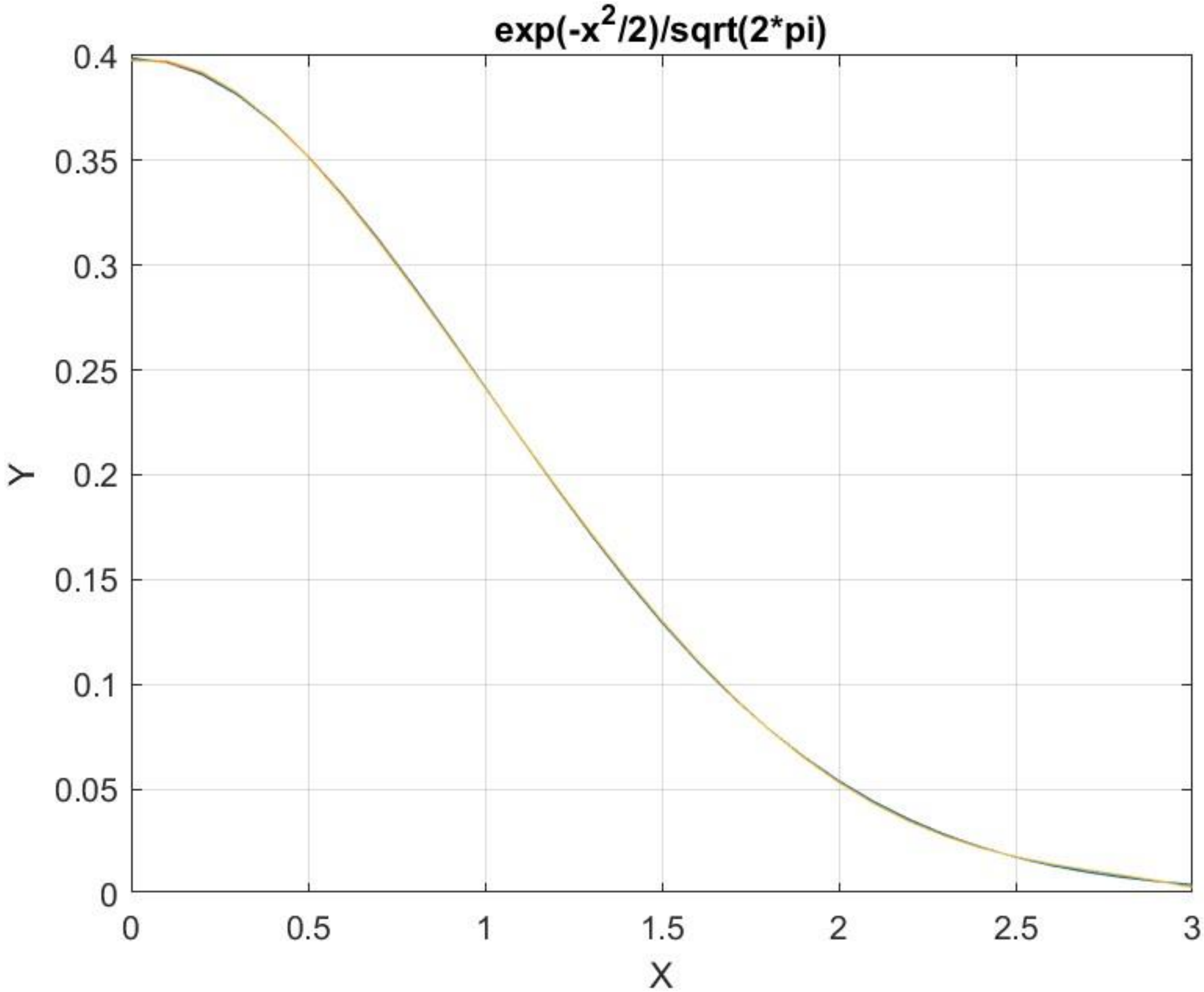

Figure 15. The graph from file Right_GaussBell_x_halton_random.jpg.

The above graph shows that the two types of polynomials fit the right normal Gauss function well.

Testing the Right-Side Gauss-Bell Function Fit with Sobol Random Search Optimization

The next MATLAB script (found in file testGauss1Sobol.m) tests fitting normal N(0, 1) for x in the range (0, 3) and samples at 0.1 steps, and using the Sobol quasi-random search optimization. The curve fits use a fourth order Quantum Shammas Polynomial and a fourth order classical polynomial.

```
clc
clear
```

```
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "Right_GaussBell_x_sobol_random";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "exp(-x^2/2)/sqrt(2*pi)";
fprintf(sEqn);
fprintf("x=0:0.1:3\n")
xData= 0:0.1:3;
xData = xData';
n = length(xData);
yData = exp(-xData.^2/2)/sqrt(2*pi);
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.5, 1.7, 2.3);

[bestX,bestFx] =
sobolRandomSearch(@quantShammasPoly,Lb,Ub,1000000);

SSE = quantShammasPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Polynomial Powers\n");
bestX
fprintf("Quantum Shammas Polynomial Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular polynomial fit\n");
c = polyfit(xData,yData,order)
yPoly = polyval(c,xData);
r = rsqr(yData,yPoly);
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);
```

```
QSPpwr = bestX;
Coeff = flip(c);
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A1");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A4");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A7");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr1, maxPwr1, minPwr2,
maxPwr2)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  delta1 = maxPwr1 - minPwr1;
  delta2 = maxPwr2 - minPwr2;
  gap = minPwr2 - maxPwr1;
  Lb(1) = minPwr1;
  Ub(1) = maxPwr1;
  for i=2:order
    if mod(i,2)>0
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta1;
    else
      Lb(i) = Ub(i-1) + gap;
      Ub(i) = Lb(i) + delta2;
    end
  end
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

The above script uses random search optimization by calling function sobolRandomSearch() and requests a million random searches. The above code generates the following summary Excel table:

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	
1.643997697	2.524161076	2.668369565	3.345187112	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.398140007	0.038520119	-2.190418893	2.139941619	-0.144530318
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.397644494	0.028633101	-0.306018517	0.139989216	-0.018592075
r_sqr1	r_sqr2			
0.999981547	0.999967249			

Table 16. Summary of the results appearing in file Right_GaussBell_x_soboln_random.xlsx.

The adjusted coefficient of determination for the Quantum Shammas Polynomial is higher (by a proverbial hair) than the one for classical polynomials. I consider the difference between the two results as statistically insignificant.

Here is the graph (from file Right_GaussBell_x_sobol_random.jpg) for the right normal Gauss function and the two fitted polynomials:

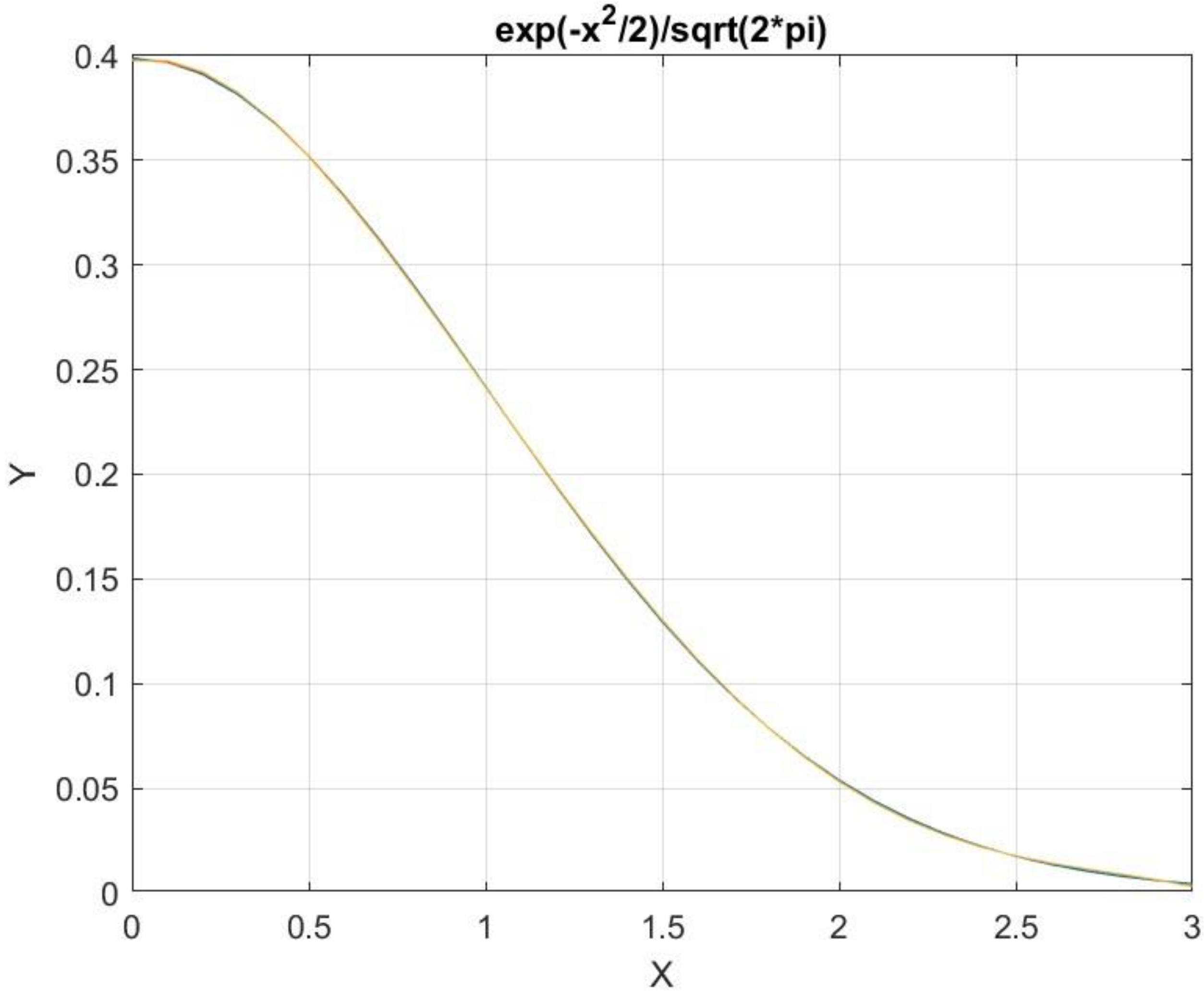

Figure 16. The graph from file Right_GaussBell_x_sobol_random.jpg.

The above graph shows that the two types of polynomials fit the right normal Gauss function well.

Conclusion for fitting the Right-Side Normal Gaussian Function

The above four subsections show that fitting the right-side normal Gaussian function in the range of (0, 3) using the Quantum Shammas Polynomial is a success. These polynomials yield adjusted coefficients of determination that are slightly higher than the corresponding classical polynomials.

Conclusion for Part 1D

The Quantum Shammas Polynomials (with its special power range pattern) did well in fitting the sample test cases. One should keep in mind that these polynomials (as well as the classical ones) may not always perform well for every single math function and for any/all ranges—that would be a very tall order! The results so far are encouraging.

Next is Part 1E

Part 1E of this study looks at the Optimum Quantum Shammas Polynomials that .

Document History

Date	***Version***	***Comments***
6/15/2023	1.0.0	Initial release.