Quantum Shammas Polynomials
Part 1B of the Study

By
Namir Shammas

Contents

Introduction

Part 1 introduced you to Quantum Shammas Polynomials. In this part we look at the first variant of these polynomials. The variation concerns the ranges of random powers from which random values 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, $0.5 <= r_1 <= 2.4$, $2.5 <= r_2 <= 4.4$, …, and $2*(n-1)+0.5 <= r_n < 2*(n-1)+2.4$.. 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 wider ranges* (compared to the ranges used in Part 1) 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 (<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 and a modified version that has even powers. The adjusted coefficient of determinations are good indicators of how the three 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, a fourth order classical polynomial, and a fourth order even-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("\nEven-powered polynomial fit\n");
c2 = polyfit(xData.^2),yData,order)
yPoly2 = polyval(c2,xData.^2));
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);
EvenCoeff = 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(EvenCoeff);
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 (in file besselj_0_x_run1.xlsx), shown below:

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	
1.994544474	4.331090584	5.018678574	7.503351907	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
1.00000838	-0.248245217	0.018213191	-0.004763794	5.24139E-06
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.980927341	0.138170634	-0.457980428	0.113695746	-0.007357698
EvenCoeff1	EvenCoeff2	EvenCoeff3	EvenCoeff4	EvenCoeff5
0.999419662	-0.248661584	0.015150603	-0.000374942	3.66132E-06
r_sqr1	r_sqr2	r_sqr3		

0.999999997	0.999803041	0.999998869		

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 rest of the coefficients for the 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 even-powered 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 cell under r_sqr3 shows the adjusted coefficient of determination for the fitted even-powered classical polynomial. The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is greater than the one for the classical polynomial and greater than the one for the even-power classical polynomial. The latter has an adjusted coefficient of determination that is greater than the classical polynomial.

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

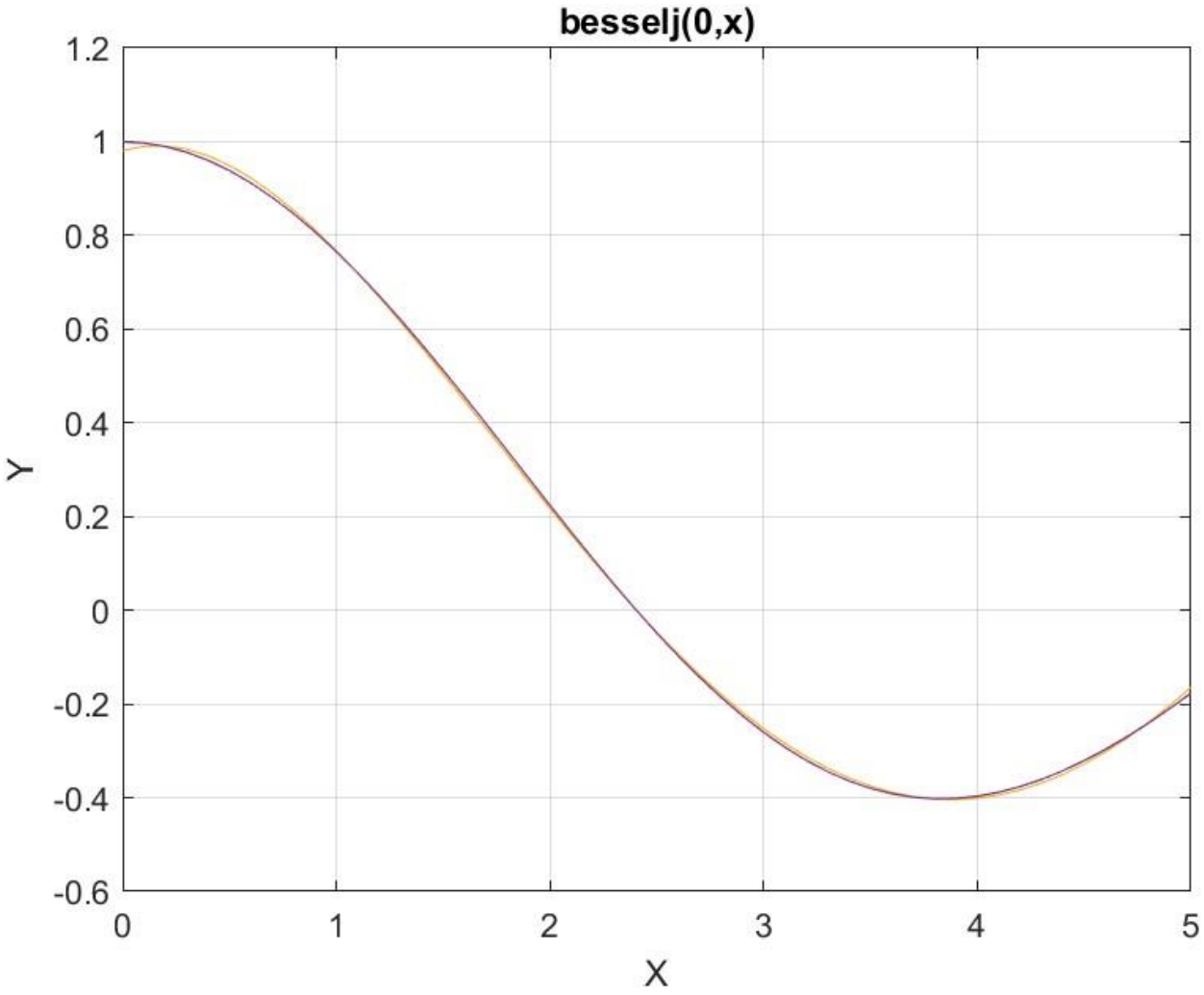

Figure 1. The graph from file besselj_0_x_run1.jpg.

The above graph shows a fairly good fit for all three polynomials.

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, a fourth order classical polynomial, and a fourth order even-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, 2.4, 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("\nEven-powered polynomial fit\n");
c2 = polyfit(xData.^2,yData,order)
yPoly2 = polyval(c2,xData.^2);
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);
EvenCoeff = 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(EvenCoeff);
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	
1.994505065	4.219325687	5.39589876	6.907180638	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
1.000028922	-0.248450872	0.015442805	-0.001842506	3.16611E-05
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.980927341	0.138170634	-0.457980428	0.113695746	-0.007357698
EvenCoeff1	EvenCoeff2	EvenCoeff3	EvenCoeff4	EvenCoeff5
0.999419662	-0.248661584	0.015150603	-0.000374942	3.66132E-06
r_sqr1	r_sqr2	r_sqr3		
0.999999998	0.999803041	0.999998869		

Table 2. 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 12. Again, the adjusted coefficient of determination for the Quantum Shammas Polynomial is higher than that for the two versions of classical polynomials. All three polynomials give good values. Again, the even-powered classical polynomial does better than the regular classical polynomial.

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

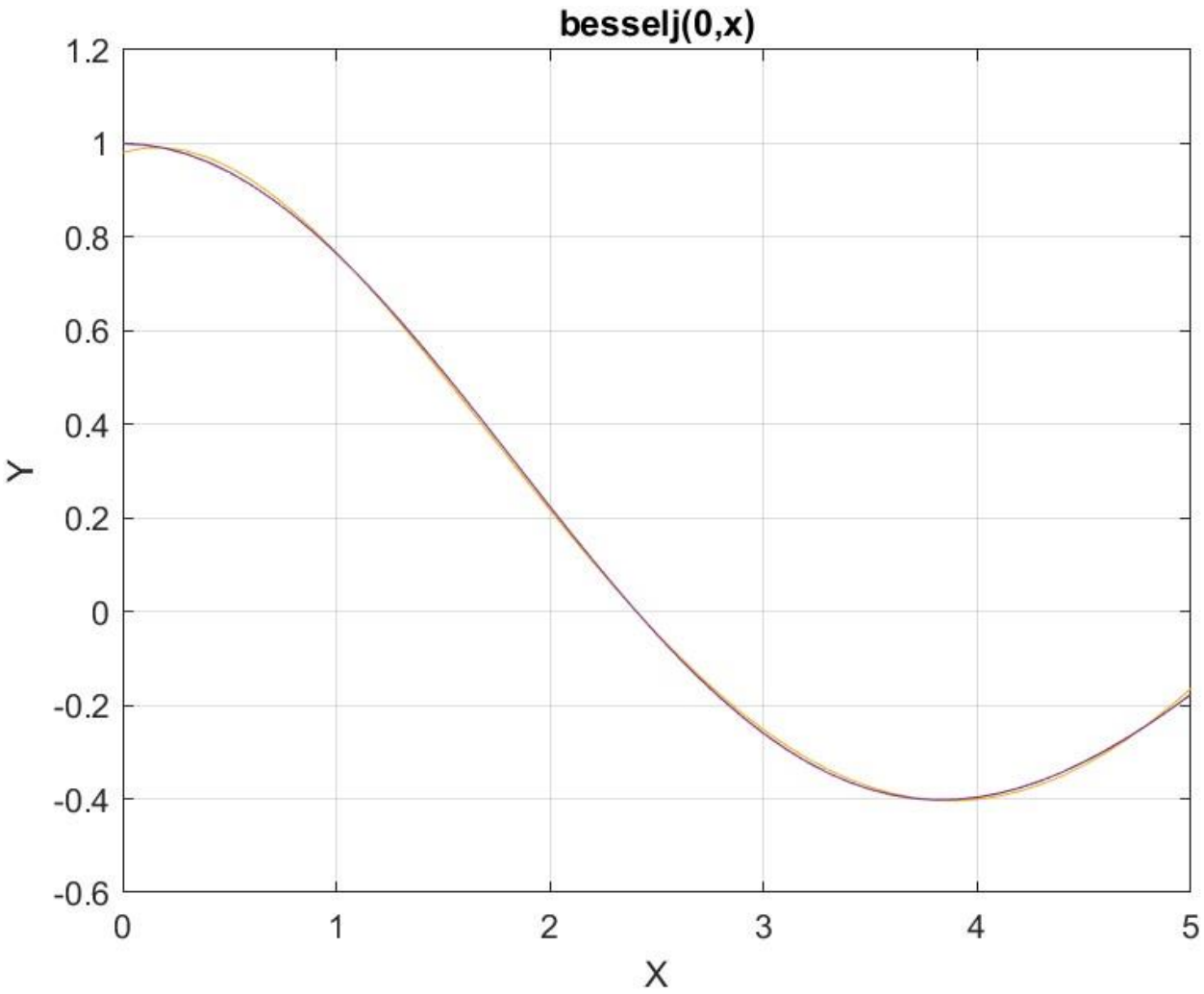

Figure 2. The graph from file besselj_0_x_random_run1.jpg.

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

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, a fourth order classical polynomial, and a fourth order even-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, 2.4, 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("\nEven-powered polynomial fit\n");
c2 = polyfit(xData.^2,yData,order)
yPoly2 = polyval(c2,xData.^2);
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);
EvenCoeff = 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(EvenCoeff);
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 the 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	
1.99330203	4.225663932	5.453714091	6.773637421	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
1.000047064	-0.248306698	0.015170004	-0.001752126	4.96531E-05
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.980927341	0.138170634	-0.457980428	0.113695746	-0.007357698
EvenCoeff1	EvenCoeff2	EvenCoeff3	EvenCoeff4	EvenCoeff5
0.999419662	-0.248661584	0.015150603	-0.000374942	3.66132E-06
r_sqr1	r_sqr2	r_sqr3		
0.99999998	0.999803041	0.999998869		

Table 3. 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 two classical polynomials. Using the Halton 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_halton_random_run1.jpg) for the Bessel function and the two fitted polynomials:

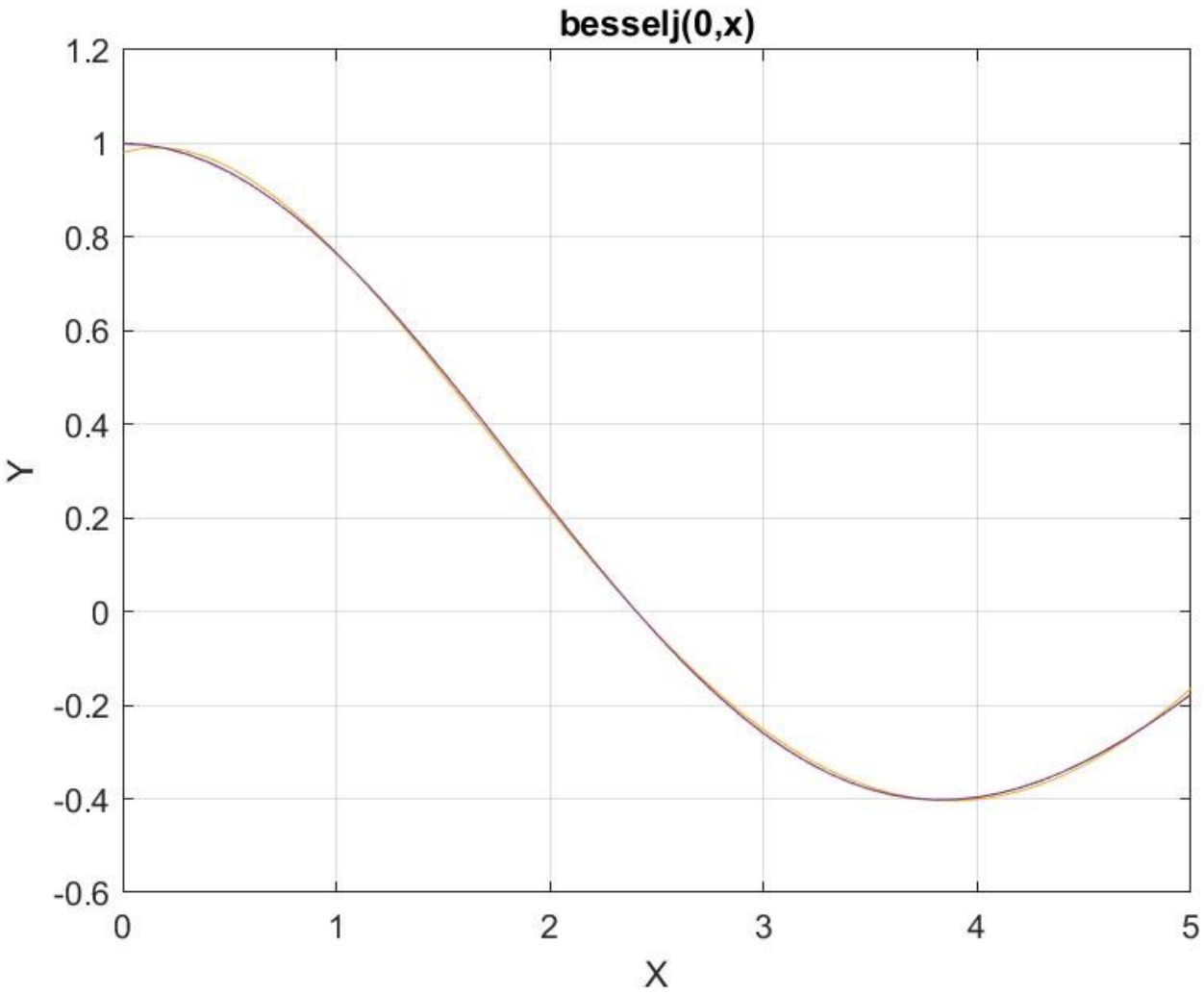

Figure 3. The graph from file besselj_0_x_halton_random_run1.jpg.

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

Testing Bessel Function Fit with 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, a fourth order classical polynomial, and a fourth order even-power 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, 2.4, 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("\nEven-powered polynomial fit\n");
c2 = polyfit(xData.^2,yData,order)
yPoly2 = polyval(c2,xData.^2);
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);
EvenCoeff = 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(EvenCoeff);
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 the 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	
1.990751129	4.421658167	4.988071618	7.149007487	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
1.000067565	-0.247692404	0.019463478	-0.006640473	1.3105E-05
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.980927341	0.138170634	-0.457980428	0.113695746	-0.007357698
EvenCoeff1	EvenCoeff2	EvenCoeff3	EvenCoeff4	EvenCoeff5
0.999419662	-0.248661584	0.015150603	-0.000374942	3.66132E-06
r_sqr1	r_sqr2	r_sqr3		
0.999999998	0.999803041	0.999998869		

Table 4. 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 two classical polynomials. All coefficients of determination 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:

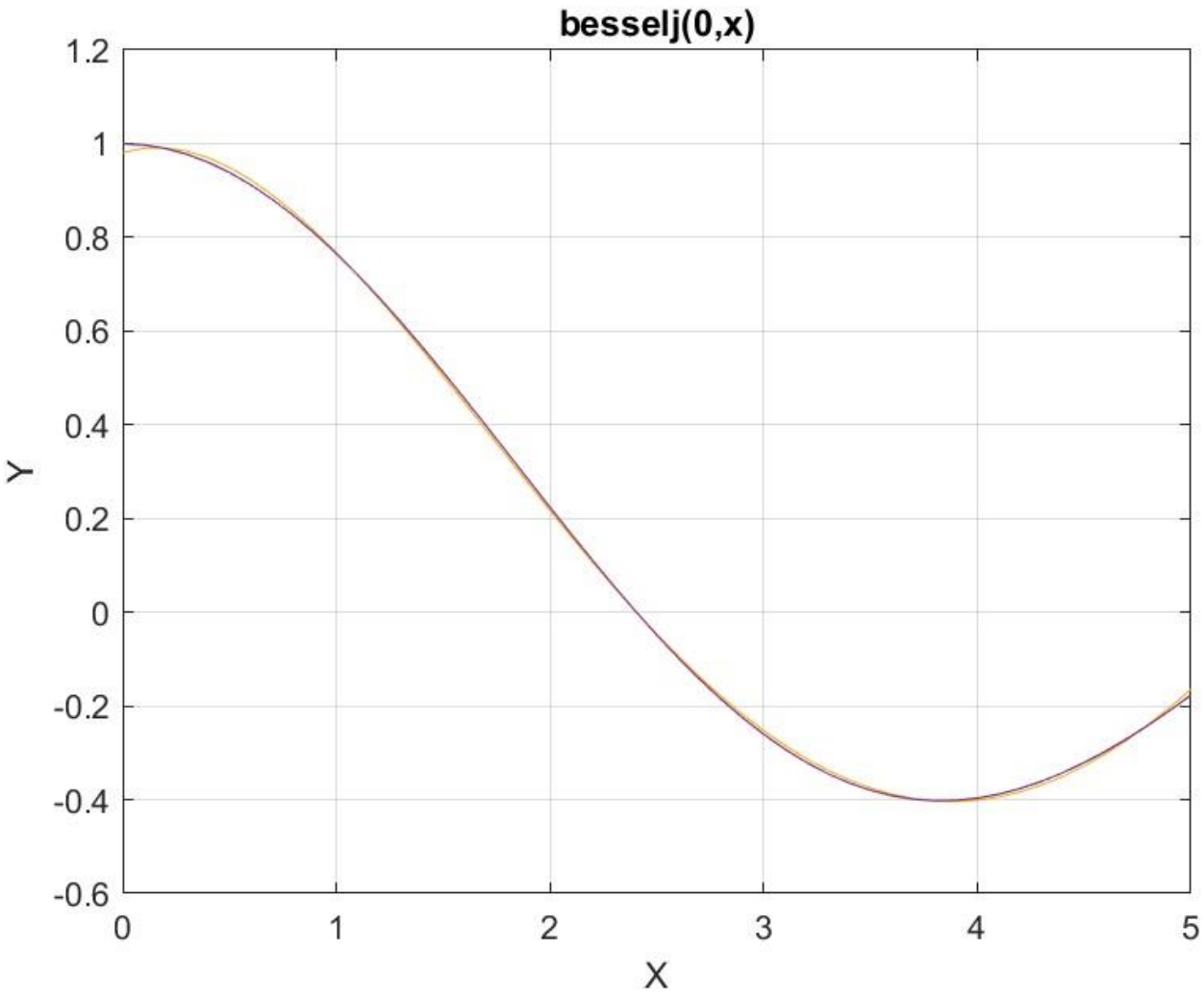

Figure 4. The graph from file besselj_0_x_sobol_random_run1.jpg.

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

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 results also show that the even-powered classical polynomials came in second!

The next four sections 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, a fourth order classical polynomial, and a fourth order even-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, 2.4, 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("\nEven-powered polynomial fit\n");
c2 = polyfit(xData.^2,yData,order)
yPoly2 = polyval(c2,xData.^2);
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);
EvenCoeff = 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(EvenCoeff);
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	
0.745834664	2.501233863	5.002344719	7.504090422	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
-0.019570664	0.72188514	-0.016888166	0.00015754	-7.6181E-07
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.019526836	0.850579688	-0.210226108	0.031956872	-0.001944825
EvenCoeff1	EvenCoeff2	EvenCoeff3	EvenCoeff4	EvenCoeff5
0.33973614	0.25131791	-0.019660294	0.000684647	-8.33099E-06
r_sqr1	r_sqr2	r_sqr3		
0.999880585	0.99989954	0.972909381		

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

The adjusted coefficient of determination for the Quantum Shammas Polynomial is less (by the proverbial hair) than the one for classical polynomial. I consider the difference as statistically insignificant. The even-power classical polynomial did not do as well in fitting function ln(x).

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

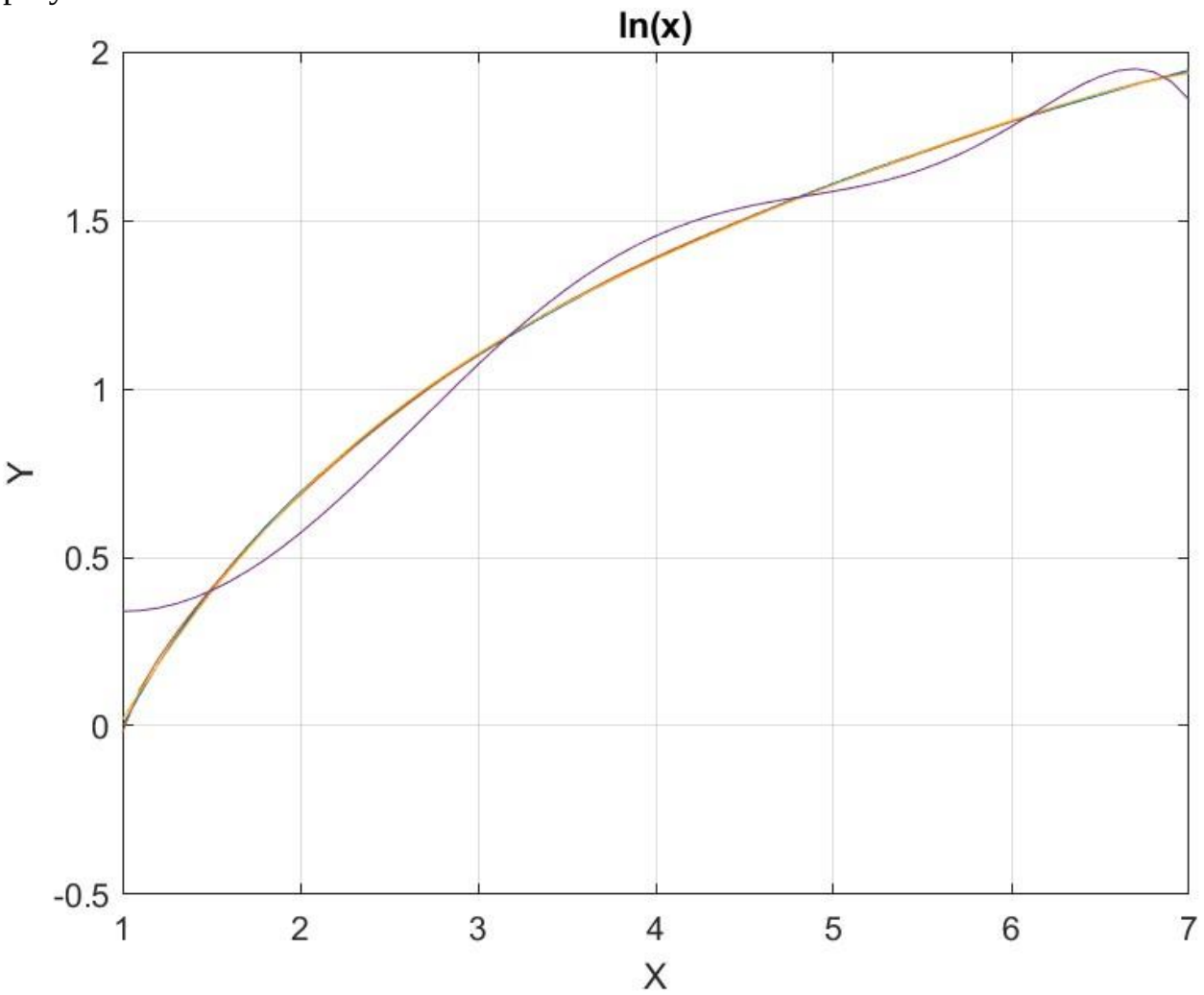

Figure 5. The graph from file ln_x_pso.jpg.

The above graph shows that the Quantum Shammas Polynomial and regular classical polynomial fit the ln(x) function well. The graph shows the deviations of the even-power classical polynomial.

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, a fourth order classical polynomial, and a fourth order even-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, 2.4, 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("\nEven-powered polynomial fit\n");
c2 = polyfit(xData.^2,yData,order)
yPoly2 = polyval(c2,xData.^2);
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);
EvenCoeff = 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(EvenCoeff);
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	
0.773954207	2.285120288	4.521209715	6.821468395	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
-0.014444139	0.729371619	-0.028786446	0.000386711	-2.52221E-06
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.019526836	0.850579688	-0.210226108	0.031956872	-0.001944825
EvenCoeff1	EvenCoeff2	EvenCoeff3	EvenCoeff4	EvenCoeff5
0.33973614	0.25131791	-0.019660294	0.000684647	-8.33099E-06
r_sqr1	r_sqr2	r_sqr3		
0.999918871	0.99989954	0.972909381		

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

The adjusted coefficient of determination for the Quantum Shammas Polynomial is higher (by a hair) 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! The even-power classical polynomial did not fit function ln(x) as well.

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

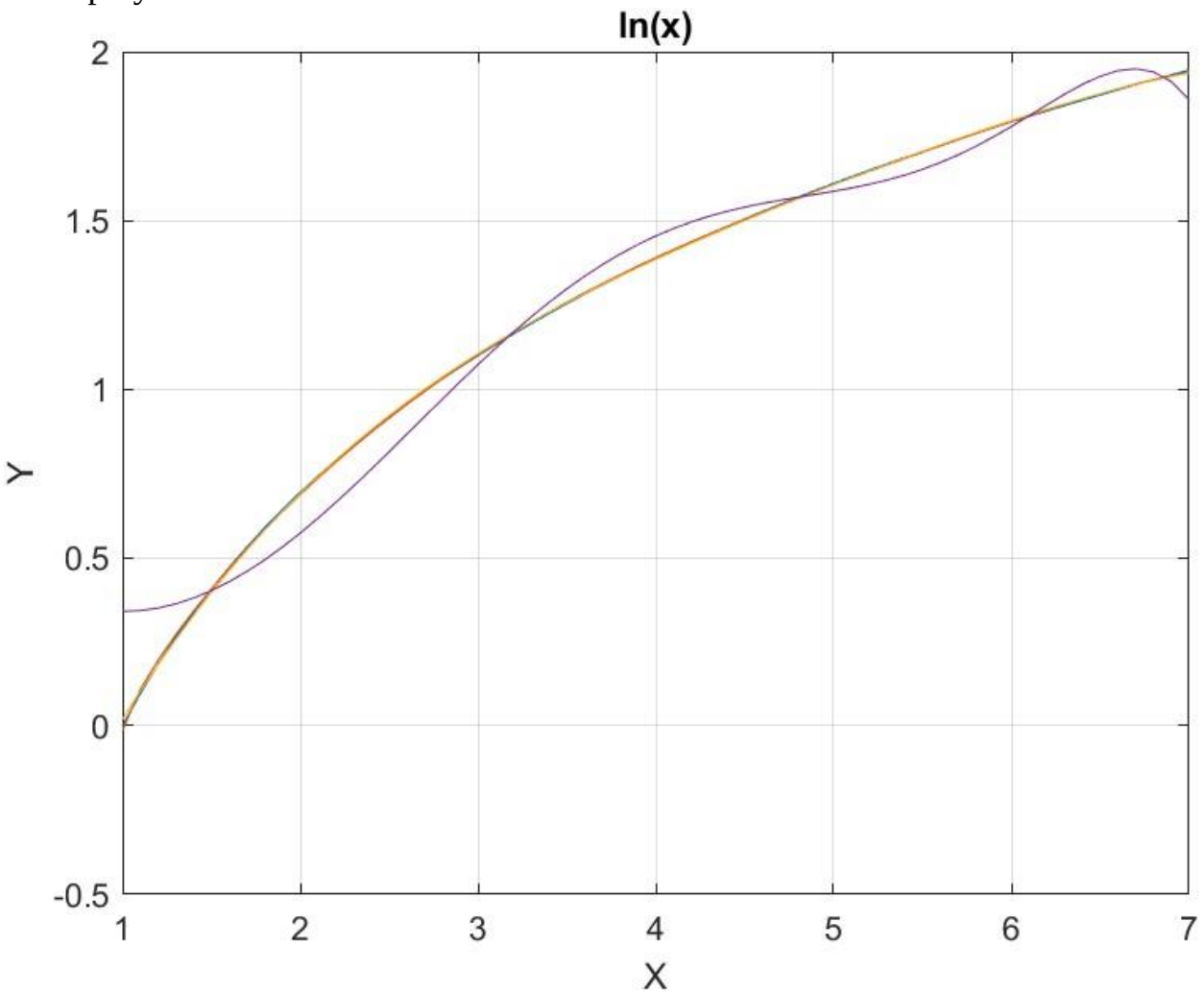

Figure 6. The graph from file ln_x_rand.jpg

The above graph affirms the same conclusion as in Figure (5).

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, a fourth order classical polynomial, and a fourth order even-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, 2.4, 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("\nEven-powered polynomial fit\n");
c2 = polyfit(xData.^2,yData,order)
yPoly2 = polyval(c2,xData.^2);
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);
EvenCoeff = 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(EvenCoeff);
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	
0.776330999	2.258032053	4.53342178	6.850252653	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
-0.01398715	0.730257569	-0.030098002	0.000368495	-2.34947E-06
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.019526836	0.850579688	-0.210226108	0.031956872	-0.001944825
EvenCoeff1	EvenCoeff2	EvenCoeff3	EvenCoeff4	EvenCoeff5
0.33973614	0.25131791	-0.019660294	0.000684647	-8.33099E-06
r_sqr1	r_sqr2	r_sqr3		
0.999920284	0.99989954	0.972909381		

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

Table 11 affirms the same conclusion as Table 6! The Halton sequence did surprisingly well, given it 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:

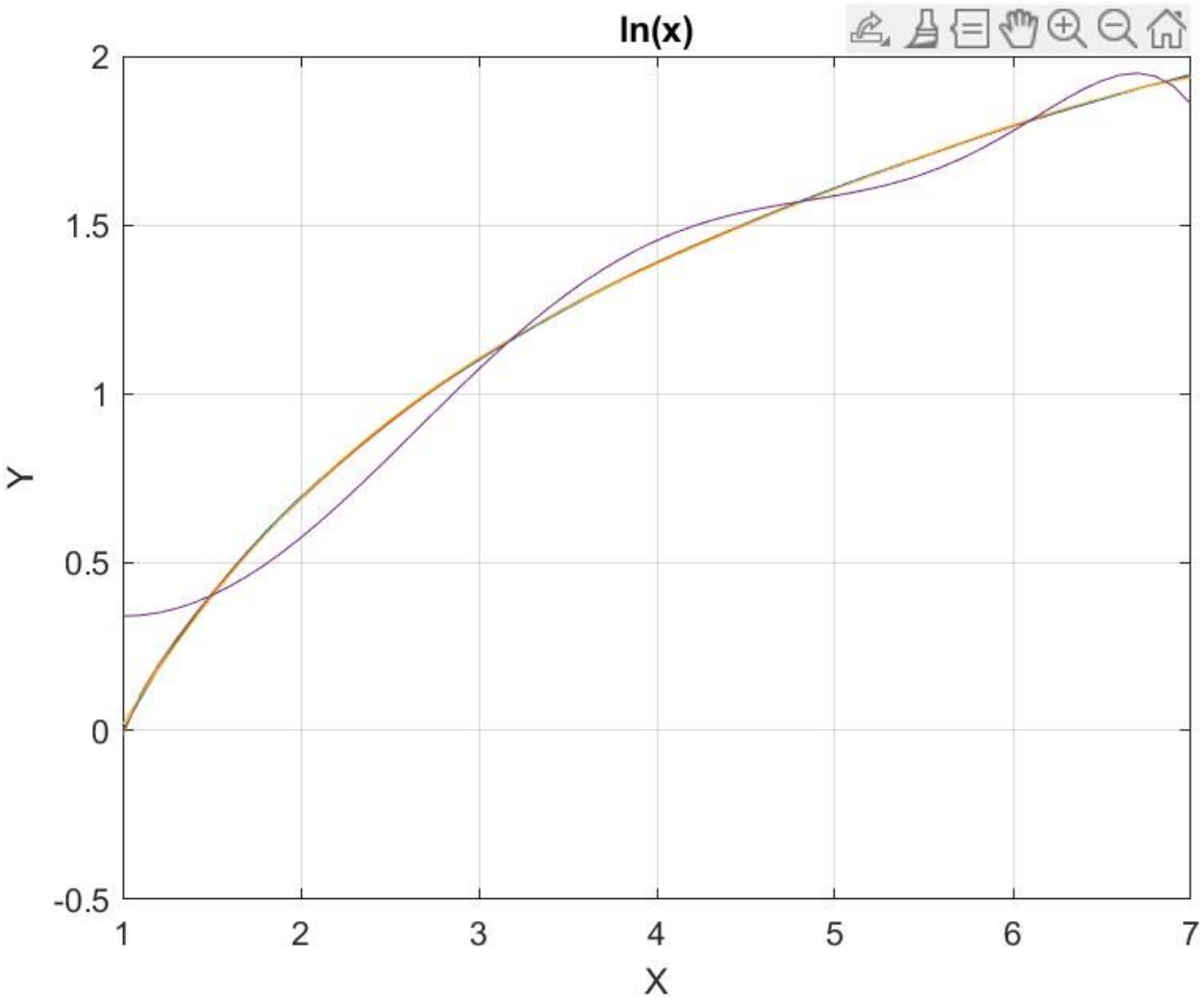

Figure 7. The graph from file ln_x_halton_rand.jpg

The above figure affirms the same conclusions as the last two figures.

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, a fourth order classical polynomial, and a fourth order even-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, 2.4, 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("\nEven-powered polynomial fit\n");
c2 = polyfit(xData.^2,yData,order)
yPoly2 = polyval(c2,xData.^2);
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);
EvenCoeff = 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(EvenCoeff);
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	
0.791245773	2.145603724	4.358938921	6.617003582	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
-0.01189454	0.73758437	-0.03912237	0.00049387	-3.42147E-06
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.019526836	0.850579688	-0.210226108	0.031956872	-0.001944825
EvenCoeff1	EvenCoeff2	EvenCoeff3	EvenCoeff4	EvenCoeff5
0.33973614	0.25131791	-0.019660294	0.000684647	-8.33099E-06
r_sqr1	r_sqr2	r_sqr3		
0.99993437	0.99989954	0.972909381		

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

Table 8 affirms the same conclusions as Table 6 and Table 7. The Sobol sequence did well, given that it 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:

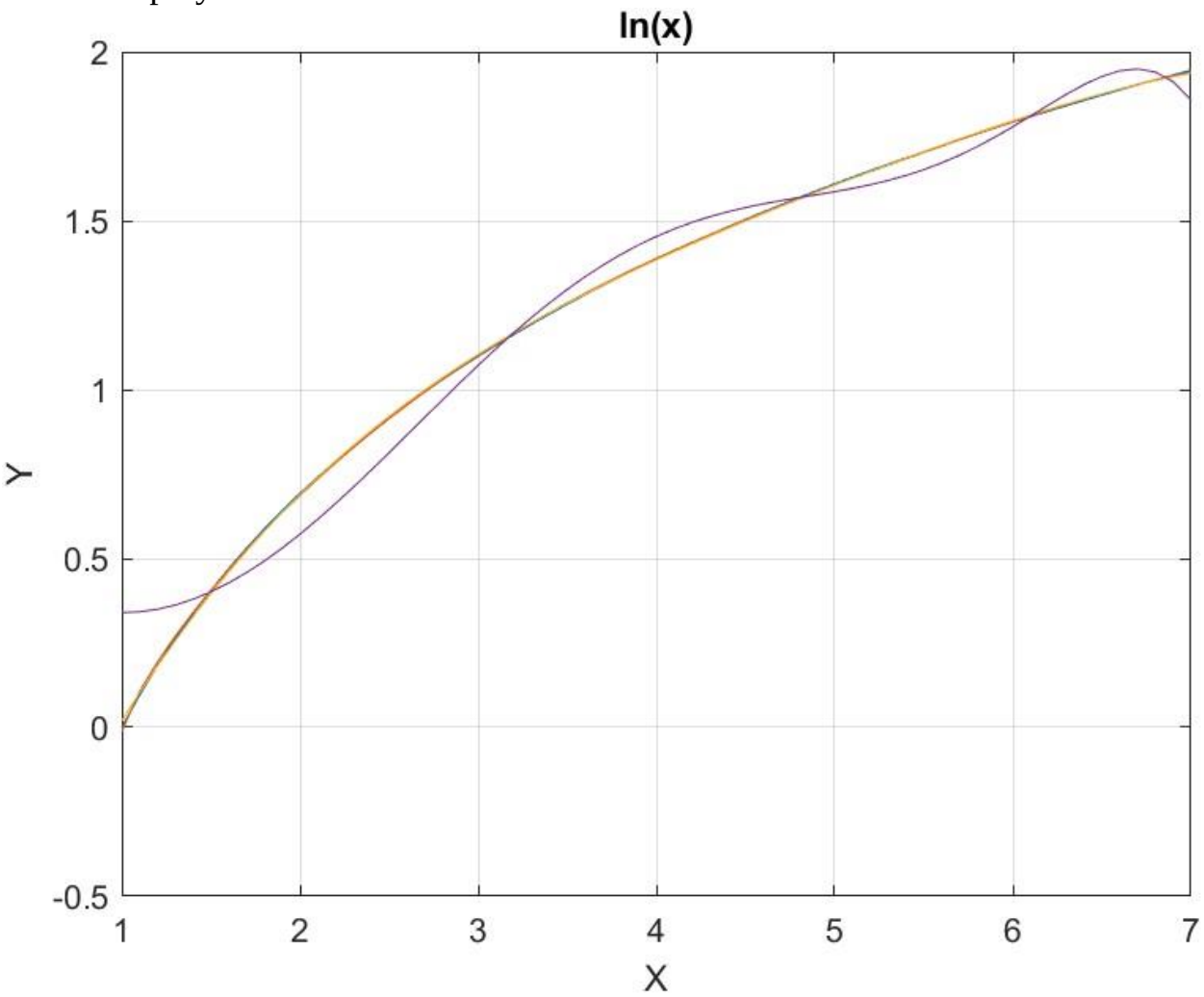

Figure 8. The graph from file ln_x_sobol_rand.jpg

The above figure affirms the same conclusions as the last three figures.

Conclusion for fitting the ln(x) Function

The above four sections show that fitting the ln(x) vs (x-1) for the range of (1, 7) using the Quantum Shammas Polynomial comes to a statistical dead heat with the regular classical polynomial. The even-power classical polynomials did not do as well as the other two types of polynomials.

The next four sections in Part 1B 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, a fourth order classical polynomial, and a fourth order even-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, 2.4, 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("\nEven-powered polynomial fit\n");
```



```
c2 = polyfit(xData.^2,yData,order)
yPoly2 = polyval(c2,xData.^2);
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);
EvenCoeff = 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(EvenCoeff);
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 normal Gaussian function. The above code generates the following Excel table.

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	
2.165087859	3.182042404	5.001547202	7.501003945	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.398687043	-0.26499591	0.114833275	-0.00663723	7.74862E-05
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.397644494	0.028633101	-0.306018517	0.139989216	-0.018592075
EvenCoeff1	EvenCoeff2	EvenCoeff3	EvenCoeff4	EvenCoeff5
0.397521371	-0.189521842	0.039032933	-0.003882461	0.000149844
r_sqr1	r_sqr2	r_sqr3		
0.999998688	0.999967249	0.999922899		

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

The adjusted coefficient of determination for the Quantum Shammas Polynomial is higher than the ones for classical polynomials. Since the PSO method uses random numbers, I consider the difference between the three 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:

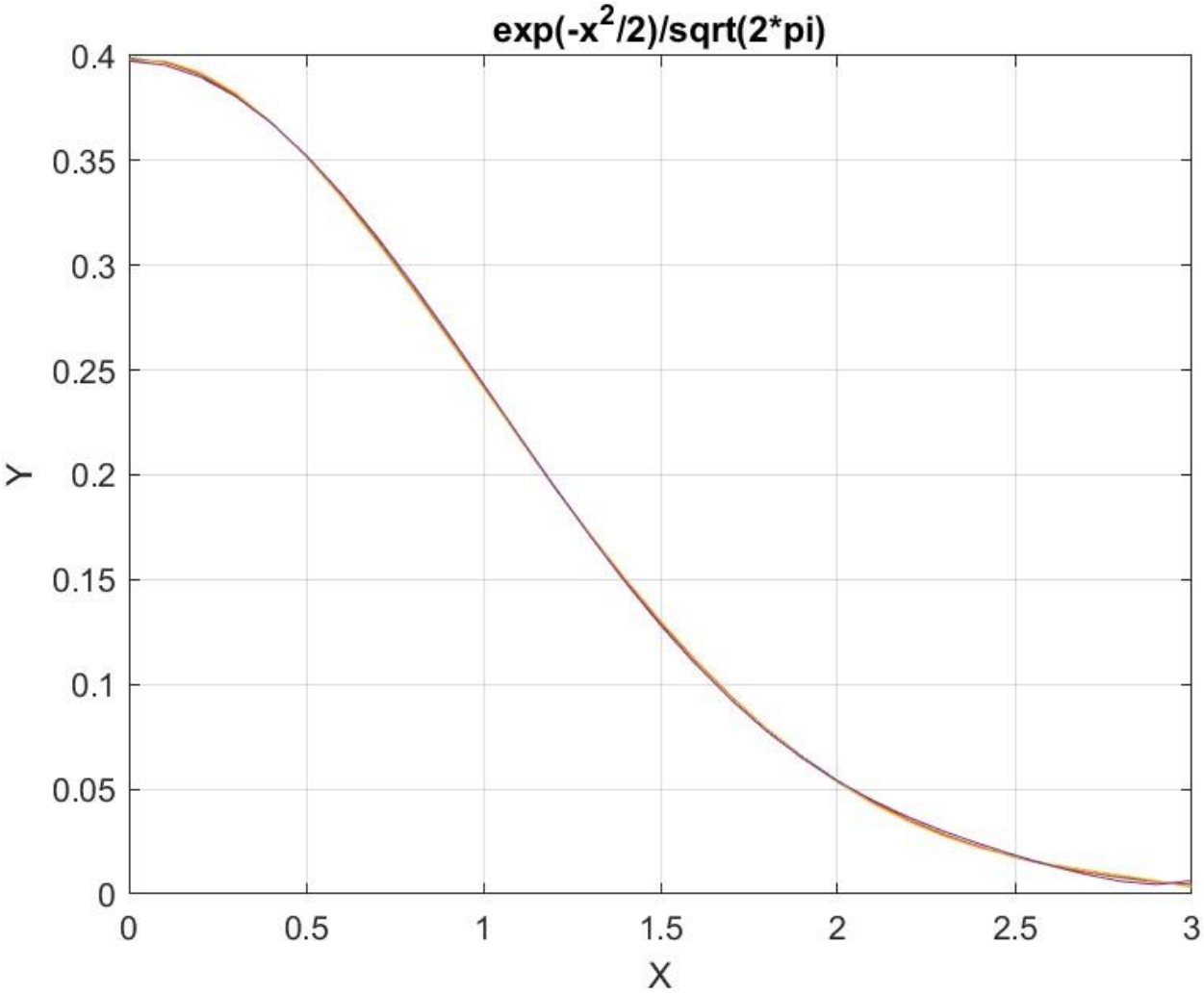

Figure 9. The graph from file Right_GaussBell_x.jpg.

The above graph shows that the three 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, a fourth order classical polynomial, and a fourth order even-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, 2.4, 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("\nEven-powered polynomial fit\n");
c2 = polyfit(xData.^2,yData,order)
yPoly2 = polyval(c2,xData.^2);
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);
EvenCoeff = 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(EvenCoeff);
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	
2.112351593	3.509423791	4.713365635	6.791249339	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.398730152	-0.234915445	0.094421104	-0.016565111	0.000272709
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.397644494	0.028633101	-0.306018517	0.139989216	-0.018592075
EvenCoeff1	EvenCoeff2	EvenCoeff3	EvenCoeff4	EvenCoeff5
0.397521371	-0.189521842	0.039032933	-0.003882461	0.000149844
r_sqr1	r_sqr2	r_sqr3		
0.999999112	0.999967249	0.999922899		

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

The adjusted coefficient of determination for the Quantum Shammas Polynomial is well higher (six 9s after the decimal compared to four 9s after the decimal) than the ones for classical polynomials.

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

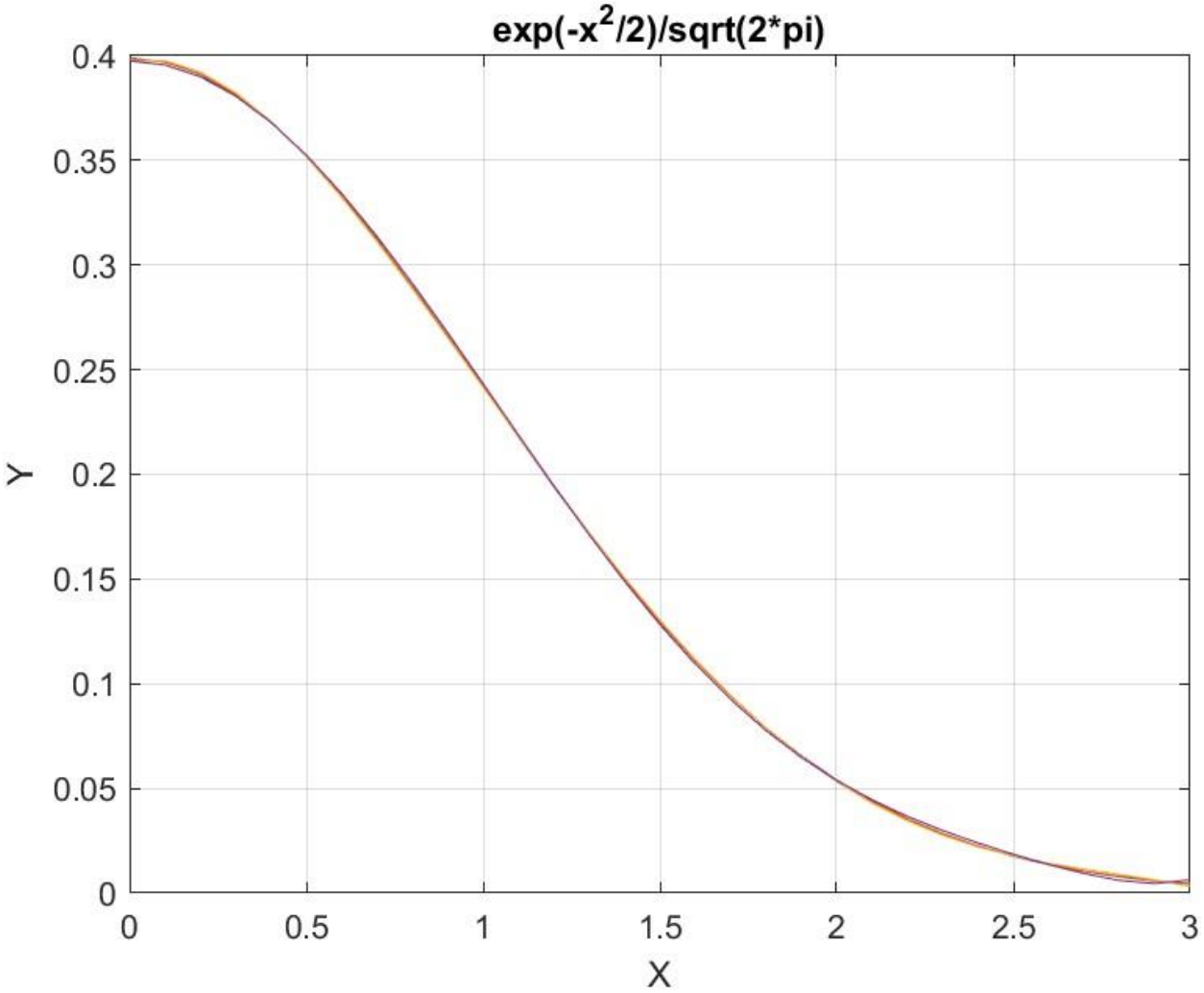

Figure 10. The graph from file Right_GaussBell_x_random.jpg.

The above graph shows that the three 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 testGauss1Random.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, a fourth order classical polynomial, and a fourth order even-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, 2.4, 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("\nEven-powered polynomial fit\n");
c2 = polyfit(xData.^2,yData,order)
yPoly2 = polyval(c2,xData.^2);
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);
EvenCoeff = 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(EvenCoeff);
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	
2.115559769	3.484135059	4.745740322	6.810206792	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.39872152	-0.236331536	0.094495006	-0.015206289	0.000266206
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.397644494	0.028633101	-0.306018517	0.139989216	-0.018592075
EvenCoeff1	EvenCoeff2	EvenCoeff3	EvenCoeff4	EvenCoeff5
0.397521371	-0.189521842	0.039032933	-0.003882461	0.000149844
r_sqr1	r_sqr2	r_sqr3		
0.999999096	0.999967249	0.999922899		

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

The above table echoes the comments made for Table 10.

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

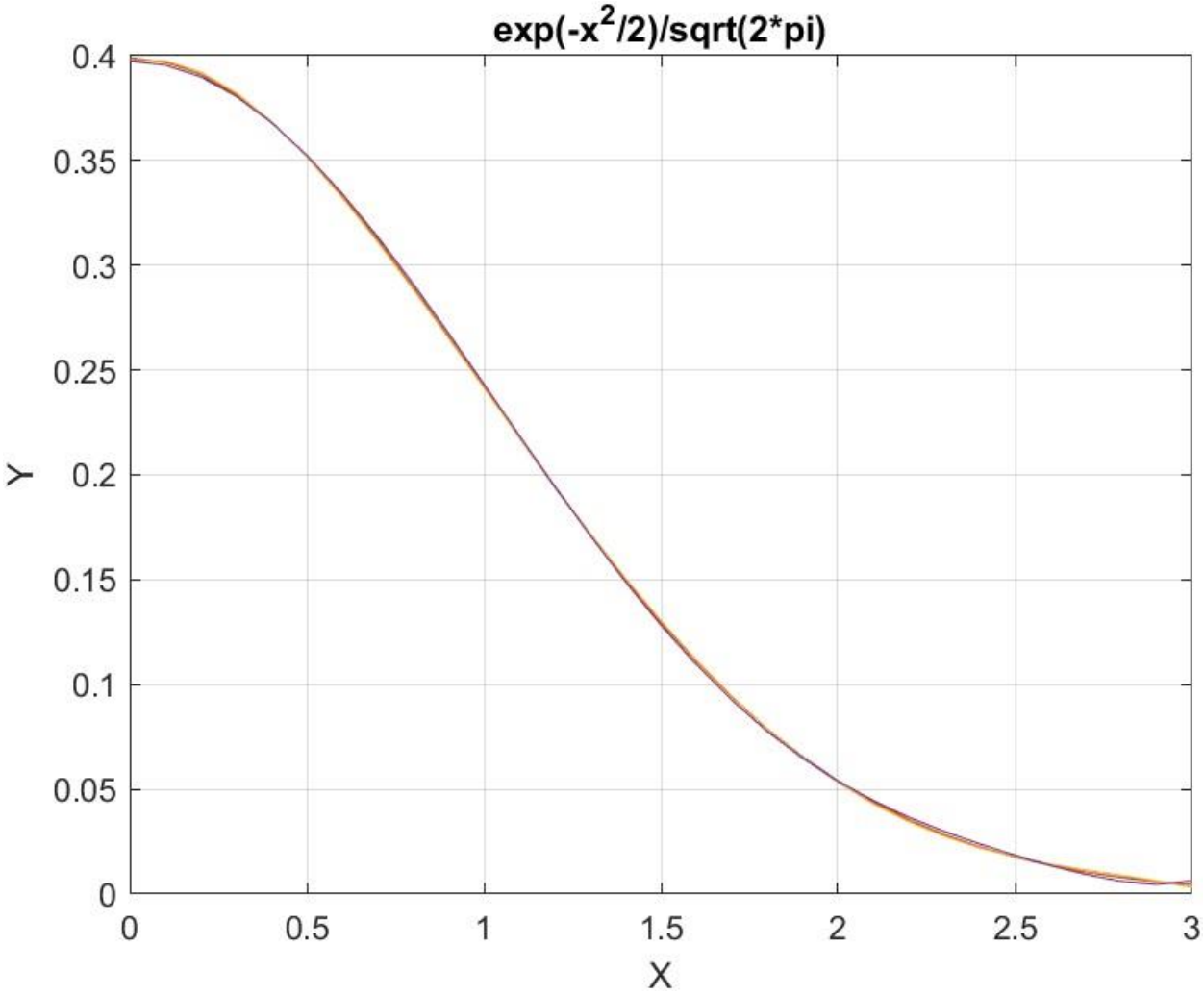

Figure 11. The graph from file Right_GaussBell_x_halton_random.jpg.

The above graph shows that the three 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 testGauss1Halton.m) tests fitting normal(1, 0) for x in the range (0, 3) and samples at 0.1 steps, and uses the Sobol quasi-random search optimization. The curve fits use a fourth order Quantum Shammas Polynomial, a fourth order classical polynomial, and a fourth order even-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, 2.4, 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("\nEven-powered polynomial fit\n");
c2 = polyfit(xData.^2,yData,order)
yPoly2 = polyval(c2,xData.^2);
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);
EvenCoeff = 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(EvenCoeff);
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	
2.127787468	3.417959072	4.805105456	6.783167285	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.398705348	-0.241728136	0.097666048	-0.012992134	0.000280937
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.397644494	0.028633101	-0.306018517	0.139989216	-0.018592075
EvenCoeff1	EvenCoeff2	EvenCoeff3	EvenCoeff4	EvenCoeff5
0.397521371	-0.189521842	0.039032933	-0.003882461	0.000149844
r_sqr1	r_sqr2	r_sqr3		
0.999999054	0.999967249	0.999922899		

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

The above table echoes the same conclusions as Tables 10 and 11.

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

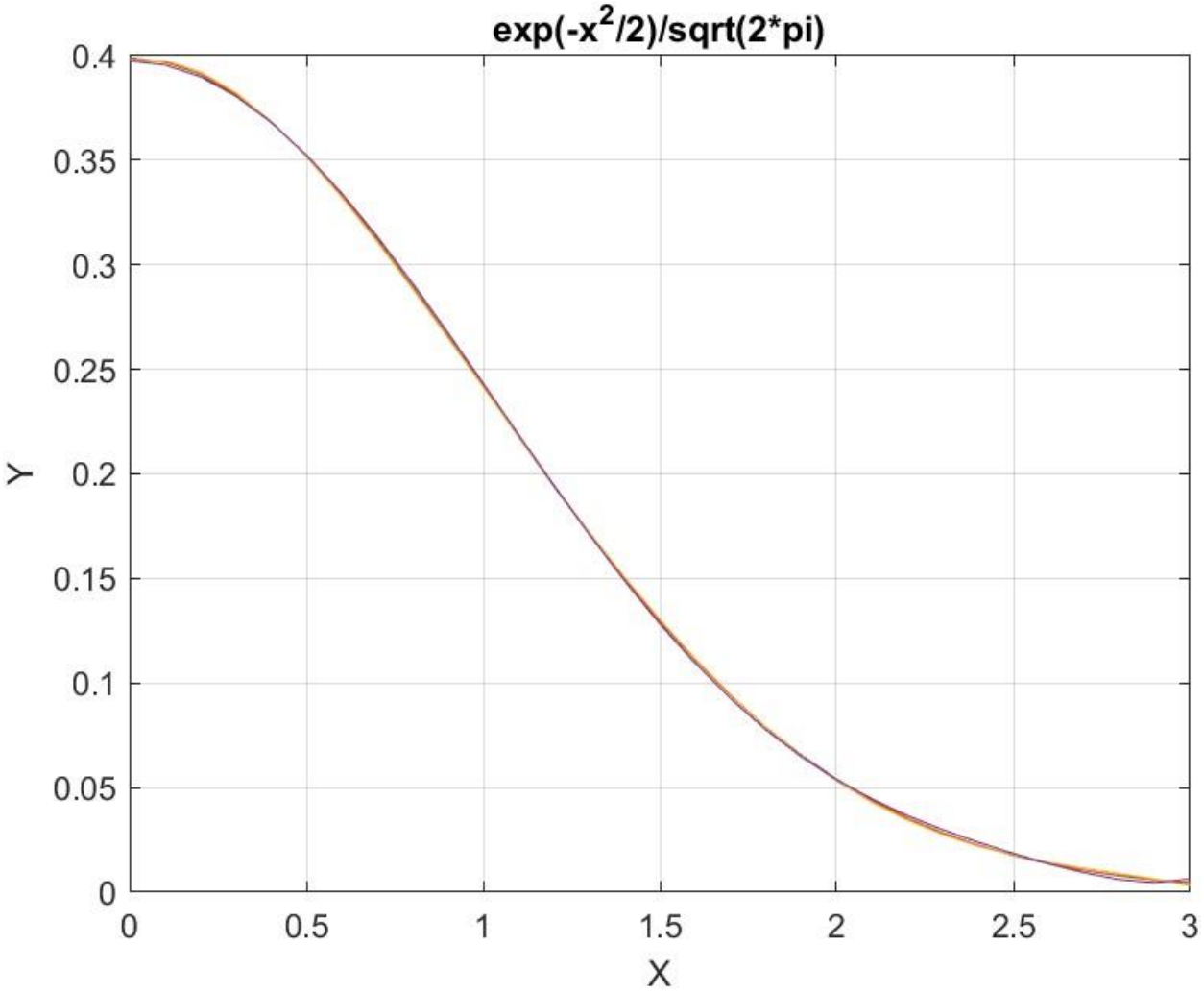

Figure 12. The graph from file Right_GaussBell_x_sobol_random.jpg.

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

Conclusion for fitting the Right-Side Normal Gaussian Function

The above four sections 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 1B

The Quantum Shammas Polynomials did well in fitting the sample test cases.

Next is Part 1C

Part 1C of this study looks at the Quantum Shammas Polynomials with smaller ranges of powers, than in Part 1, for the same test cases presented in this part.

Document History

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

