Quantum Shammas Polynomials Part 1C

By

Namir Shammas

Contents

Introduction

Part 1 introduced you to Quantum Shammas Polynomials. Part 1B introduced Quantum Shammas Polynomials with wider power ranges than for the ones in Part 1. In this part we look at the second variant of these polynomials. The variation uses *narrower* ranges of random powers than those in Part 1. Random values, drawn from these ranges, are selected to yield the least square-errors models for Quantum Shammas Polynomials. The general 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 study we have, $0.5 <= r_1 <= 0.9$, $1.0 <= r_2 <= 1.4$, …, and $n*0.5 <= r_n < 0.9+(n-1)*0.5$. Notice that the upper value of a random power is 0.1 less than the lower value of its successor. This gap ensures that no two random powers have the same exact value. These relatively narrow ranges 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 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 (<1). 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 subsections 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 as well as multiple-half-power classical polynomials (i.e. with power of 0.5, 1, 1.5, 2, 2.5, and so on). The adjusted coefficient of determinations are good indicators of how he 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 Polynomials, a fourth order classical polynomial, and a fourth order multiple-half-power 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, 0.9, 0.1);

[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);
fprintf("\nMultiple-half-power  polynomial fit\n");
c2 = polyfit(sqrt(xData),yData,order)
yPoly2 = polyval(c2,sqrt(xData));
r2 = rsqr(yData,yPoly2);
% calculate adjusted value of the coefficient of determination
r2 = 1 - (1 - r2)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r2);

figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly,xData,yPoly2);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
HalfCoeff = flip(c2);
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");
T4 = array2table(HalfCoeff);
```

```
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
r_sqr = [glbRsqr r r2];
T5 = array2table(r_sqr);
writetable(T5,xlFile,"Sheet","Sheet1","Range","A13");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr, diffPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = Ub(i-1)+diffPwr;
    Ub(i) = Lb(i) + maxPwr;
  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	
0.894946875	1.899347914	2.898896383	3.896754167	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.974235737	0.178553538	-0.49445042	0.114456798	-0.006675031
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.980927341	0.138170634	-0.457980428	0.113695746	-0.007357698

HalfCoeff1	HalfCoeff2	HalfCoeff3	HalfCoeff4	HalfCoeff5
1.007829532	-0.516126123	1.902850569	-2.256654906	0.628649655
r_sqr1	r_sqr2	r_sqr3		
0.999653281	0.999803041	0.999393195		

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. Notice that the second to the fifth powers have a common fractional part that begins with 0.39. The fifth row shows the intercept (below QSPcoeff1) and to its right the coefficients for the various Quantum Shammas Polynomial. The eighth row shows the intercept and coefficients for the classical polynomial. The eleventh row shows the intercept and coefficients for the multiple-half-power 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 multiple-half-power classical polynomial. The cell under r_sqr3 shows the adjusted coefficient of determination for the fitted multiple-half-power classical polynomial. The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is less than the one for the classical polynomial, but higher than the multiple-half-power classical polynomial.

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

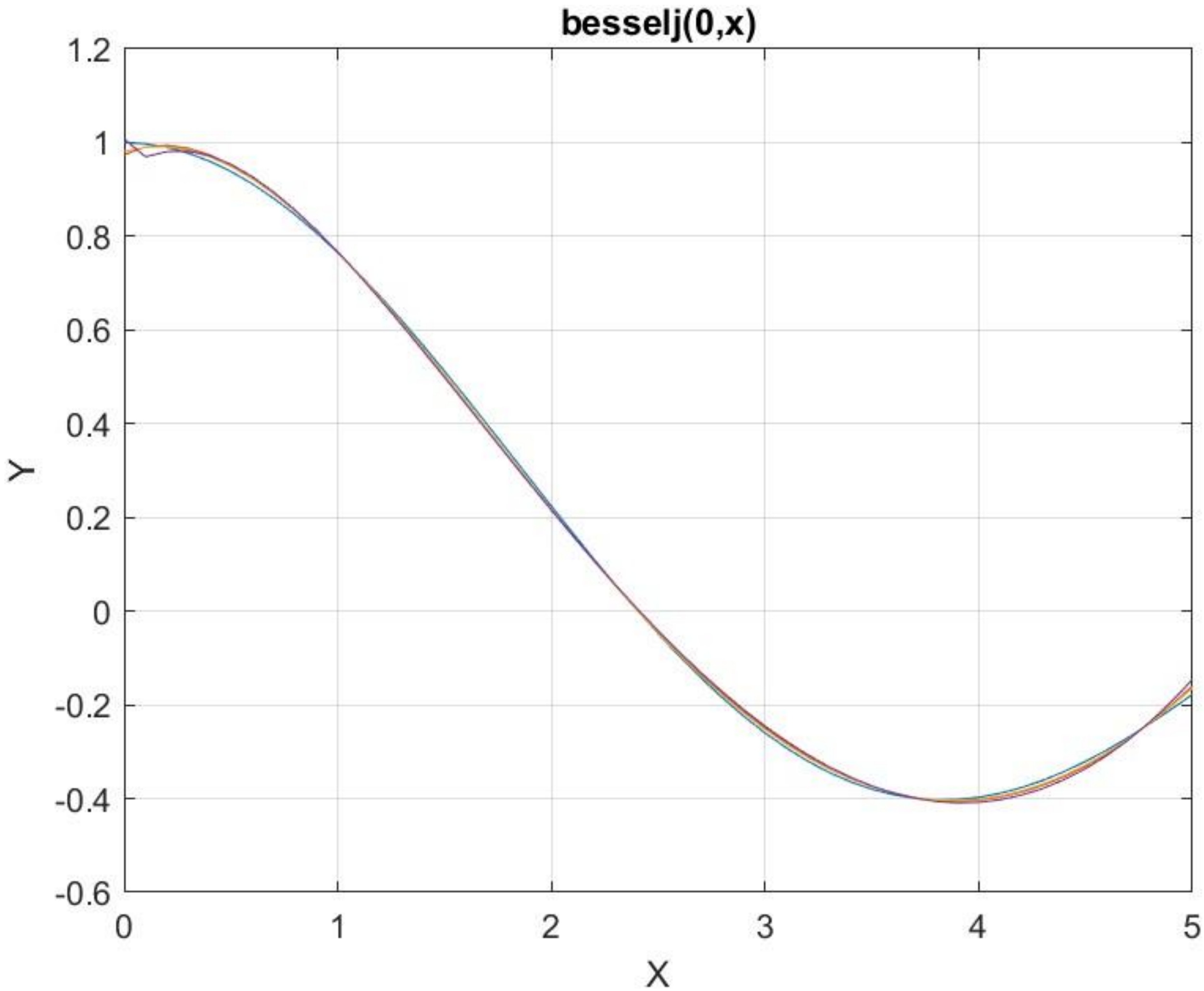

Figure 1. The graph from file besselj_0_x_run1.jpg.

The above graph shows a fairly acceptable fit for all polynomials.

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 Polynomials, a sixth order classical polynomial, and a sixth order multiple-half-power 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, 0.9, 0.1);

[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);
fprintf("\nMultiple-half-power  polynomial fit\n");
c2 = polyfit(sqrt(xData),yData,order)
yPoly2 = polyval(c2,sqrt(xData));
r2 = rsqr(yData,yPoly2);
% calculate adjusted value of the coefficient of determination
r2 = 1 - (1 - r2)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r2);

figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly,xData,yPoly2);
title(sEqn)
```

```
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
HalfCoeff = flip(c2);
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");
T4 = array2table(HalfCoeff);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
r_sqr = [glbRsqr r r2];
T5 = array2table(r_sqr);
writetable(T5,xlFile,"Sheet","Sheet1","Range","A13");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr, diffPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = Ub(i-1)+diffPwr;
    Ub(i) = Lb(i) + maxPwr;
  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	
0.830212731	1.203662296	2.025147845	3.004348865	4.004122713	5.001415585	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.013564998	-0.714290491	1.292552049	-1.069089103	0.285281824	-0.030366974	0.001142974
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
HalfCoeff1	HalfCoeff2	HalfCoeff3	HalfCoeff4	HalfCoeff5	HalfCoeff6	HalfCoeff7
1.074934616	-3.913080473	14.34688996	-18.70343239	10.34218791	-2.548257178	0.230499745
r_sqr1	r_sqr2	r_sqr3				
0.997798241	0.996718149	0.964226335				

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 various Quantum Shammas Polynomial. The eighth row shows the intercept and coefficients for the classical polynomial. The eleventh row shows the intercept and coefficients for the multiple-half-power 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 cell under r_sqr3 shows the adjusted coefficient of determination for the fitted multiple-half-power classical polynomial. The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is slightly higher than the one for the classical polynomial. The adjusted coefficient of determination for the classical polynomial is higher than the one for the multiple-half-power 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_run2.jpg) for the Bessel function and the two fitted polynomials:

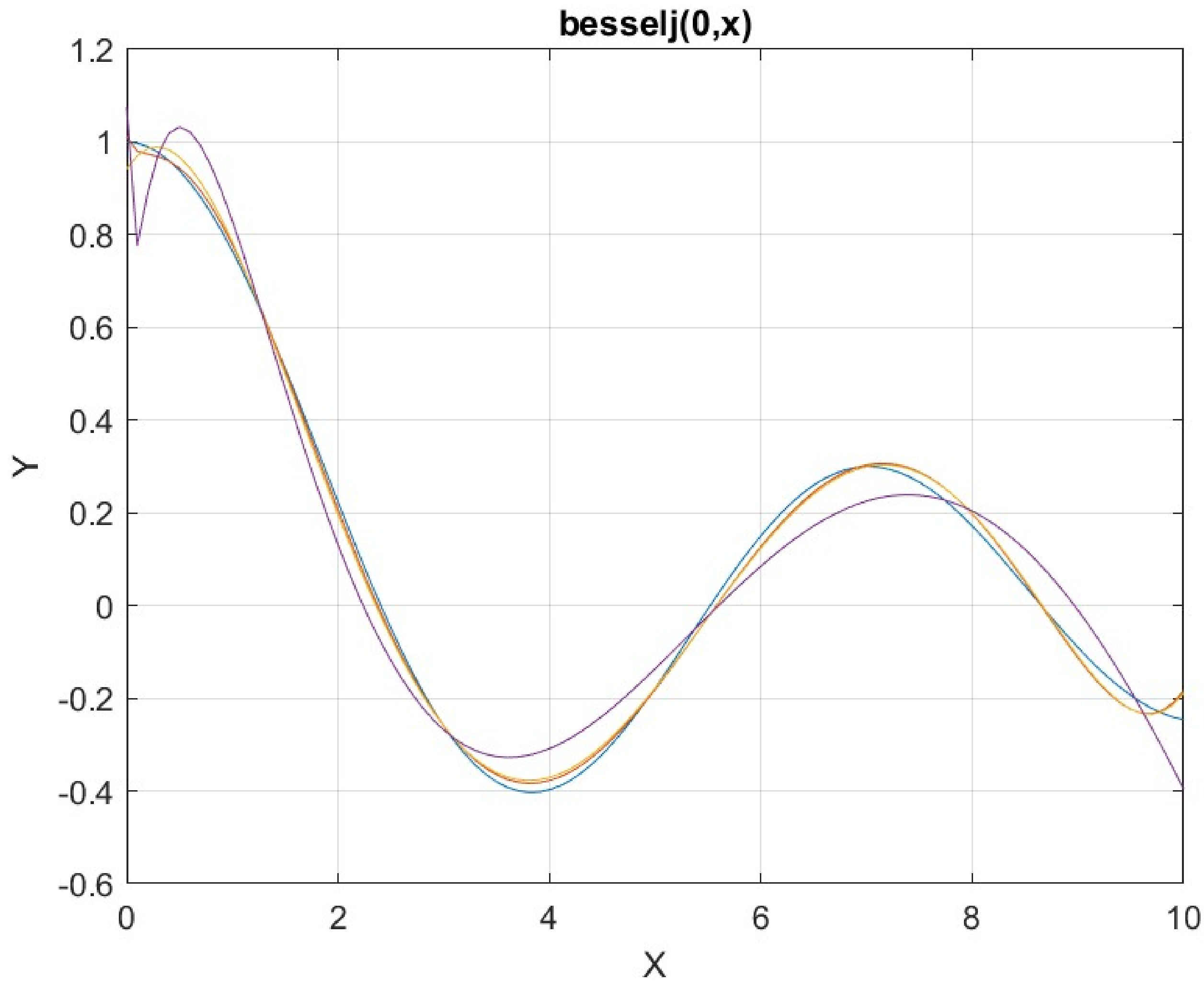

Figure 2. The graph from file besselj_0_x_run2.jpg .

The above graphs let you detect some significant deviations between the Bessel function and the fitted multiple-half-power classical polynomials. The classical polynomial shows lesser deviation from the Bessel function. 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 Polynomials, a fourth order classical polynomial, and a fourth order multiple-half-power 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, 0.9, 0.1);

[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);
fprintf("\nMultiple-half-power  polynomial fit\n");
c2 = polyfit(sqrt(xData),yData,order)
yPoly2 = polyval(c2,sqrt(xData));
r2 = rsqr(yData,yPoly2);
% calculate adjusted value of the coefficient of determination
r2 = 1 - (1 - r2)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r2);
```



```
figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly,xData,yPoly2);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
HalfCoeff = flip(c2);
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");
T4 = array2table(HalfCoeff);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
r_sqr = [glbRsqr r r2];
T5 = array2table(r_sqr);
writetable(T5,xlFile,"Sheet","Sheet1","Range","A13");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr, diffPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = Ub(i-1)+diffPwr;
    Ub(i) = Lb(i) + maxPwr;
  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	
0.966140966	2.08507617	3.174835719	4.280727045	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.987688042	0.070207861	-0.369616522	0.083855597	-0.0048699
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.980927341	0.138170634	-0.457980428	0.113695746	-0.007357698
HalfCoeff1	HalfCoeff2	HalfCoeff3	HalfCoeff4	HalfCoeff5
1.007829532	-0.516126123	1.902850569	-2.256654906	0.628649655
r_sqr1	r_sqr2	r_sqr3		
0.999918258	0.999803041	0.999393195		

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

The above table shows that the adjusted coefficient of determination for the Quantum Shammas Polynomial is higher than that for the other two types of classical polynomials. All three coefficients are good values.

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

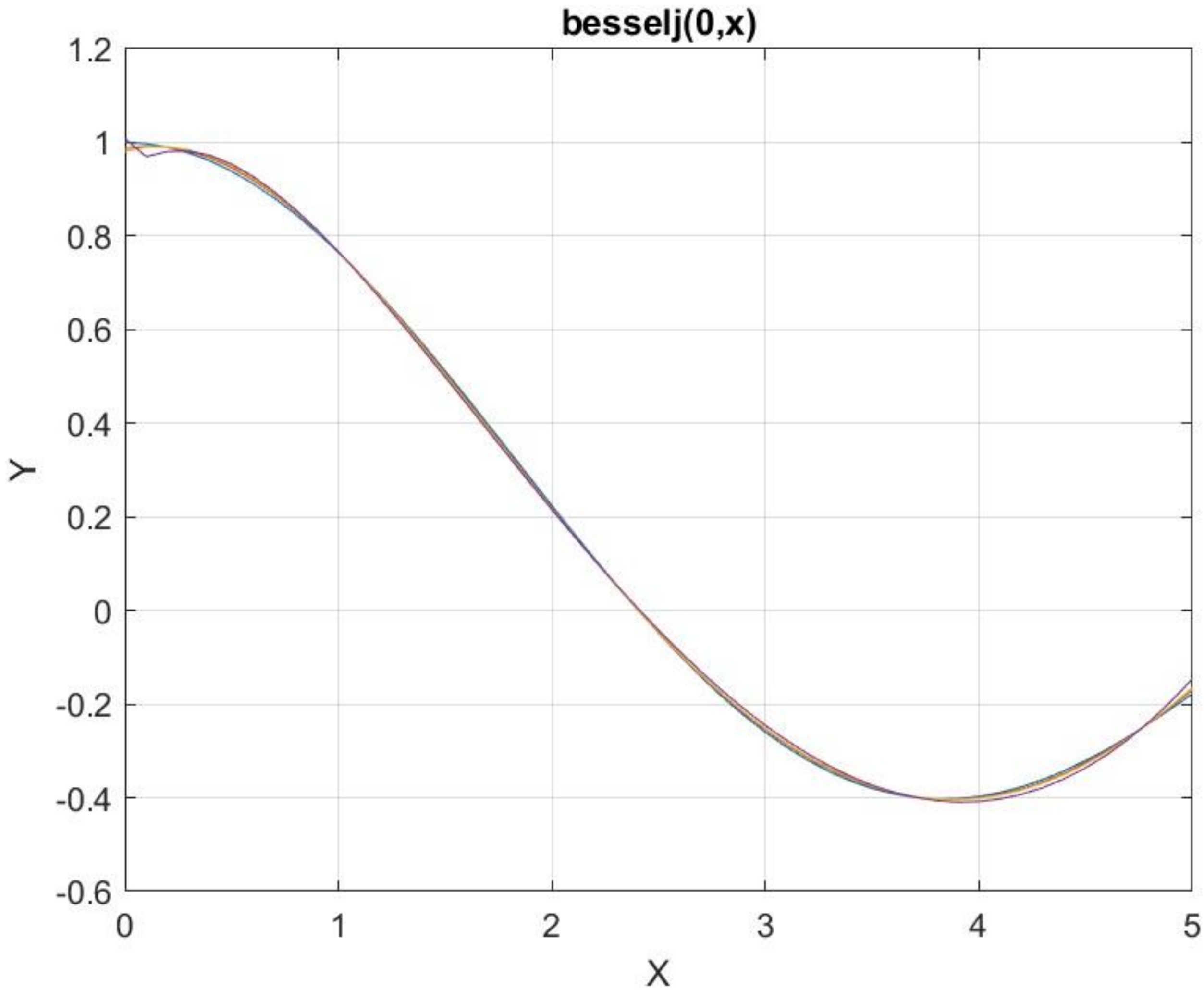

Figure 3. The graph from file besselj_0_x_random_run1.jpg.

The figure shows that all three types of polynomials fit the Bessel function reasonably 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 Polynomials, a sixth order classical polynomial, and a sixth order multiple-half-power classical polynomial.


```
clc
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, 0.9, 0.1);

[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);
fprintf("\nMultiple-half-power  polynomial fit\n");
c2 = polyfit(sqrt(xData),yData,order)
yPoly2 = polyval(c2,sqrt(xData));
r2 = rsqr(yData,yPoly2);
% calculate adjusted value of the coefficient of determination
r2 = 1 - (1 - r2)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r2);
```

```
figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly,xData,yPoly2);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
HalfCoeff = flip(c2);
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");
T4 = array2table(HalfCoeff);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
r_sqr = [glbRsqr r r2];
T5 = array2table(r_sqr);
writetable(T5,xlFile,"Sheet","Sheet1","Range","A13");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr, diffPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = Ub(i-1)+diffPwr;
    Ub(i) = Lb(i) + maxPwr;
  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	
0.938293714	1.143704647	2.223731032	3.152367553	3.641705601	4.526376943	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.025178913	-1.826474903	2.362804318	-1.196586576	0.62068766	-0.209598363	0.006517932
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
HalfCoeff1	HalfCoeff2	HalfCoeff3	HalfCoeff4	HalfCoeff5	HalfCoeff6	HalfCoeff7
1.074934616	-3.913080473	14.34688996	-18.70343239	10.34218791	-2.548257178	0.230499745
r_sqr1	r_sqr2	r_sqr3				
0.99816932	0.996718149	0.964226335				

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 3, albeit lower values for the adjusted coefficients of determination. These lower values are due to the extended range of the x values..

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

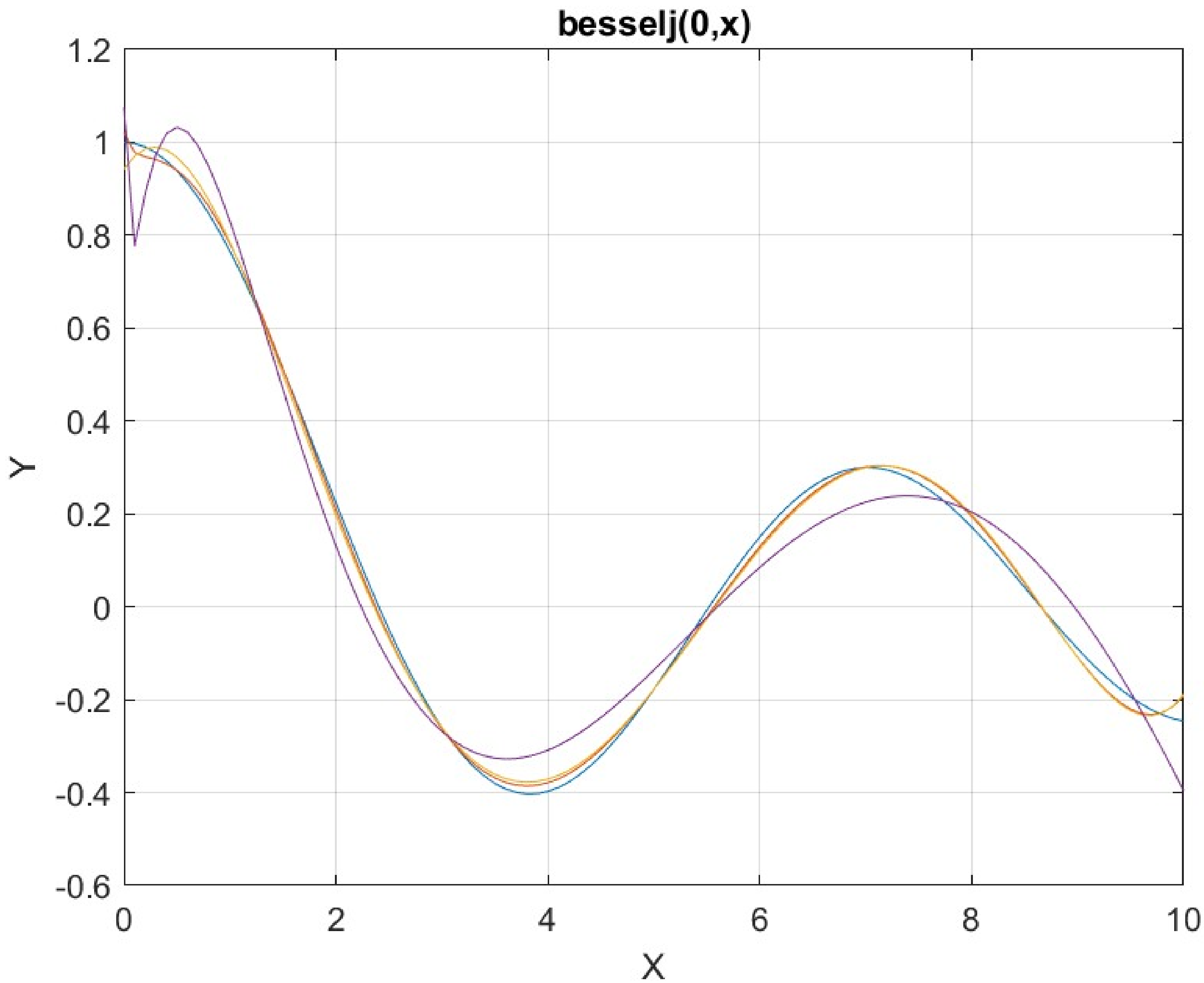

Figure 4. The graph from file besselj_0_x_random_run2.jpg.

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 Polynomials, a fourth order classical polynomial, and a fourth order multiple-half-power 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, 0.9, 0.1);

[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);
fprintf("\nMultiple-half-power  polynomial fit\n");
c2 = polyfit(sqrt(xData),yData,order)
yPoly2 = polyval(c2,sqrt(xData));
r2 = rsqr(yData,yPoly2);
% calculate adjusted value of the coefficient of determination
r2 = 1 - (1 - r2)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r2);
```

```
figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly,xData,yPoly2);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
HalfCoeff = flip(c2);
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");
T4 = array2table(HalfCoeff);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
r_sqr = [glbRsqr r r2];
T5 = array2table(r_sqr);
writetable(T5,xlFile,"Sheet","Sheet1","Range","A13");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr, diffPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = Ub(i-1)+diffPwr;
    Ub(i) = Lb(i) + maxPwr;
  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	
0.961070739	2.085481583	3.174494039	4.222714445	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.987265315	0.072372796	-0.373183773	0.08657281	-0.005726002
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.980927341	0.138170634	-0.457980428	0.113695746	-0.007357698
HalfCoeff1	HalfCoeff2	HalfCoeff3	HalfCoeff4	HalfCoeff5
1.007829532	-0.516126123	1.902850569	-2.256654906	0.628649655
r_sqr1	r_sqr2	r_sqr3		
0.999911452	0.999803041	0.999393195		

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 3. Again, the adjusted coefficient of determination for the Quantum Shammas Polynomial is higher than that for the other two classical polynomials. All coefficients 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:

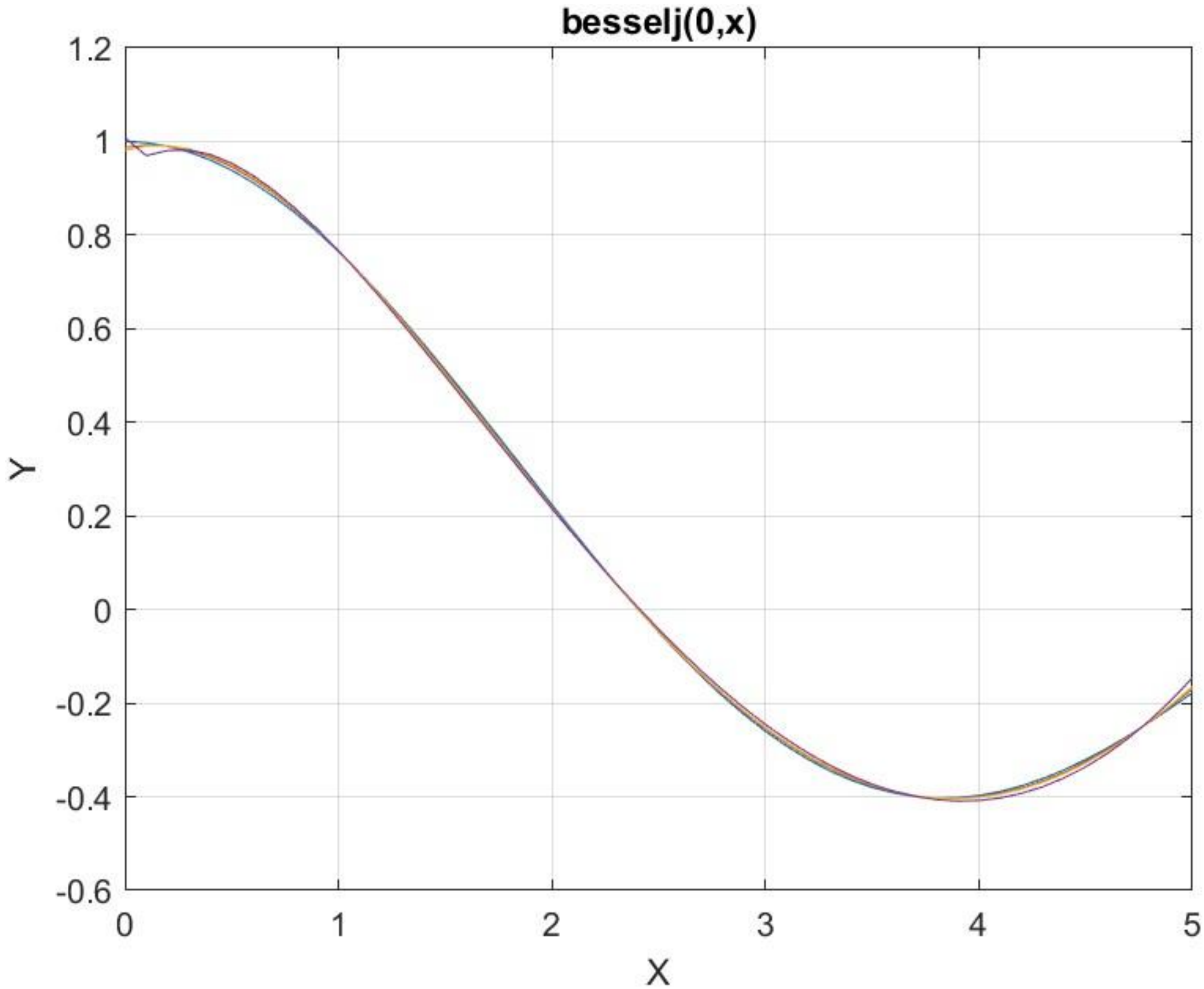

Figure 5. The graph from file besselj_0_x_halton_random_run1.jpg.

The figure shows that all 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 Polynomials, a sixth order classical polynomial, and a sixth order multiple-half-power 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, 0.9, 0.1);

[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);
fprintf("\nMultiple-half-power  polynomial fit\n");
c2 = polyfit(sqrt(xData),yData,order)
yPoly2 = polyval(c2,sqrt(xData));
r2 = rsqr(yData,yPoly2);
% calculate adjusted value of the coefficient of determination
r2 = 1 - (1 - r2)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r2);

figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly,xData,yPoly2);
```

```
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
HalfCoeff = flip(c2);
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");
T4 = array2table(HalfCoeff);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
r_sqr = [glbRsqr r r2];
T5 = array2table(r_sqr);
writetable(T5,xlFile,"Sheet","Sheet1","Range","A13");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr, diffPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = Ub(i-1)+diffPwr;
    Ub(i) = Lb(i) + maxPwr;
  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	
0.703217617	1.608058724	2.224142644	3.098984635	3.658793373	4.512667872	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.020227385	-0.303961534	1.176835744	-1.618136179	0.691040016	-0.192050096	0.00710033
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
HalfCoeff1	HalfCoeff2	HalfCoeff3	HalfCoeff4	HalfCoeff5	HalfCoeff6	HalfCoeff7
1.074934616	-3.913080473	14.34688996	-18.70343239	10.34218791	-2.548257178	0.230499745
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2	r_sqr3				
0.998210559	0.996718149	0.964226335				

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 4. Again, the adjusted coefficient of determination for the Quantum Shammas Polynomial is slightly higher than those for the classical polynomials.

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

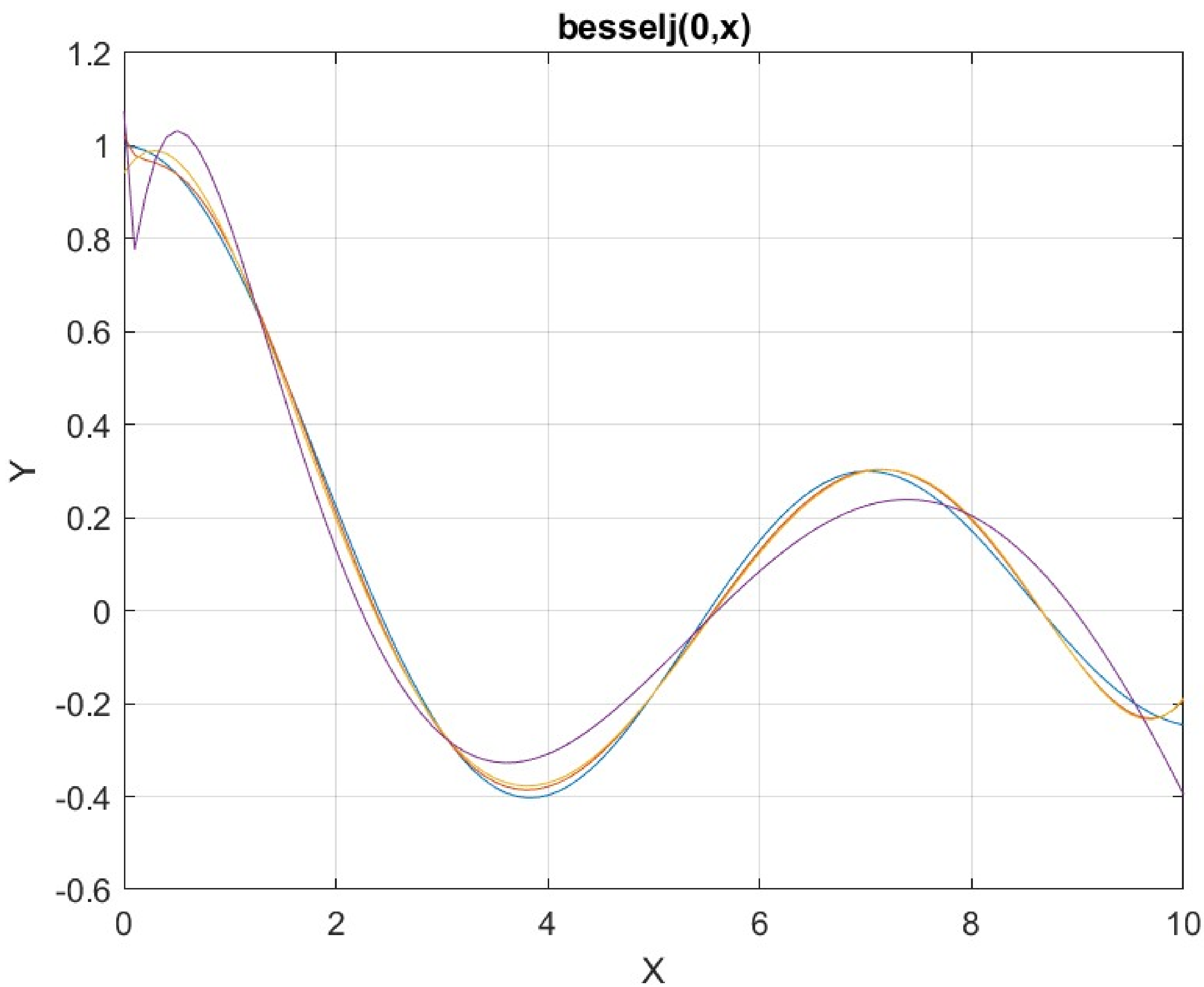

Figure 6. The graph from file besselj_0_x_halton_random_run2.jpg.

The curves in the above figure show some deviations between the three 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 Polynomials, a fourth order classical polynomial, and a fourth order multiple-half-power 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, 0.9, 0.1);

[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);

fprintf("\nMultiple-half-power  polynomial fit\n");
c2 = polyfit(sqrt(xData),yData,order)
yPoly2 = polyval(c2,sqrt(xData));
r2 = rsqr(yData,yPoly2);
% calculate adjusted value of the coefficient of determination
r2 = 1 - (1 - r2)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r2);
```

```
figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly,xData,yPoly2);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
HalfCoeff = flip(c2);
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");
T4 = array2table(HalfCoeff);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
r_sqr = [glbRsqr r r2];
T5 = array2table(r_sqr);
writetable(T5,xlFile,"Sheet","Sheet1","Range","A13");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr, diffPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = Ub(i-1)+diffPwr;
    Ub(i) = Lb(i) + maxPwr;
  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	
0.93445719	2.079889698	3.167459654	4.282865413	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.987160687	0.070815854	-0.369757668	0.08368228	-0.004728475
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.980927341	0.138170634	-0.457980428	0.113695746	-0.007357698
HalfCoeff1	HalfCoeff2	HalfCoeff3	HalfCoeff4	HalfCoeff5
1.007829532	-0.516126123	1.902850569	-2.256654906	0.628649655
r_sqr1	r_sqr2	r_sqr3		
0.999912041	0.999803041	0.999393195		

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 5. Again, the adjusted coefficient of determination for the Quantum Shammas Polynomial is higher than those for the classical polynomials. All three coefficients have 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:

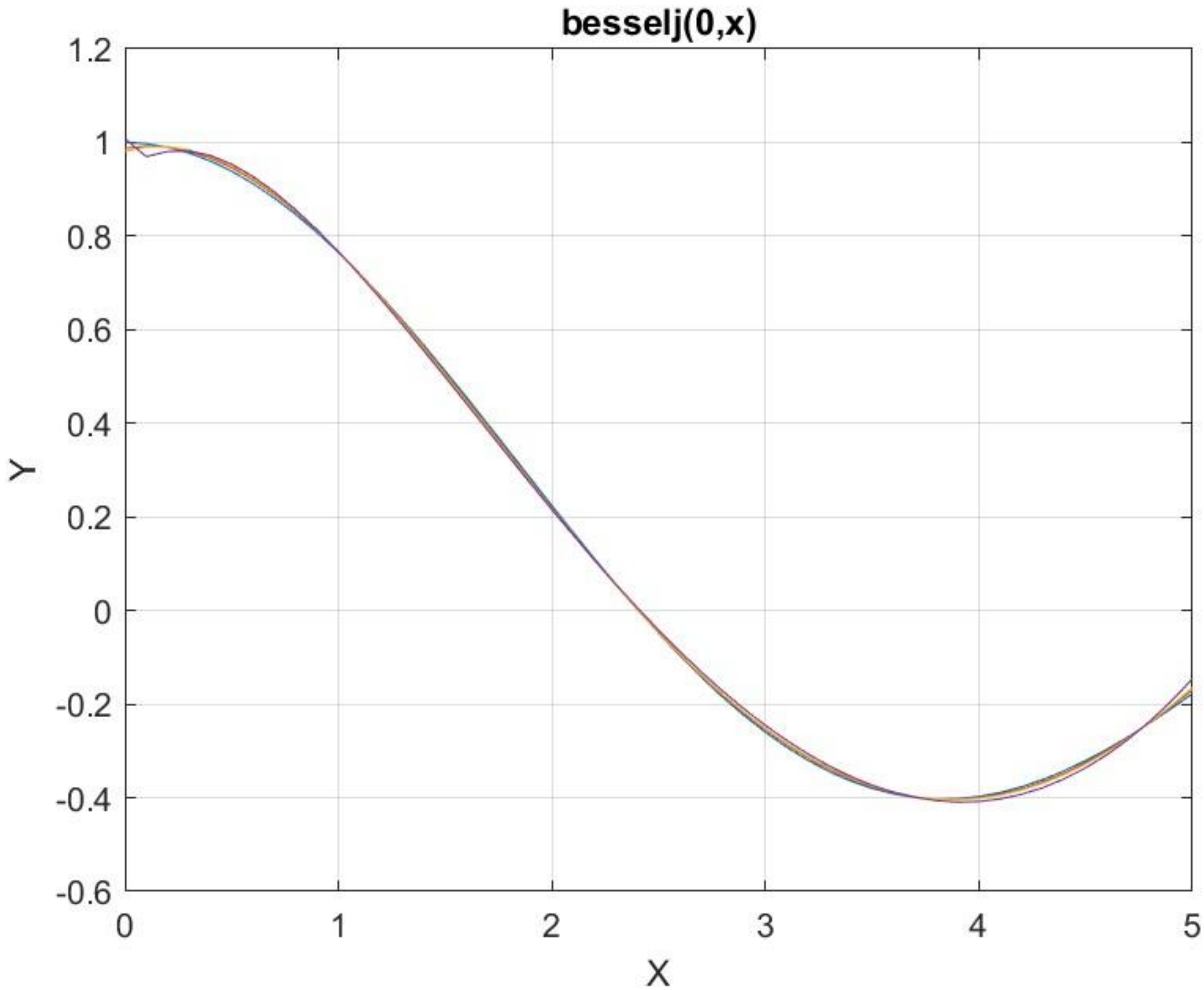

Figure 7. The graph from file besselj_0_x_sobol_random_run1.jpg.

The figure shows that all three 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 Polynomials, a sixth order classical polynomial, and a sixth order multiple-half-power 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, 0.9, 0.1);

[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);

fprintf("\nMultiple-half-power  polynomial fit\n");
c2 = polyfit(sqrt(xData),yData,order)
yPoly2 = polyval(c2,sqrt(xData));
r2 = rsqr(yData,yPoly2);
% calculate adjusted value of the coefficient of determination
r2 = 1 - (1 - r2)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r2);
```

```
figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly,xData,yPoly2);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
HalfCoeff = flip(c2);
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");
T4 = array2table(HalfCoeff);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
r_sqr = [glbRsqr r r2];
T5 = array2table(r_sqr);
writetable(T5,xlFile,"Sheet","Sheet1","Range","A13");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr, diffPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = Ub(i-1)+diffPwr;
    Ub(i) = Lb(i) + maxPwr;
  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	
0.944691194	1.370798537	2.189795949	3.181299055	3.607044555	4.528852968	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.020940681	-0.867125201	1.607687299	-1.402568459	0.677565943	-0.262294621	0.006464675
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
HalfCoeff1	HalfCoeff2	HalfCoeff3	HalfCoeff4	HalfCoeff5	HalfCoeff6	HalfCoeff7
1.074934616	-3.913080473	14.34688996	-18.70343239	10.34218791	-2.548257178	0.230499745
r_sqr1	r_sqr2	r_sqr3				
0.998224675	0.996718149	0.964226335				

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 ones 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:

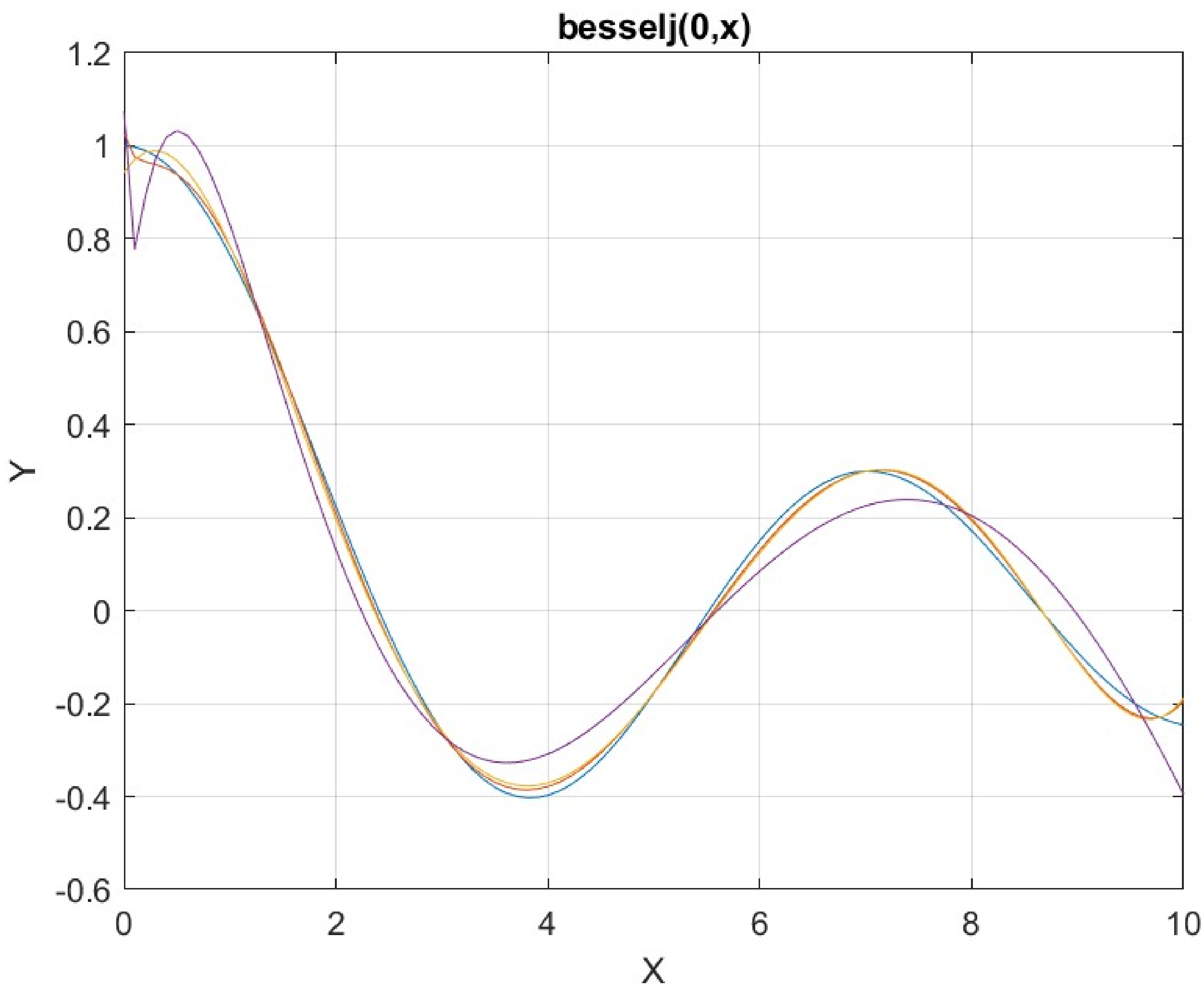

Figure 8. The graph from file besselj_0_x_sobol_random_run2.jpg

Again, the above curves show some deviations between the three 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 using the Quantum Shammas Polynomials 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 Polynomials, a fourth order classical polynomial, and a fourth order multiple-half-power classical polynomial.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "Ln_x_pso";
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, 0.9, 0.1);

[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);

fprintf("\nMultiple-half-power  polynomial fit\n");
c2 = polyfit(sqrt(xData),yData,order)
yPoly2 = polyval(c2,sqrt(xData));
r2 = rsqr(yData,yPoly2);
% calculate adjusted value of the coefficient of determination
r2 = 1 - (1 - r2)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r2);

figure(1)
plot(xData0,yData,xData0,yCalc,xData0,yPoly,xData0,yPoly2);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
HalfCoeff = flip(c2);
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");
T4 = array2table(HalfCoeff);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
r_sqr = [glbRsqr r r2];
T5 = array2table(r_sqr);
writetable(T5,xlFile,"Sheet","Sheet1","Range","A13");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr, diffPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = Ub(i-1)+diffPwr;
    Ub(i) = Lb(i) + maxPwr;
  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 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	
0.899984012	1.897240263	2.725123225	3.044743814	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
-0.005624187	0.853854947	-0.202435017	0.068650455	-0.022782971
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.019526836	0.850579688	-0.210226108	0.031956872	-0.001944825
HalfCoeff1	HalfCoeff2	HalfCoeff3	HalfCoeff4	HalfCoeff5
-0.004531013	0.046083384	1.061031303	-0.485458483	0.07253001
r_sqr1	r_sqr2	r_sqr3		
0.999993563	0.99989954	0.999977288		

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

The adjusted coefficient of determination for the Quantum Shammas Polynomial is higher than the ones for classical polynomials. Also, the adjusted coefficient of determination for the multiple-half-power classical polynomial is higher than the one for classical polynomial.

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

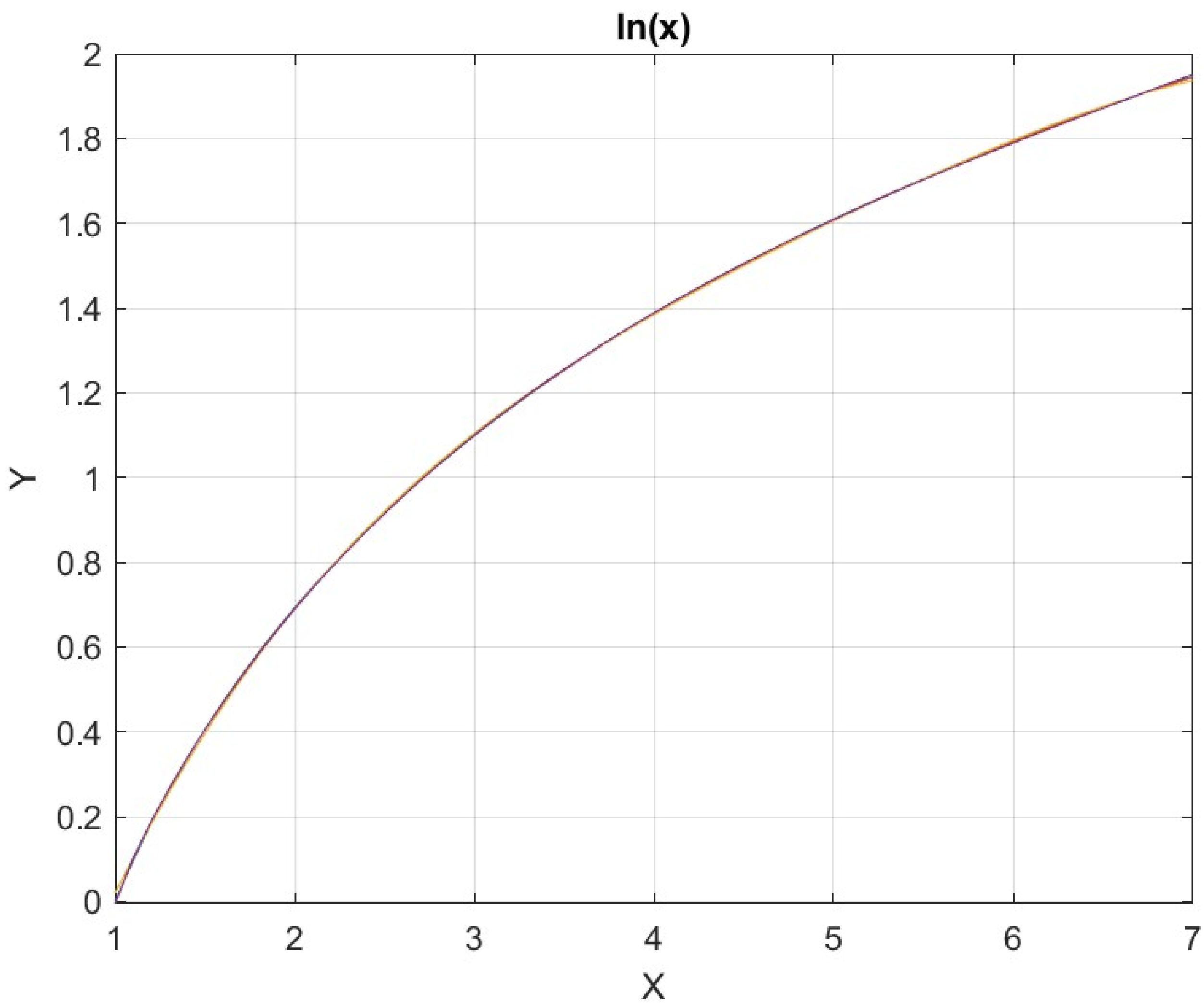

Figure 9. The graph from file Ln_x_pso.jpg.

The above graph shows that the three 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 Polynomials, a fourth order classical polynomial, and a fourth order multiple-half-power 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, 0.9, 0.1);

[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);
fprintf("\nMultiple-half-power  polynomial fit\n");
c2 = polyfit(sqrt(xData),yData,order)
yPoly2 = polyval(c2,sqrt(xData));
r2 = rsqr(yData,yPoly2);
% calculate adjusted value of the coefficient of determination
r2 = 1 - (1 - r2)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r2);
```

```
figure(1)
plot(xData0,yData,xData0,yCalc,xData0,yPoly,xData0,yPoly2);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
HalfCoeff = flip(c2);
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");
T4 = array2table(HalfCoeff);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
r_sqr = [glbRsqr r r2];
T5 = array2table(r_sqr);
writetable(T5,xlFile,"Sheet","Sheet1","Range","A13");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr, diffPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = Ub(i-1)+diffPwr;
    Ub(i) = Lb(i) + maxPwr;
  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 testLog1pso.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	
0.980498795	1.707571371	2.457725889	2.729662457	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
-0.00119309	1.013632733	-0.411909225	0.143763625	-0.051841352
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.019526836	0.850579688	-0.210226108	0.031956872	-0.001944825
HalfCoeff1	HalfCoeff2	HalfCoeff3	HalfCoeff4	HalfCoeff5
-0.004531013	0.046083384	1.061031303	-0.485458483	0.07253001
r_sqr1	r_sqr2	r_sqr3		
0.999998199	0.99989954	0.999977288		

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 ones for classical polynomials. Also, the adjusted coefficient of determination for the multiple-half-power classical polynomial is higher than the one for classical polynomial.

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

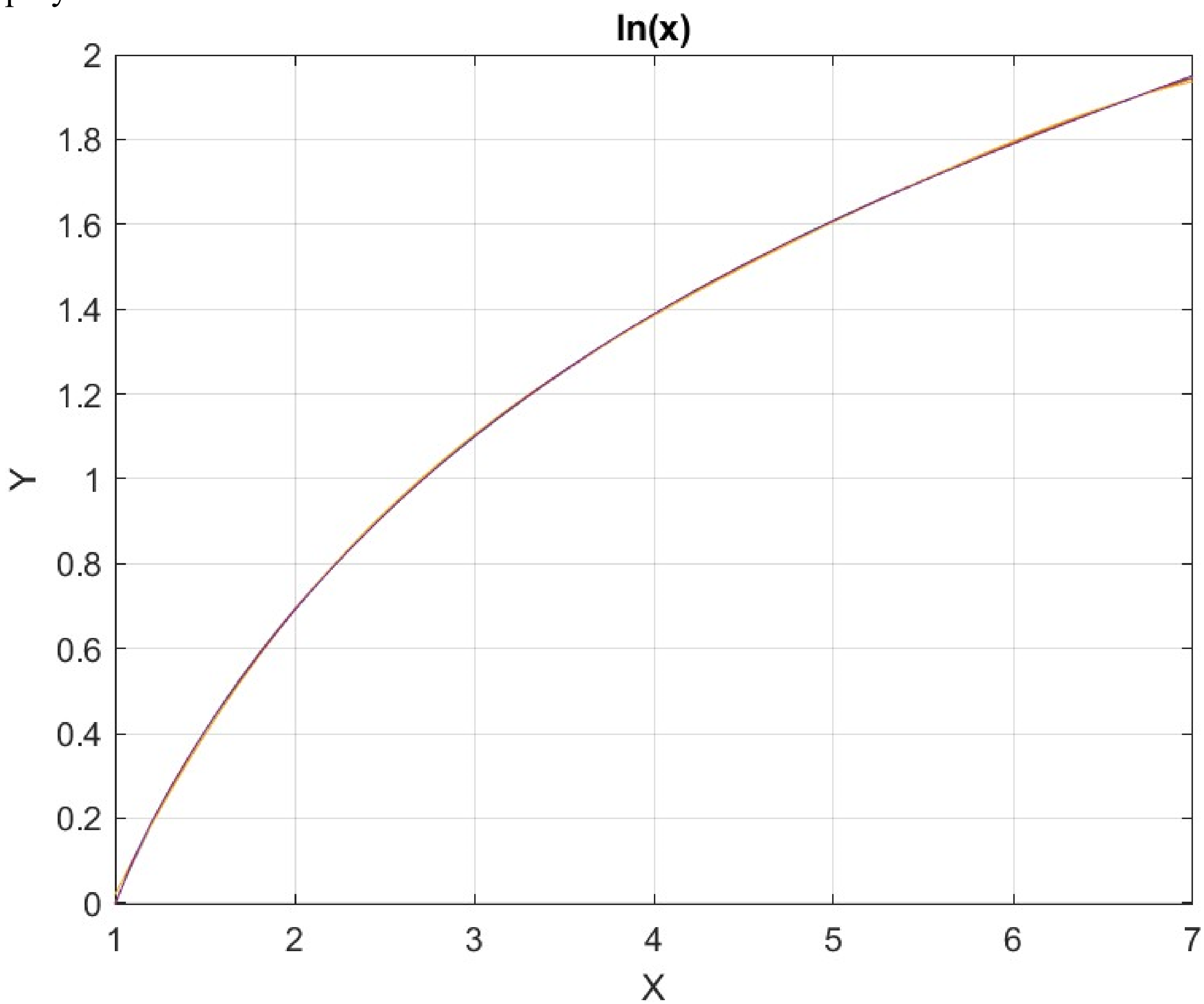

Figure 10. The graph from file ln_x_rand.jpg

The above graph shows that the three 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 Polynomials, a fourth order classical polynomial, and a fourth order multiple-half-power 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, 0.9, 0.1);

[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);
fprintf("\nMultiple-half-power  polynomial fit\n");
c2 = polyfit(sqrt(xData),yData,order)
yPoly2 = polyval(c2,sqrt(xData));
r2 = rsqr(yData,yPoly2);
% calculate adjusted value of the coefficient of determination
r2 = 1 - (1 - r2)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r2);
```

```
figure(1)
plot(xData0,yData,xData0,yCalc,xData0,yPoly,xData0,yPoly2);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
HalfCoeff = flip(c2);
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");
T4 = array2table(HalfCoeff);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
r_sqr = [glbRsqr r r2];
T5 = array2table(r_sqr);
writetable(T5,xlFile,"Sheet","Sheet1","Range","A13");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr, diffPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = Ub(i-1)+diffPwr;
    Ub(i) = Lb(i) + maxPwr;
  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 haltonlRandomSearch() and requests a million random searches. The above file generates the following Excel table summary.

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	
0.974525856	1.718234604	2.521890735	2.741571108	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
-0.001223902	0.99536169	-0.382992382	0.146609323	-0.06534608
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.019526836	0.850579688	-0.210226108	0.031956872	-0.001944825
HalfCoeff1	HalfCoeff2	HalfCoeff3	HalfCoeff4	HalfCoeff5
-0.004531013	0.046083384	1.061031303	-0.485458483	0.07253001
r_sqr1	r_sqr2	r_sqr3		
0.999997924	0.99989954	0.999977288		

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

The results in Table 11 agree with those in Table 10. The Quantum Shammas Polynomial has the highest adjusted coefficient of determination.

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

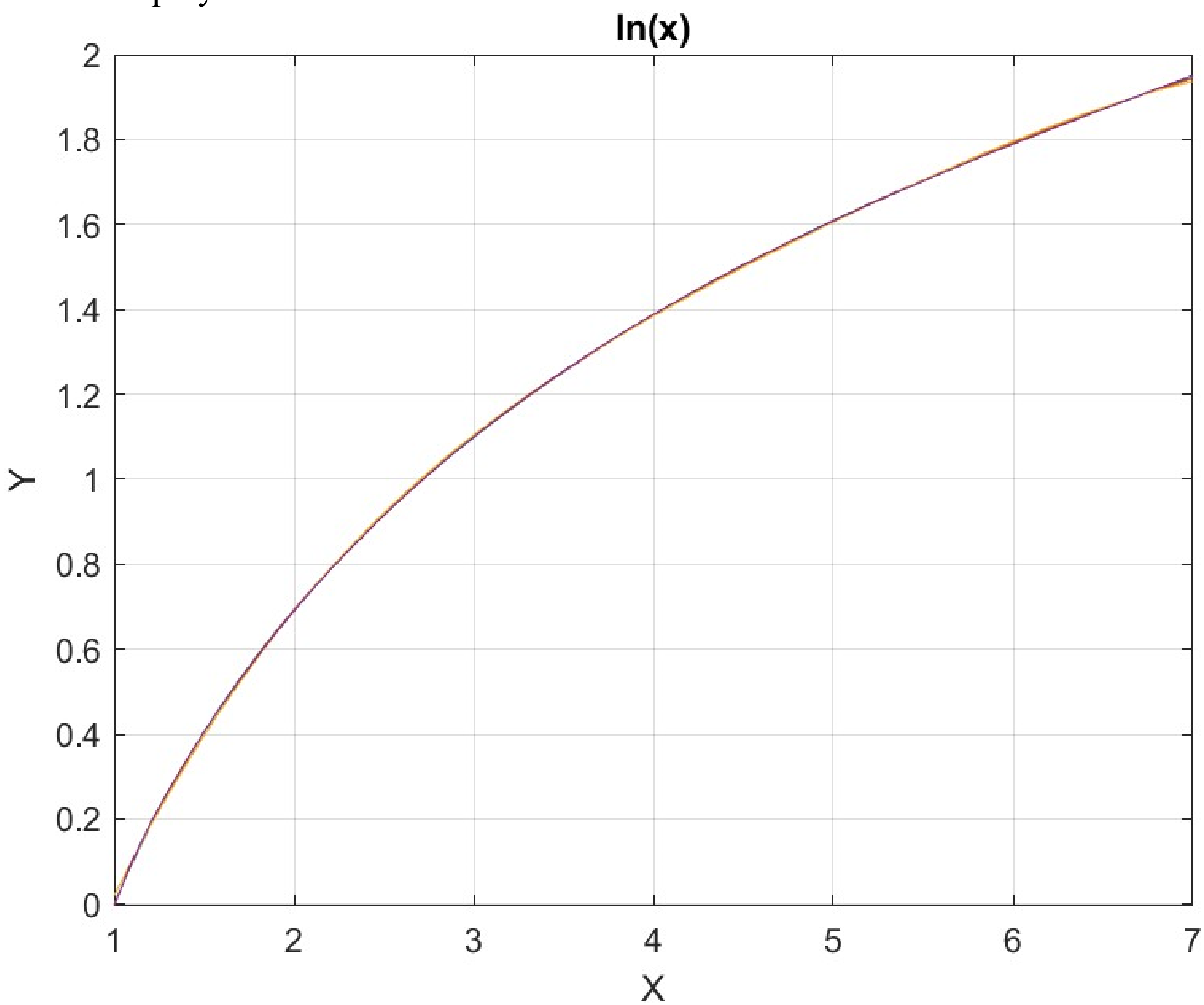

Figure 11. The graph from file ln_x_halton_rand.jpg

The above graph shows that the three 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 Polynomials, a fourth order classical polynomial, and a fourth order multiple-half-power 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, 0.9, 0.1);

[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);

fprintf("\nMultiple-half-power  polynomial fit\n");
c2 = polyfit(sqrt(xData),yData,order)
yPoly2 = polyval(c2,sqrt(xData));
r2 = rsqr(yData,yPoly2);
% calculate adjusted value of the coefficient of determination
r2 = 1 - (1 - r2)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r2);
```



```
figure(1)
plot(xData0,yData,xData0,yCalc,xData0,yPoly,xData0,yPoly2);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
HalfCoeff = flip(c2);
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");
T4 = array2table(HalfCoeff);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
r_sqr = [glbRsqr r r2];
T5 = array2table(r_sqr);
writetable(T5,xlFile,"Sheet","Sheet1","Range","A13");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr, diffPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = Ub(i-1)+diffPwr;
    Ub(i) = Lb(i) + maxPwr;
  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	
0.973254289	1.719141588	2.381499582	2.829155211	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
-0.001802469	0.9999281	-0.402983718	0.119298669	-0.022064167
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.019526836	0.850579688	-0.210226108	0.031956872	-0.001944825
HalfCoeff1	HalfCoeff2	HalfCoeff3	HalfCoeff4	HalfCoeff5
-0.004531013	0.046083384	1.061031303	- 0.485458483	0.07253001
r_sqr1	r_sqr2	r_sqr3		
0.999998136	0.99989954	0.999977288		

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

The results in Table 12 agree with those in Table 10. The Quantum Shammas Polynomial has the highest adjusted coefficient of determination.

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

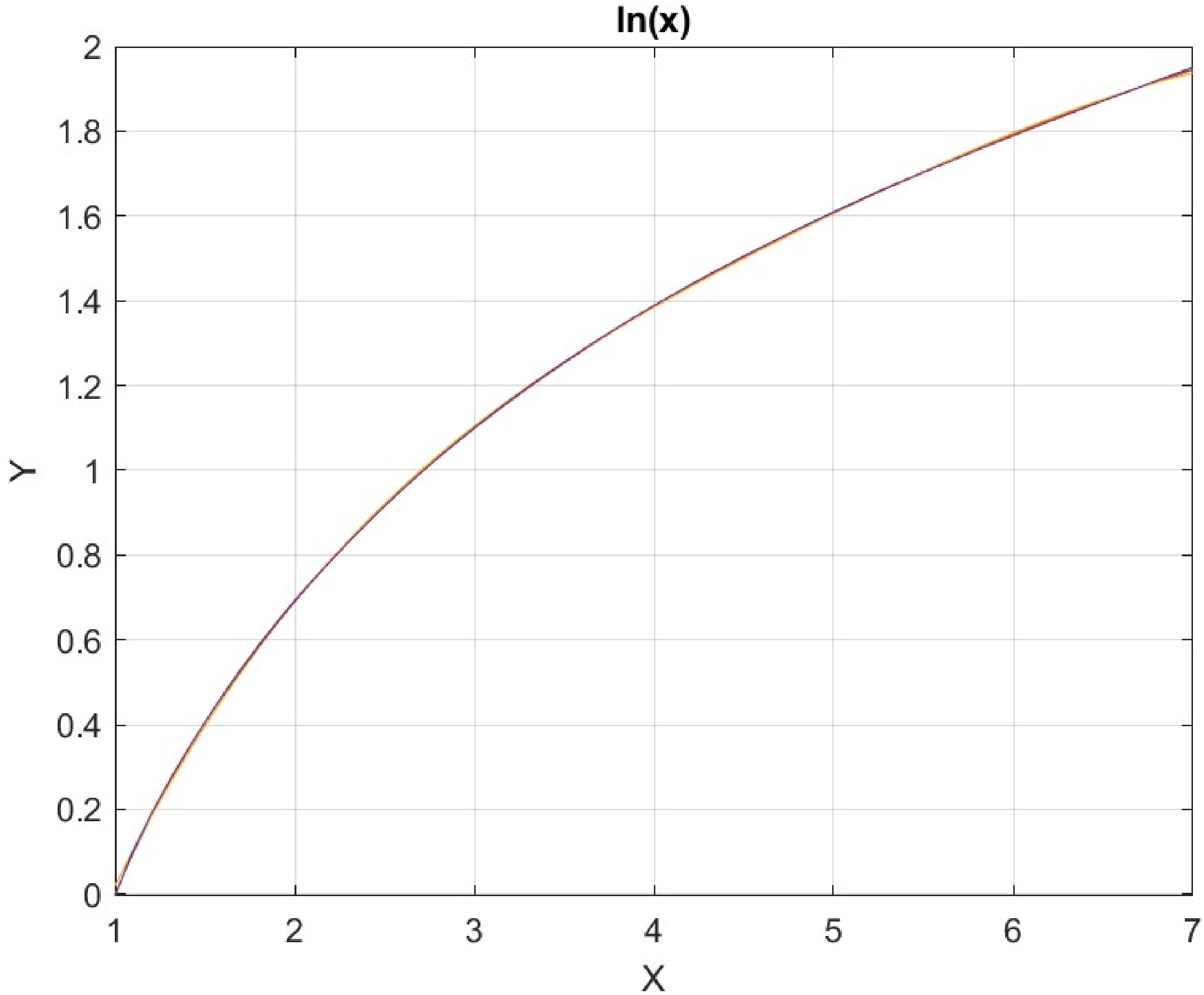

Figure 12. The graph from file ln_x_sobol_rand.jpg

The above graph shows that the three 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 1C 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 Polynomials, a fourth order classical polynomial, and a fourth order multiple-half-power 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, 0.9, 0.1);

[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);
```

```
fprintf("\nMultiple-half-power  polynomial fit\n");
c2 = polyfit(sqrt(xData),yData,order)
yPoly2 = polyval(c2,sqrt(xData));
r2 = rsqr(yData,yPoly2);
% calculate adjusted value of the coefficient of determination
r2 = 1 - (1 - r2)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r2);

figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly,xData,yPoly2);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
HalfCoeff = flip(c2);
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");
T4 = array2table(HalfCoeff);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
r_sqr = [glbRsqr r r2];
T5 = array2table(r_sqr);
writetable(T5,xlFile,"Sheet","Sheet1","Range","A13");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr, diffPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = Ub(i-1)+diffPwr;
    Ub(i) = Lb(i) + maxPwr;
  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	
0.896636535	1.899929816	2.893178186	3.898574975	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.396102099	0.04764752	-0.327132974	0.142274653	-0.01791295
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.397644494	0.028633101	-0.306018517	0.139989216	-0.018592075
HalfCoeff1	HalfCoeff2	HalfCoeff3	HalfCoeff4	HalfCoeff5
0.396918876	0.000134222	0.215508152	-0.60968181	0.237656369
r_sqr1	r_sqr2	r_sqr3		
0.999933723	0.999967249	0.998959391		

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

The adjusted coefficient of determination for the Quantum Shammas Polynomial is slightly less than the one for classical polynomial, but higher than that of the multiple-half-power classical polynomial.

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

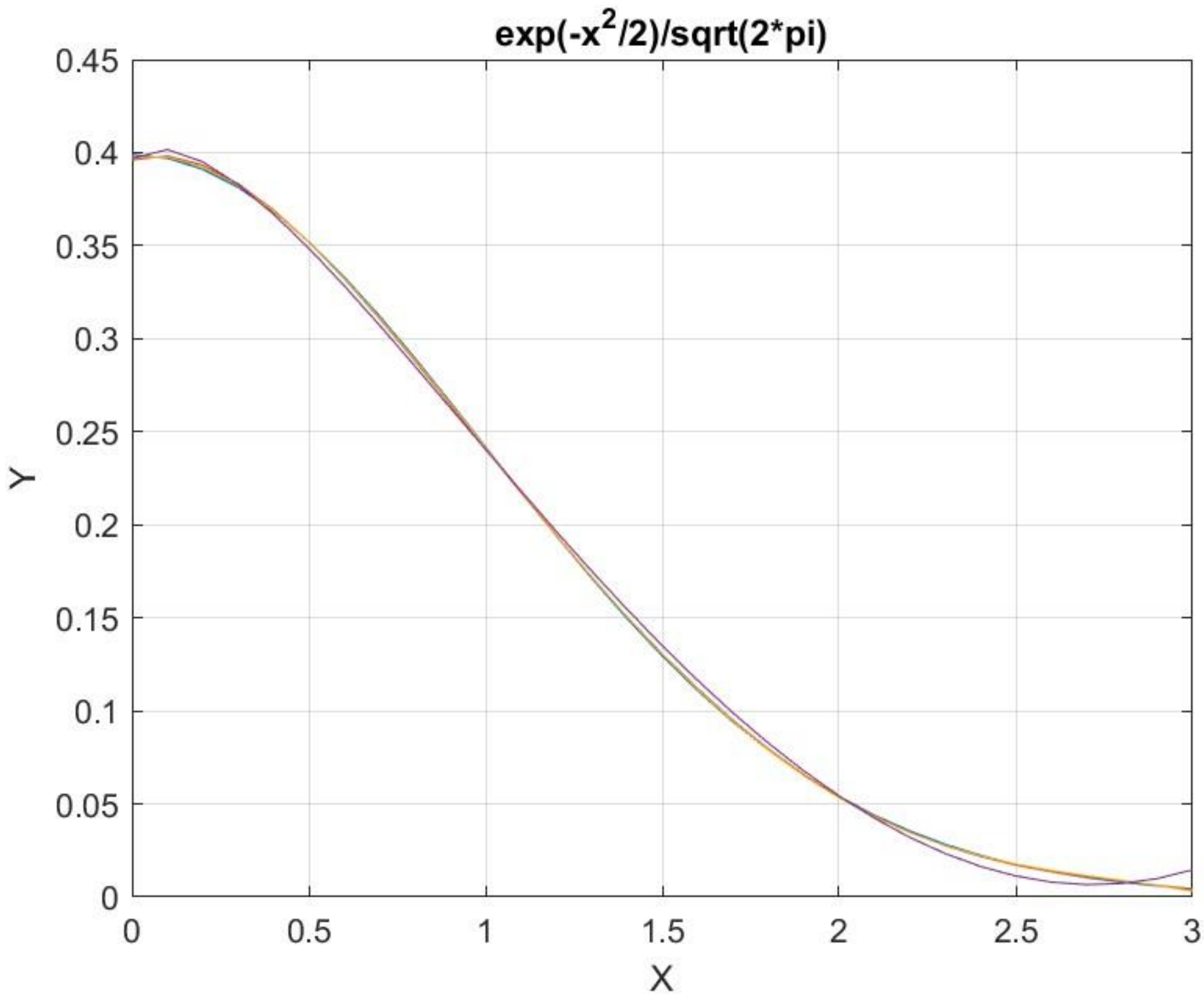

Figure 13. The graph from file Right_GaussBell_x.jpg.

The above graph shows that the Quantum Shammas Polynomial and the classical polynomial fit the right normal Gauss function well. The multiple-half-power classical polynomial shows more deviation from the Gauss bell curve.

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 Polynomials, a fourth order classical polynomial, and a fourth order multiple-half-power 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, 0.9, 0.1);

[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);
fprintf("\nMultiple-half-power  polynomial fit\n");
c2 = polyfit(sqrt(xData),yData,order)
yPoly2 = polyval(c2,sqrt(xData));
r2 = rsqr(yData,yPoly2);
```

```
% calculate adjusted value of the coefficient of determination
r2 = 1 - (1 - r2)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r2);

figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly,xData,yPoly2);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
HalfCoeff = flip(c2);
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");
T4 = array2table(HalfCoeff);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
r_sqr = [glbRsqr r r2];
T5 = array2table(r_sqr);
writetable(T5,xlFile,"Sheet","Sheet1","Range","A13");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr, diffPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = Ub(i-1)+diffPwr;
    Ub(i) = Lb(i) + maxPwr;
  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	
0.973670568	2.087118867	3.165850956	3.620544949	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.39792387	0.018065735	-0.310054896	0.220335359	-0.084565043
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.397644494	0.028633101	-0.306018517	0.139989216	-0.018592075
HalfCoeff1	HalfCoeff2	HalfCoeff3	HalfCoeff4	HalfCoeff5
0.396918876	0.000134222	0.215508152	-0.60968181	0.237656369
r_sqr1	r_sqr2	r_sqr3		
0.999972658	0.999967249	0.998959391		

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, and higher than the one for the multiple-half-power classical polynomial.

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

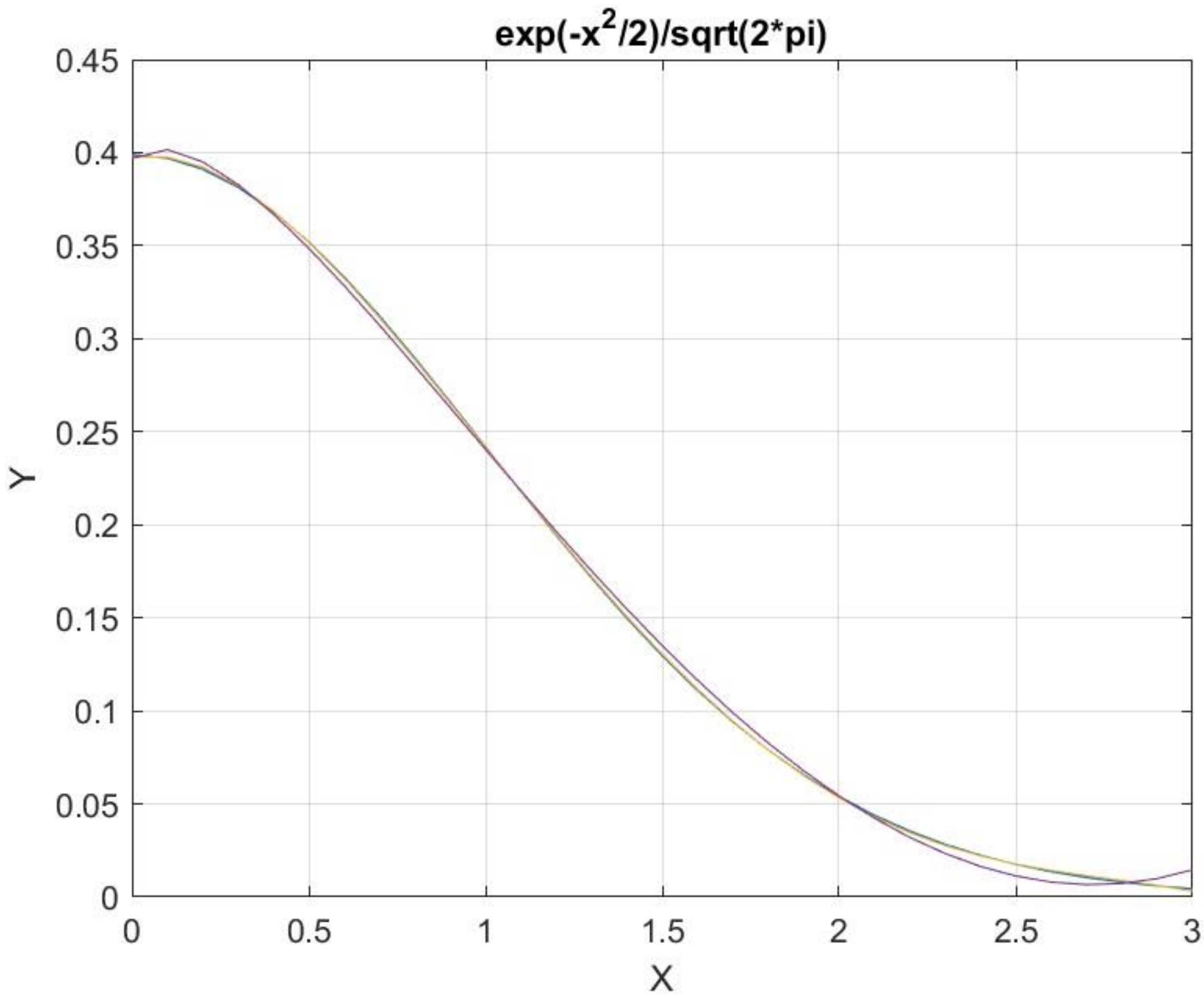

Figure 14. The graph from file Right_GaussBell_x_random.jpg.

The above graph shows that the Quantum Shammas Polynomial and the classical polynomial fit the right normal Gauss function well. The multiple-half-power classical polynomial shows more deviation from the Gauss bell curve.

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

Polynomials, a fourth order classical polynomial, and a fourth order multiple-half-power 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, 0.9, 0.1);

[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);
fprintf("\nMultiple-half-power  polynomial fit\n");
c2 = polyfit(sqrt(xData),yData,order)
yPoly2 = polyval(c2,sqrt(xData));
r2 = rsqr(yData,yPoly2);
```



```
% calculate adjusted value of the coefficient of determination
r2 = 1 - (1 - r2)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r2);

figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly,xData,yPoly2);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
HalfCoeff = flip(c2);
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");
T4 = array2table(HalfCoeff);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
r_sqr = [glbRsqr r r2];
T5 = array2table(r_sqr);
writetable(T5,xlFile,"Sheet","Sheet1","Range","A13");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr, diffPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = Ub(i-1)+diffPwr;
    Ub(i) = Lb(i) + maxPwr;
  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	
0.984425916	2.088389501	3.172752238	3.574004836	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.397847548	0.019473417	-0.315059691	0.243246354	-0.103843917
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.397644494	0.028633101	-0.306018517	0.139989216	-0.018592075
HalfCoeff1	HalfCoeff2	HalfCoeff3	HalfCoeff4	HalfCoeff5
0.396918876	0.000134222	0.215508152	-0.60968181	0.237656369
r_sqr1	r_sqr2	r_sqr3		
0.9999729	0.999967249	0.998959391		

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, and higher than the one for the multiple-half-power classical polynomial.

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

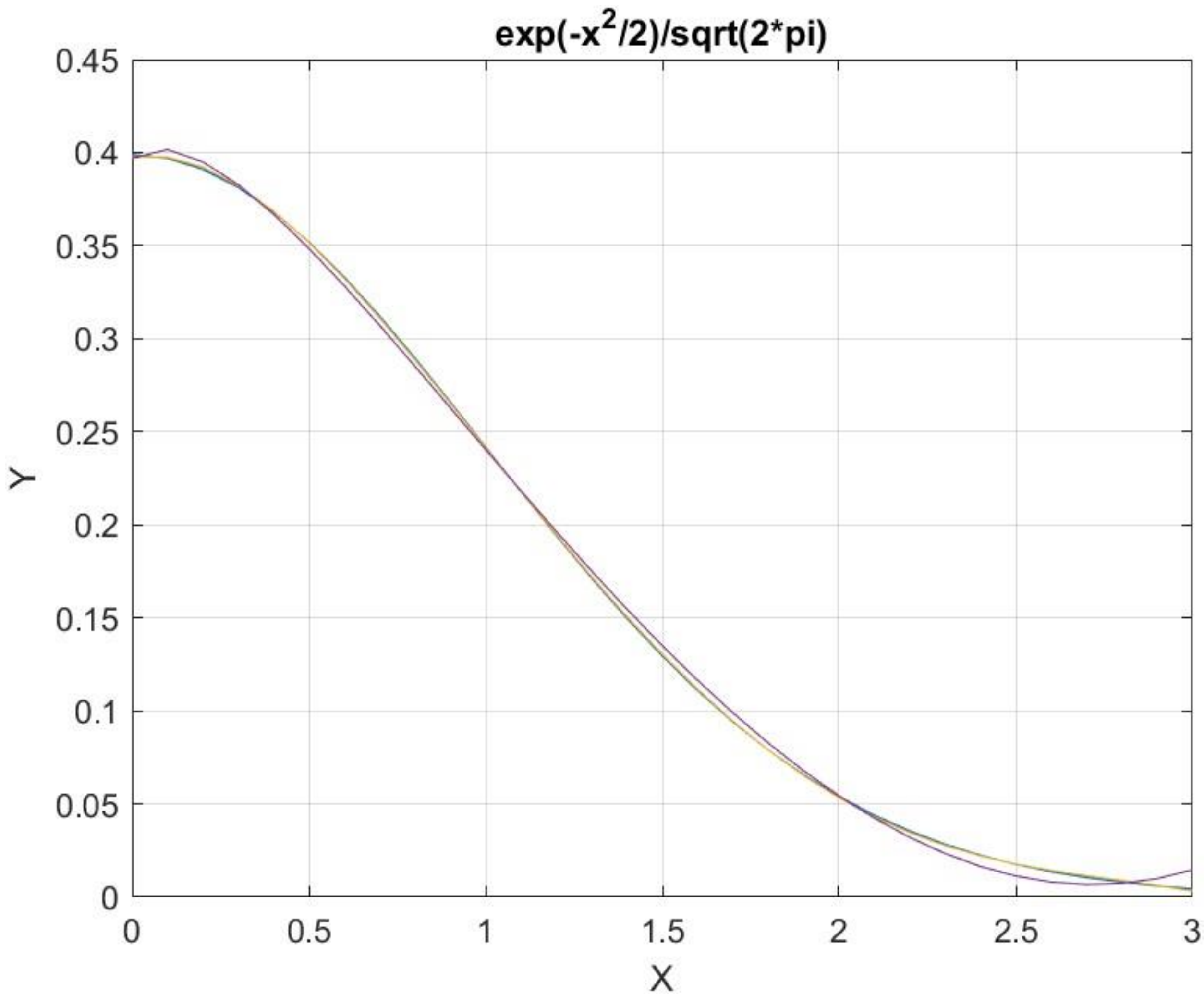

Figure 15. The graph from file Right_GaussBell_x_halton_random.jpg.

The above graph shows that the Quantum Shammas Polynomial and the classical polynomial fit the right normal Gauss function well. The multiple-half-power classical polynomial shows more deviation from the Gauss bell curve.

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

Polynomials, a fourth order classical polynomial, and a fourth order multiple-half-power 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, 0.9, 0.1);

[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);
fprintf("\nMultiple-half-power  polynomial fit\n");
c2 = polyfit(sqrt(xData),yData,order)
yPoly2 = polyval(c2,sqrt(xData));
```

```
r2 = rsqr(yData,yPoly2);
% calculate adjusted value of the coefficient of determination
r2 = 1 - (1 - r2)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", r2);

figure(1)
plot(xData,yData,xData,yCalc,xData,yPoly,xData,yPoly2);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
HalfCoeff = flip(c2);
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");
T4 = array2table(HalfCoeff);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
r_sqr = [glbRsqr r r2];
T5 = array2table(r_sqr);
writetable(T5,xlFile,"Sheet","Sheet1","Range","A13");

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr, diffPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = Ub(i-1)+diffPwr;
    Ub(i) = Lb(i) + maxPwr;
  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	
0.982618236	2.084156284	3.175713414	3.612630613	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.397927187	0.018480133	-0.309172965	0.223734882	-0.089262375
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.397644494	0.028633101	-0.306018517	0.139989216	-0.018592075
HalfCoeff1	HalfCoeff2	HalfCoeff3	HalfCoeff4	HalfCoeff5
0.396918876	0.000134222	0.215508152	-0.60968181	0.237656369
r_sqr1	r_sqr2	r_sqr3		
0.999972712	0.999967249	0.998959391		

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

Table 16 affirms the same conclusions as in Tables 14 and 15. The Quantum Shammas Polynomial has the highest adjusted coefficient of determination.

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

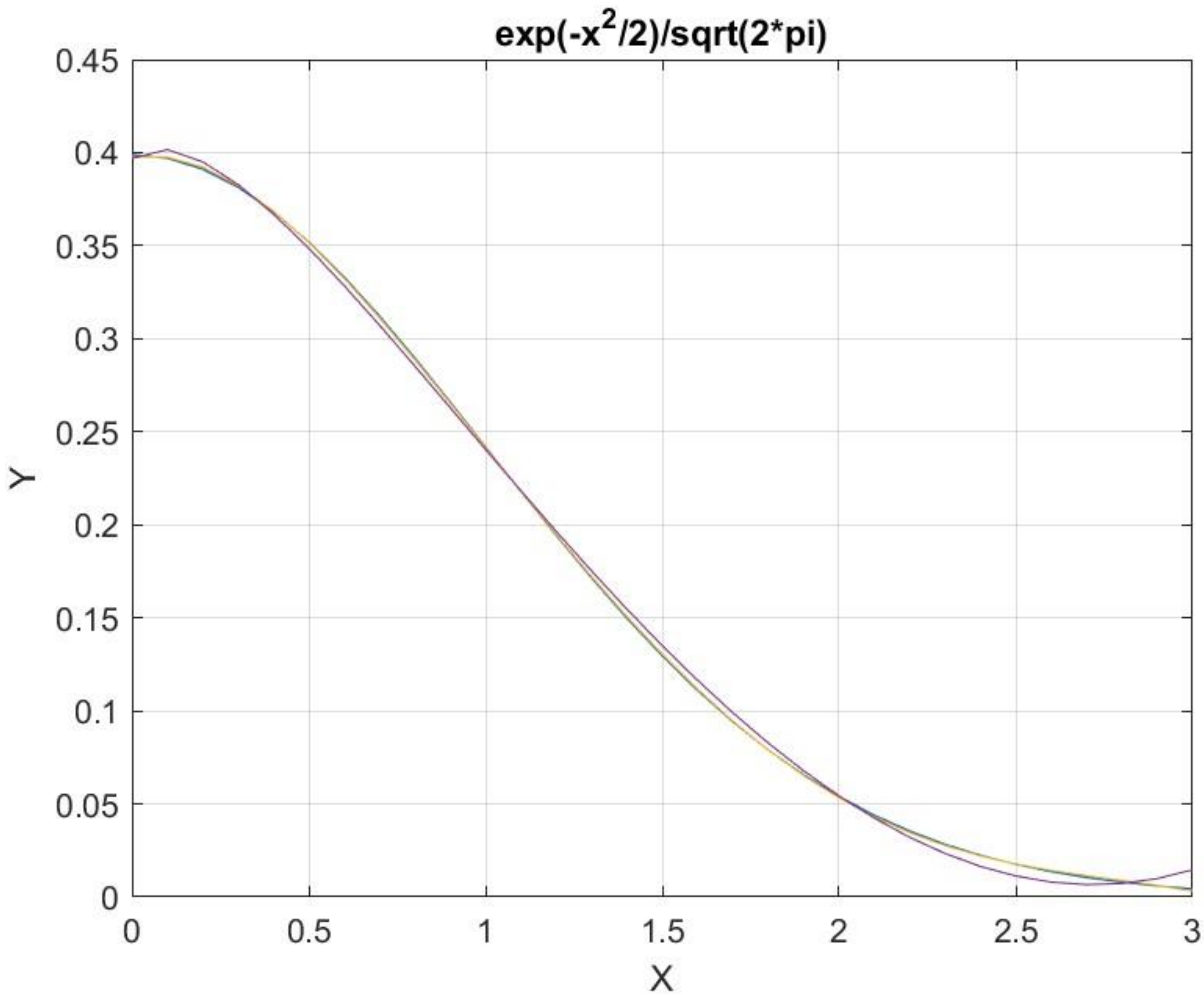

Figure 16. The graph from file Right_GaussBell_x_sobol_random.jpg.

The above graph shows that the Quantum Shammas Polynomial and the classical polynomial fit the right normal Gauss function well. The multiple-half-power classical polynomial shows more deviation from the Gauss bell curve.

Conclusion for fitting the Right-Side Normal Gaussian Function

The above four subsections show that fitting the right-side normal N(0, 1) 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 1C

The Quantum Shammas Polynomials, with narrower power ranges, 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 1D

Part 1D of this study looks at the Quantum Shammas Polynomials with a special varying pattern for the polynomial powers.

Document History

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

