Quantum Shammas Polynomials
Part 1

By
Namir Shammas

Math is the Chemistry of Numbers--NS

Contents

Core of this Study

This study is about a new category of polynomials, Padé polynomials, and Fourier series. Their aim is to perform better least-squares curve fitting than classical polynomials, classical Padé polynomials, and classical Fourier series, respectively. Therefore, the purpose of these families of new functions is mainly statistical rather than mathematical.

Map of the Study's Documents

This study spans over the following multiple documents, labeled as ***parts***:

- Part 1 is this document. It covers the basic Quantum Shammas Polynomials. The source code and output files for this part reside in the ZIP file qsp1.zip.
- Part 1B covers the first variant of the Quantum Shammas Polynomials. The source code and output files for this part reside in the ZIP file qsp1b.zip.
- Part 1C covers the second variant of the Quantum Shammas Polynomials. The source code and output files for this part reside in the ZIP file qsp1c.zip.
- Part 1D covers the third variant of the Quantum Shammas Polynomials. The source code and output files for this part reside in the ZIP file qsp1d.zip.
- Part 1E covers the fourth variant of the Quantum Shammas Polynomials. The source code and output files for this part reside in the ZIP file qsp1e.zip.
- Part 2 covers the Quantum Shammas Padé Polynomials. The source code and output files for this part reside in the ZIP file qsp2.zip.
- Part 3 covers the Quantum Shammas Fourier Series. The source code and output files for this part reside in the ZIP file qsp3.zip.

The difference between the topics in Parts 1, 1B, 1C, 1D, and 1E concerns the operational parameters of the Quantum Shammas Polynomials.

You can download the ZIP files mentioned above from the web page at www.namirshammas.com/NEW/qsp.html. **Keep the files extracted from each ZIP file in separate folders, since they have files that share similar filenames.**

Introduction

Quantum Shammas Polynomials are inspired by how quantum physics views the probabilistic orbits of the electrons in an atom. These ***non-orthogonal*** polynomials have nothing to do with the new art of quantum computing per se. Early on, scientists assumed that the electrons in an atom had distinct orbits that were thought to be fixed. This concept parallels the fixed powers of classical polynomials. By contrast, the Heisenberg uncertainty principle suggests that the orbits of the electrons are more probabilistic than fixed. This is the inspiration for Quantum Shammas Polynomials. While classical polynomials have the familiar fixed integer powers, shown next:

$$y(x) = a_0 + a_1*x + a_2*x^2 + \ldots + a_n*x^n \qquad (1)$$

The ***non-orthogonal*** Quantum Shammas Polynomials have random powers that typically vary around integer powers. For examples they can use ranges between (i – 1) + 0.5 to i + 0.4 where i is the term number. The general form of the Quantum Shammas Polynomial is:

$$y(x) = a_0 + a_1*x^{r1} + a_2*x^{r2} + \ldots + a_n*x^{rn} \quad \text{for } x>=0 \qquad (2)$$

Where $0.5 <= r_1 <= 1.4$, $1.5 <= r_2 <= 2.4$, …, and $(n-1)+0.5 <= r_n < n+0.4$. 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. I chose the above ranges for the random powers r_i as arbitrary values (a kind of starting point or first run, if you will). The subsequent parts of this study show you how to use different schemes to calculate different ranges. In all cases, the values of the random powers (r_i) are chosen to minimize the sum of errors squared between some observed values of y(x) (this study uses mathematical functions to generate these values of y(x)) and the ones calculated using equation (2). 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 study also looks at Padé and Fourier versions of the Quantum Shammas Polynomials. A Quantum Shammas Padé Polynomials looks like:

$$y(x) = (a_0 + a_1*x^{r1} + a_2*x^{r2} + \ldots + a_{np}*x^{rp}) \;/\; (1 + b_1*x^{s1} + b_2*x^{s2} + \ldots + b_q*x^{sq}) \quad \text{for } x>=0 \qquad (3)$$

Where the values for r_i and s_i follow the ranges of values that I discussed above.

The study also looks at Fourier Quantum Shammas Series. They have the following general equation:

$$y(x) = a_0 + a*\sin(s_1* \pi *x) + b_1*\cos(c_1* \pi *x) + \ldots + a_n*\sin(s_n * \pi *x) + b_n*\cos(c_n* \pi *x) \quad (4)$$

Where the values for s_i and c_i follow the ranges of values that I discussed above.

My goal is to see that the Quantum Shammas Polynomials/Series provide a better fit that using classical polynomials. My second goal is to see that Quantum Shammas Polynomials/Series provide fits that do not fall far behind those of classical polynomials.

☛ The next sections (in this and subsequent parts) show you the listing for the numerous MATLAB files. I am including these files so that this document can be as self-sufficient as possible. Each set of source code files and output files comes in a separate ZIP file.

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. Why use global variables? This approach makes it easy for the function to be called by the particle swarm optimization which needs to return the value of the optimized function only.

The Quantum Padé Shammas Polynomial Function

The Quantum Shammas Padé Polynomial function in MATLAB is:

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

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

```
  yCalc = zeros(n,1);
  for i=1:n
    sumP = QSPcoeff(1);
    for j=1:orderP
      sumP = sumP + QSPcoeff(j+1)*xData(i)^pwr(j);
    end
    sumQ = 1;
    for j=1:orderQ
      k = orderP + j;
      sumQ = sumQ - QSPcoeff(k+1)*yData(i)*xData(i)^pwr(k);
    end
    yCalc(i) = sumP / sumQ;
    SSE = SSE + (yCalc(i) - yData(i))^2;
  end
  glbRsqr = 1 - SSE / SStot;
end
```

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

The Quantum Shammas Fourier Series Function

The Quantum Shammas Fourier Series function in MATLAB is:

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

```
    end
    SSE = SSE + (yCalc(i) - yData(i))^2;
  end
  glbRsqr = 1 - SSE / SStot;
end
```

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

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 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 elements of matrix rando as pseudo-random numbers are needed.

☛ Using the Halton and Sobol quasi-random sets in MATALB establishes a matrix of pseudo-random numbers. You can reuse the same matrix but wish to have a different sequence of pseudo-random numbers. To do this, you use the MATLAB function randperm() to generate an array of random row numbers. You rearrange the matrix rows using this array of random row numbers. The result is a modified matrix with the same values but ordered in a different random sequence.

Testing Quantum Shammas Polynomials

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

Testing Bessel Function Fit with PSO-Run1

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

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_run1";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "besselJ(0, x)";
fprintf(sEqn);
fprintf("x=0:0.1:5\n")
xData= 0:0.1:5;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.4);

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

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

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

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + 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	
1.146890335	2.397874445	3.396232875	4.399816833	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
1.001049628	-0.041962343	-0.269894175	0.083187401	-0.006548292
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.980927341	0.138170634	-0.457980428	0.113695746	-0.007357698
r_sqr1	r_sqr2			
0.999998599	0.999803041			

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

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

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

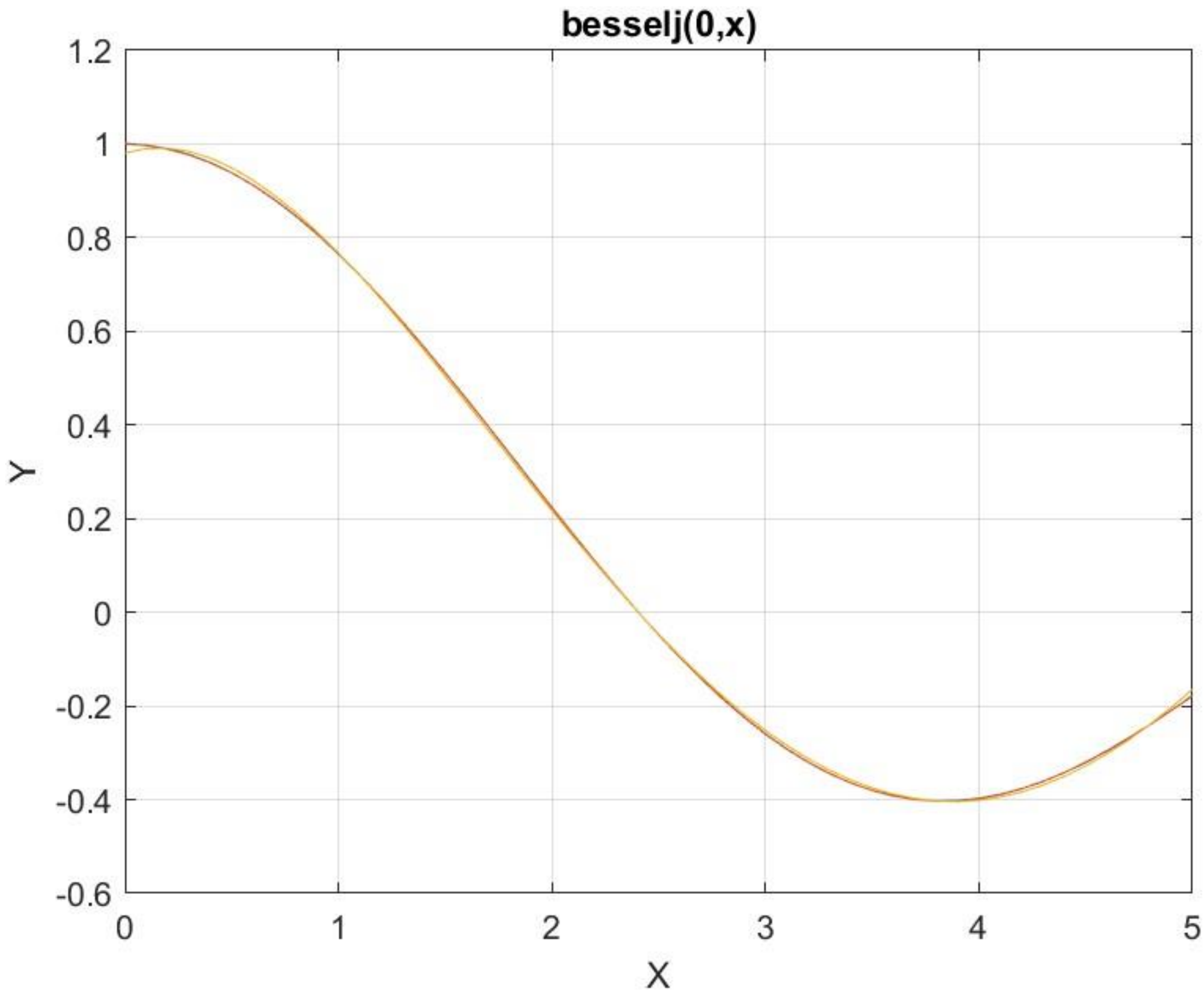

Figure 1. The graph from file besselj_0_x_run1.jpg.

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

Testing Bessel Function Fit with PSO-Run2

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

```
clc
clear
```



```
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_run2";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "besselJ(0, x)";
fprintf(sEqn);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(order, 0.5, 1.4);

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

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

QSPpwr = bestX;
```



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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + 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	
1.372668606	2.399039191	3.399926574	4.383467985	5.376741393	6.367827374	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7

0.968868841	0.151620345	-0.498803777	0.190722161	-0.029173579	0.001897274	-4.50257E-05
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.998859829	0.996718149					

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

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

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

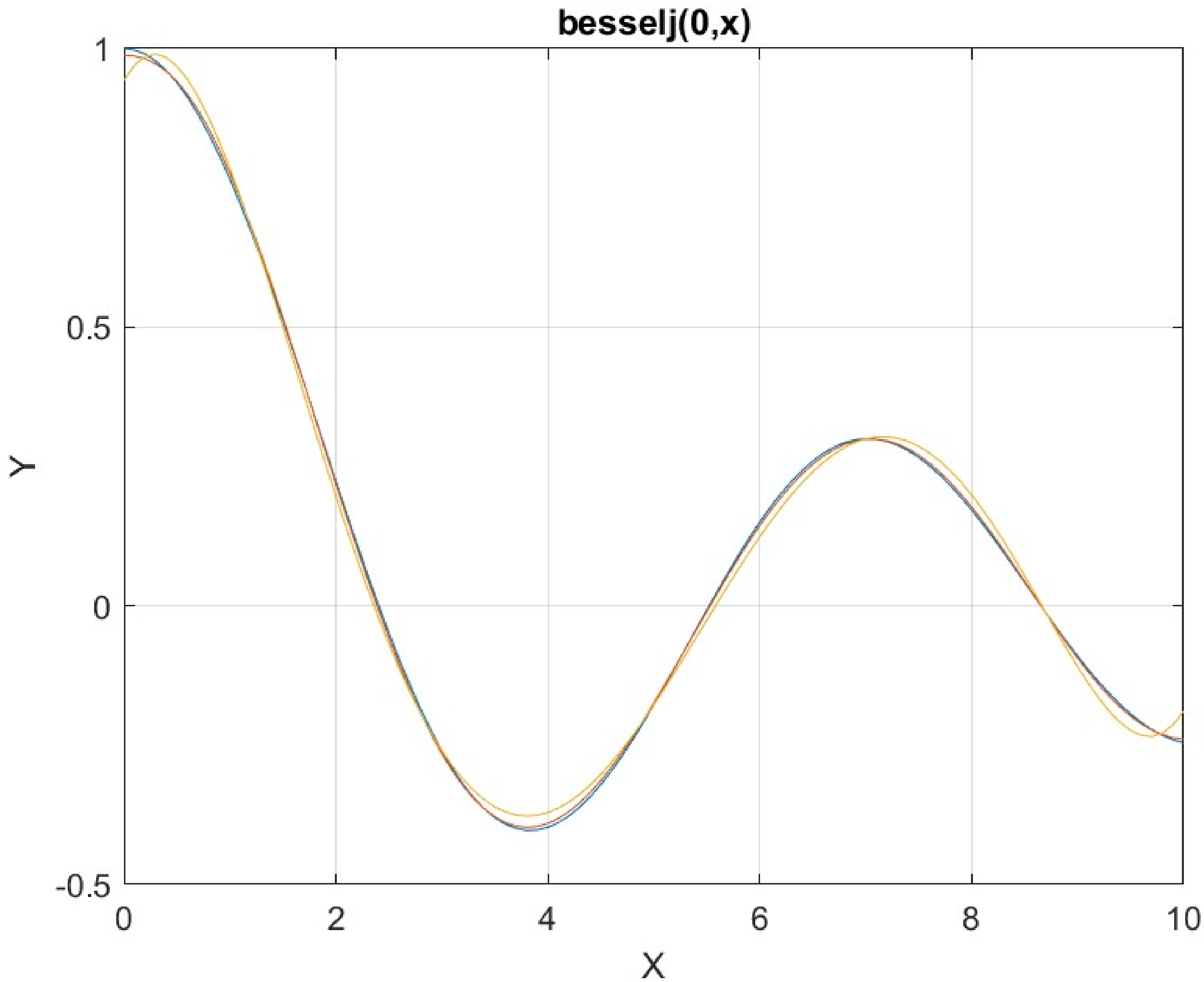

Figure 2. The graph from file besselj_0_x_run2.jpg .

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

Testing Bessel Function Fit with Random Search Optimization-Run1

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

```
clc
clear
close all
```

```
global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_random_run1";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "besselJ(0, x)";
fprintf(sEqn);
fprintf("x=0:0.1:5\n")
xData= 0:0.1:5;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.4);

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

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

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

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + 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	
1.208689038	2.371067669	3.718770338	4.06371166	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5

1.001175343	-0.047546135	-0.244256987	0.097610148	-0.041227247
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.980927341	0.138170634	-0.457980428	0.113695746	-0.007357698
r_sqr1	r_sqr2			
0.999998761	0.999803041			

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

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

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

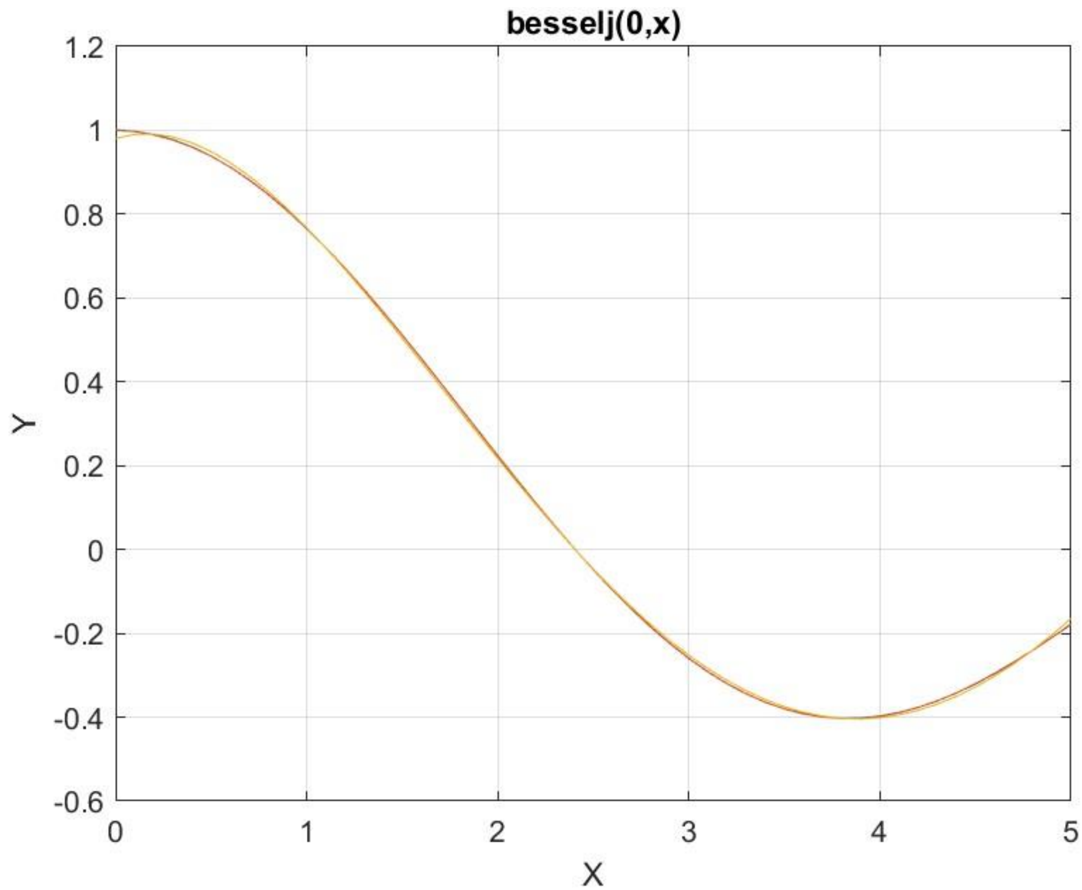

Figure 3. The graph from file besselj_0_x_random_run1.jpg.

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

Testing Bessel Function Fit with Random Search Optimization-Run2

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

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_random_run2";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
```



```
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "besselJ(0, x)";
fprintf("%s\n", sEqn);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(order, 0.5, 1.4);

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

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

QSPpwr = bestX;
Coeff = flip(c);
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A1");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A4");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A7");
```

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + 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	
1.397547017	2.593639063	3.709482101	4.76430442	5.77300541	6.977559527	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
0.988172455	-0.02442091	-0.273333587	0.099097337	-0.014912172	0.001014028	-1.42798E-05

Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.999775144	0.996718149					

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

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

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

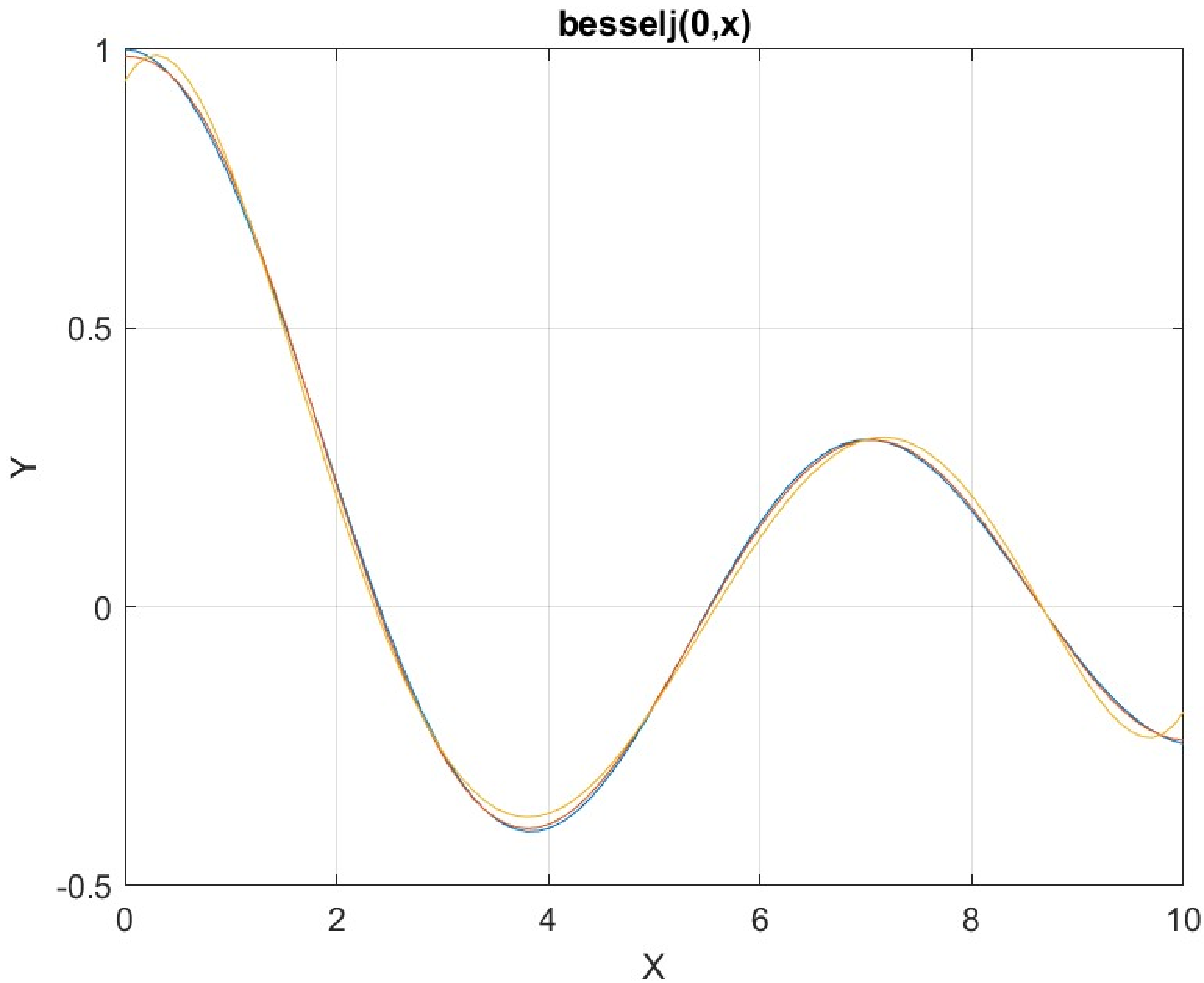

Figure 4. The graph from file besselj_0_x_random_run2.jpg.

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

Testing Bessel Function Fit with Halton Random Search Optimization-Run1

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

```
clc
```

```
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_halton_random_run1";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "besselJ(0, x)";
fprintf("%s\n",sEqn);
fprintf("x=0:0.1:5\n")
xData= 0:0.1:5;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.4);

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

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



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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + 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	
1.200216709	2.367505269	3.718058169	4.063405016	

QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
1.001146739	-0.046446745	-0.245212153	0.097382774	-0.041099665
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.980927341	0.138170634	-0.457980428	0.113695746	-0.007357698
r_sqr1	r_sqr2			
0.999998756	0.999803041			

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

The above table shows similar types of results as the ones in Table 1 and Table 3. Again, the adjusted coefficient of determination for the Quantum Shammas Polynomial is higher than that for the classical polynomial. Both are good values. Using the Halton sequence gives surprisingly good results. I suspect using one million iterations has something to do with it.

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

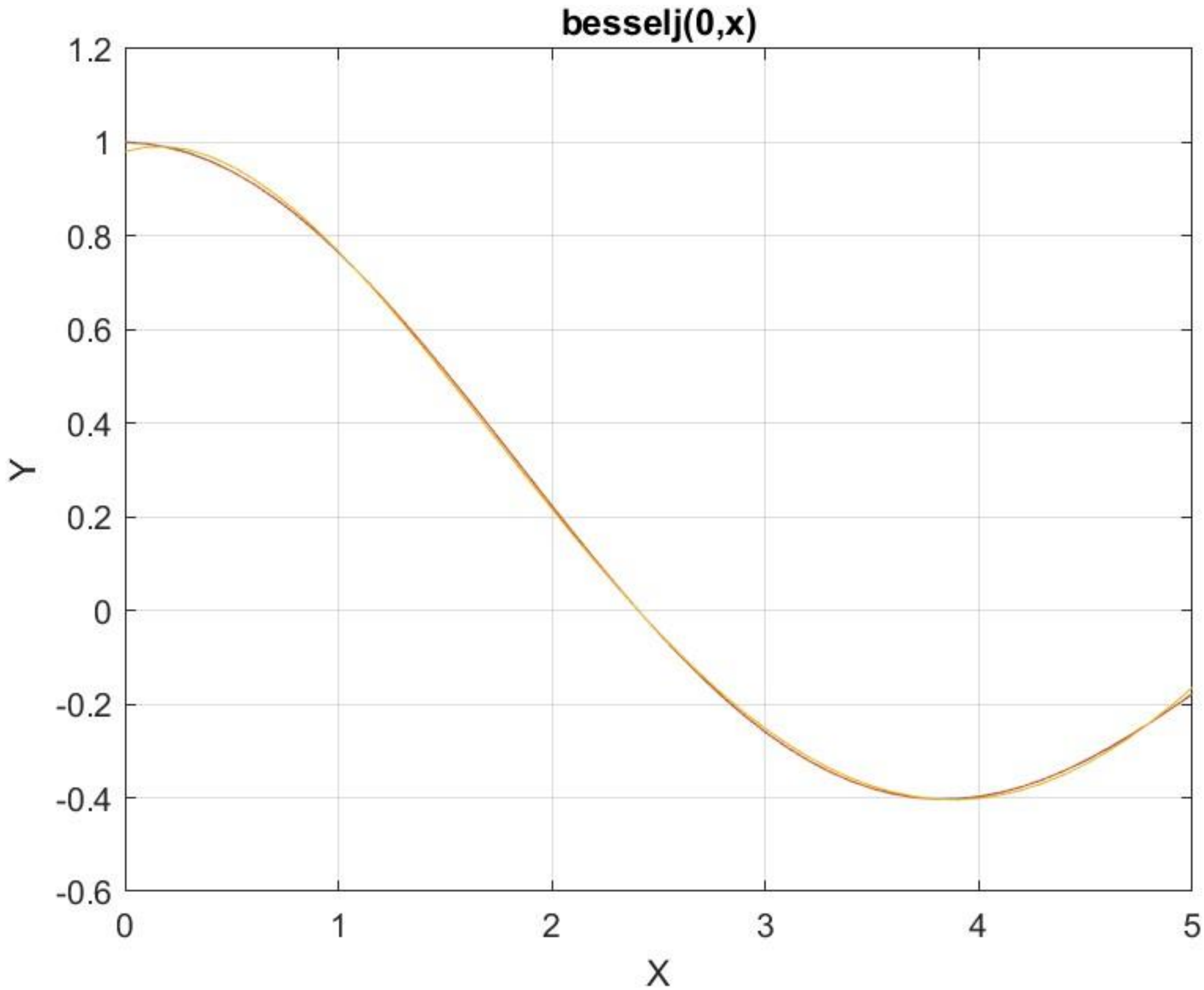

Figure 5. The graph from file besselj_0_x_halton_random_run1.jpg.

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

Testing Bessel Function Fit with Halton Random Search Optimization-Run2

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

```
clc
clear
close all
```

```
global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_halton_random_run2";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "besselJ(0, x)";
fprintf("%s\n",sEqn);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(order, 0.5, 1.4);

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

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

QSPpwr = bestX;
```



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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + 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	
1.255957298	2.619854083	3.698932578	4.745875281	5.884304742	6.755610054	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
0.989599539	-0.027639685	-0.276898654	0.104683677	-0.015261174	0.0008695	-3.94588E-05

Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.999774454	0.996718149					

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

The above table shows similar types of results as the ones in Table 2 and Table 4. Again, the adjusted coefficient of determination for the Quantum Shammas Polynomial is slightly higher than that for the classical polynomial.

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

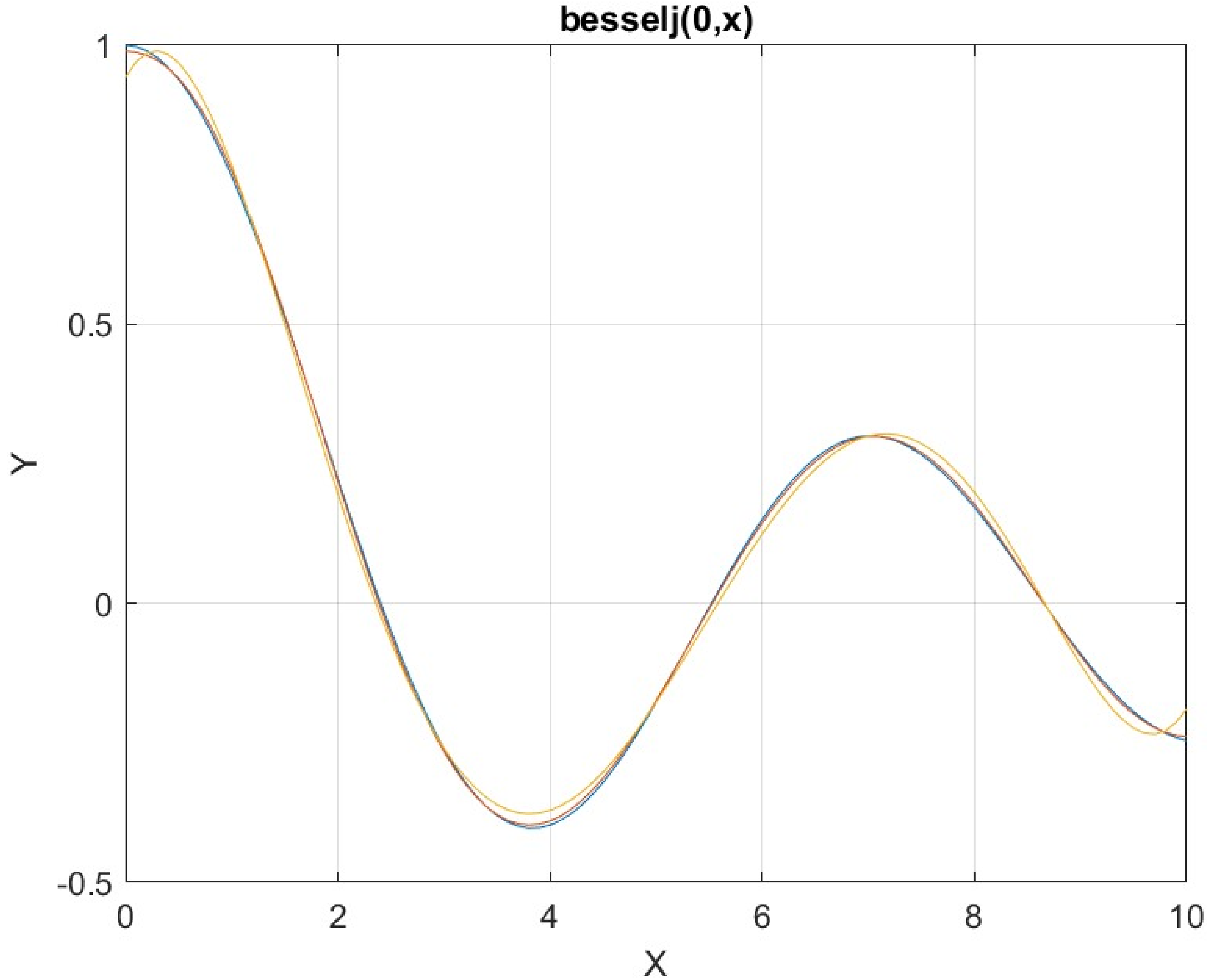

Figure 6. The graph from file besselj_0_x_halton_random_run2.jpg.

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

Testing Bessel Function Fit with Sobol Random Search Optimization-Run1

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

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
```

```
zFilename = "besselj_0_x_sobol_random_run1";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "besselJ(0, x)";
fprintf("%s\n",sEqn);
fprintf("x=0:0.1:5\n")
xData= 0:0.1:5;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.4);

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

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

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

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + 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	
1.178198979	2.36532478	3.725169921	4.043021325	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5

1.001148744	-0.044716538	-0.247344388	0.10343606	-0.04671253
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.980927341	0.138170634	-0.457980428	0.113695746	-0.007357698
r_sqr1	r_sqr2			
0.999998736	0.999803041			

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

The above table shows similar types of results as the ones in Table 1 and Table 3. Again, the adjusted coefficient of determination for the Quantum Shammas Polynomial is higher than that for the classical polynomial. Both are good values. Using the Sobol sequence gives surprisingly good results. I also suspect using one million iterations has something to do with it.

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

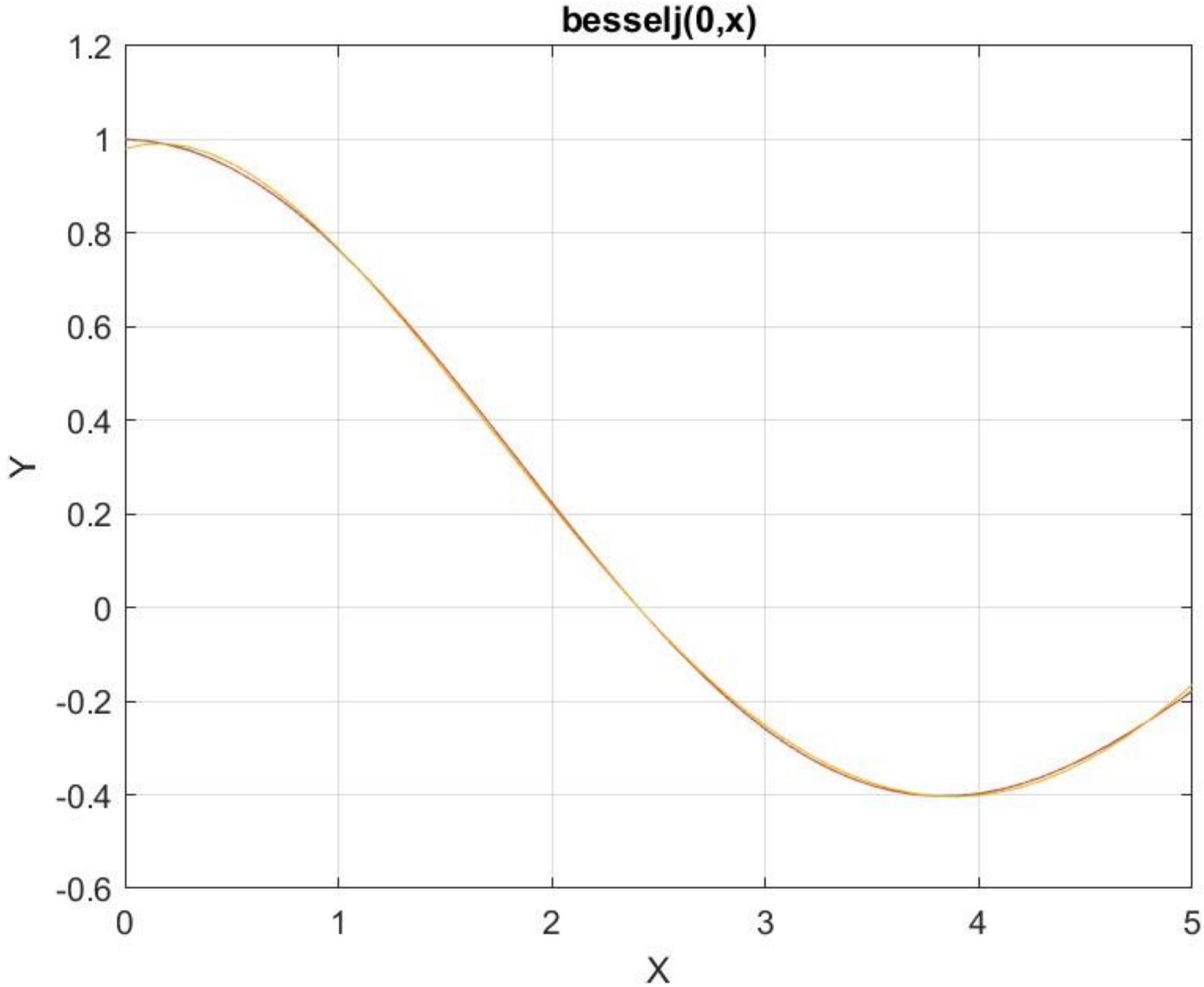

Figure 7. The graph from file besselj_0_x_sobol_random_run1.jpg.

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

Testing Bessel Function Fit with Sobol Random Search Optimization-Run2

The next MATLAB script (found in file testBessel1Sobo2.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial and a sixth order classical polynomial.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
```

```
zFilename = "besselj_0_x_sobol_random_run2";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "besselJ(0, x)";
fprintf("%s\n",sEqn);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(order, 0.5, 1.4);

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

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

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

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

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + 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	
1.268741613	2.578664106	3.729051957	4.764721434	5.858709457	6.788366254	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
0.989034301	-0.022449843	-0.272906993	0.095821571	-0.014955846	0.000922222	-3.36858E-05

Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.99977086	0.996718149					

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

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

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

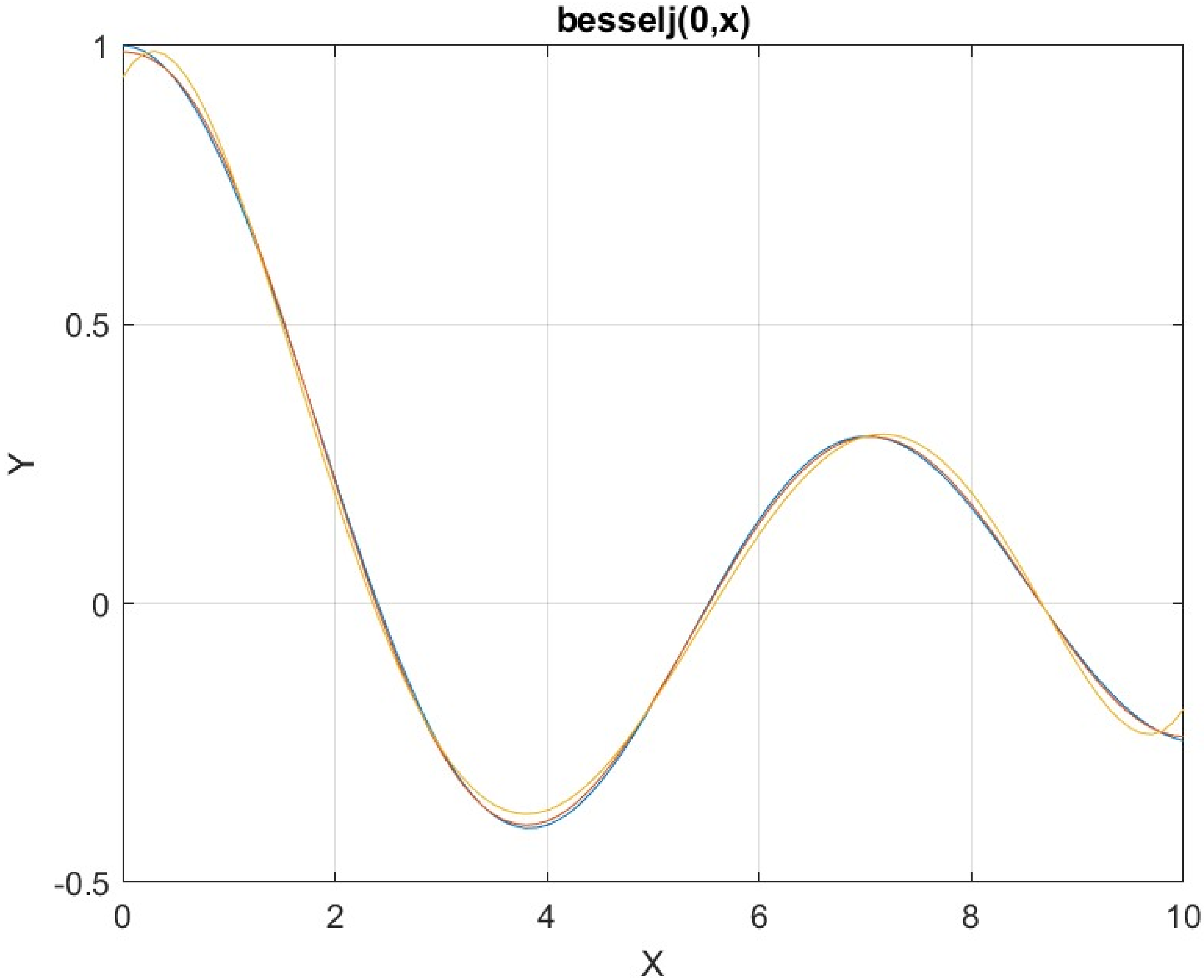

Figure 8. The graph from file besselj_0_x_sobol_random_run2.jpg

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

Conclusion for Bessel Function Fitting

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

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

Testing ln(x) Function Fit with PSO

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

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "Ln_x";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "ln(x)";
fprintf(sEqn);
fprintf("x=1:0.1:7\n")
xData0= 1:0.1:7;
xData0 = xData0';
n = length(xData0);
yData = log(xData0);
xData = xData0 - 1;
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.4);

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

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



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

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + 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	
0.991657635	1.500666526	2.500431764	3.500318656	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
-0.002091573	1.11534981	-0.450466363	0.030771063	-0.001385417
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.019526836	0.850579688	-0.210226108	0.031956872	-0.001944825
r_sqr1	r_sqr2			
0.999997545	0.99989954			

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

The adjusted coefficient of determination for the Quantum Shammas Polynomial is higher than the one for classical polynomials. Interestingly, the powers of the fitted Quantum Shammas Polynomial are approximately 1, 1.5, 2.5, and 3.5.

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

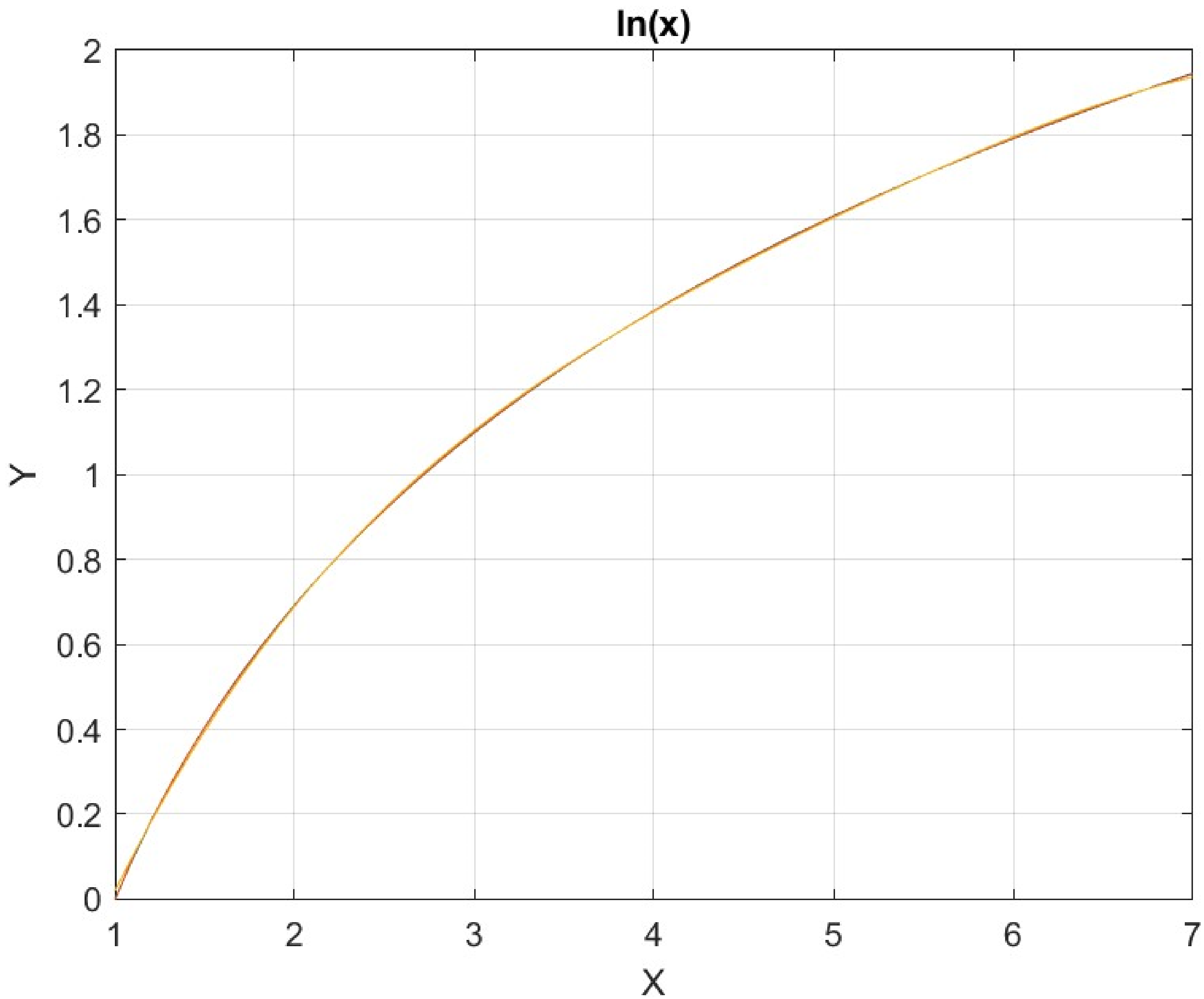

Figure 9. The graph from file ln_x.jpg

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

Testing ln(x) Function Fit with Random Search Optimization

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

```
clc
clear
```

```
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "Ln_x_rand";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "ln(x)";
fprintf(sEqn);
fprintf("x=1:0.1:7\n")
xData0= 1:0.1:7;
xData0 = xData0';
n = length(xData0);
yData = log(xData0);
xData = xData0 - 1;
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.4);

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

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



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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + 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 ln_x,m except it uses different output filenames and calls the randomSearch() function for the curve fit optimization. The above code generates the following summary Excel table:

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	
1.072746617	1.370767319	2.264574192	3.194799892	

QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
-0.001048818	1.660775415	-1.020626427	0.055841245	-0.002403173
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.019526836	0.850579688	-0.210226108	0.031956872	-0.001944825
r_sqr1	r_sqr2			
0.999999108	0.99989954			

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

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

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

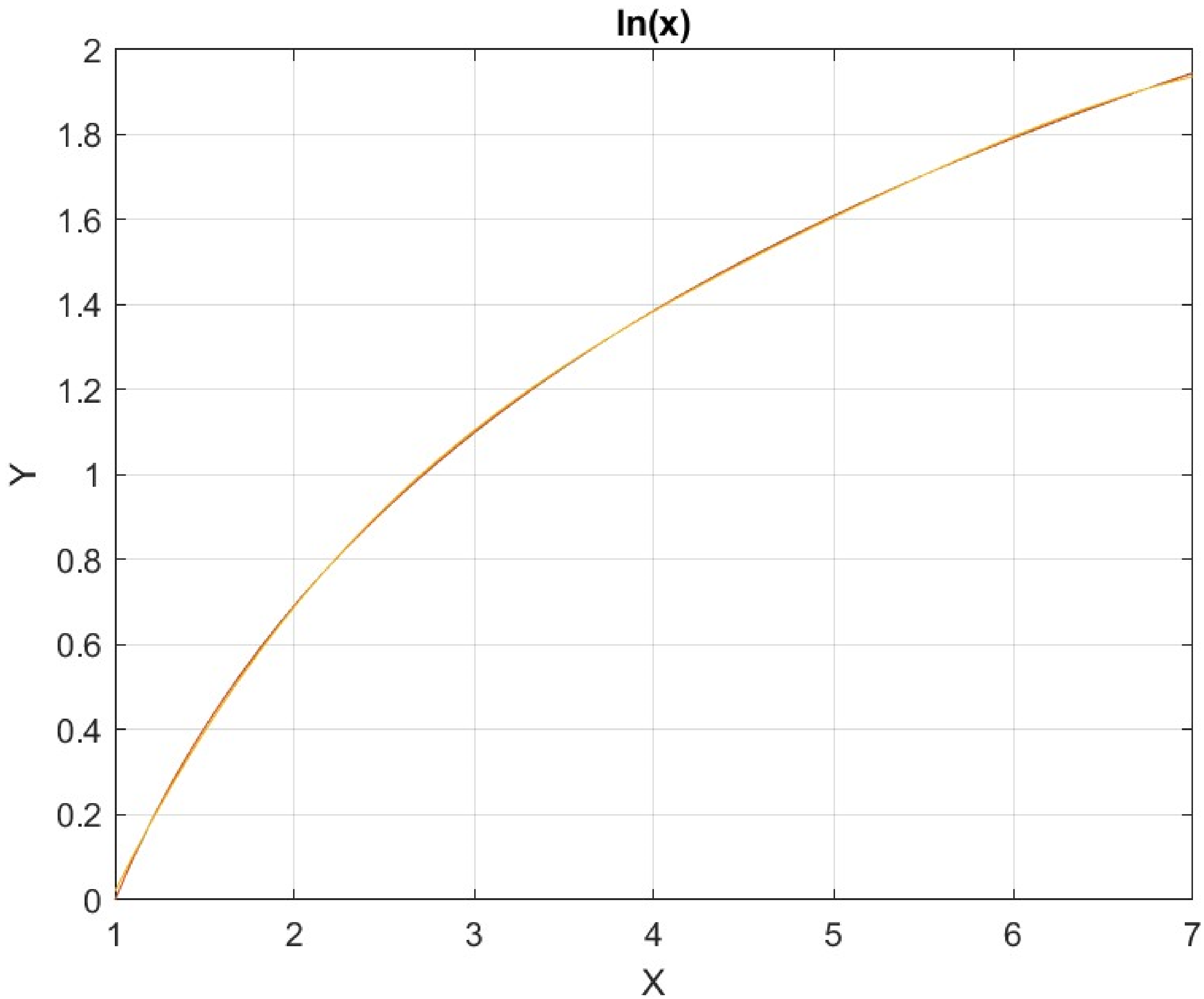

Figure 10. The graph from file ln_x_rand.jpg

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

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

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

```
clc
clear
```



```
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "Ln_x_halton_rand";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "ln(x)";
fprintf(sEqn);
fprintf("x=1:0.1:7\n")
xData0= 1:0.1:7;
xData0 = xData0';
n = length(xData0);
yData = log(xData0);
xData = xData0 - 1;
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.4);

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

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

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + 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	
1.078905121	1.358732431	2.293725799	3.171709765	

QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
-0.000878781	1.733473319	-1.089988947	0.052705316	-0.002753426
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.019526836	0.850579688	-0.210226108	0.031956872	-0.001944825
r_sqr1	r_sqr2			
0.99999911	0.99989954			

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

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

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

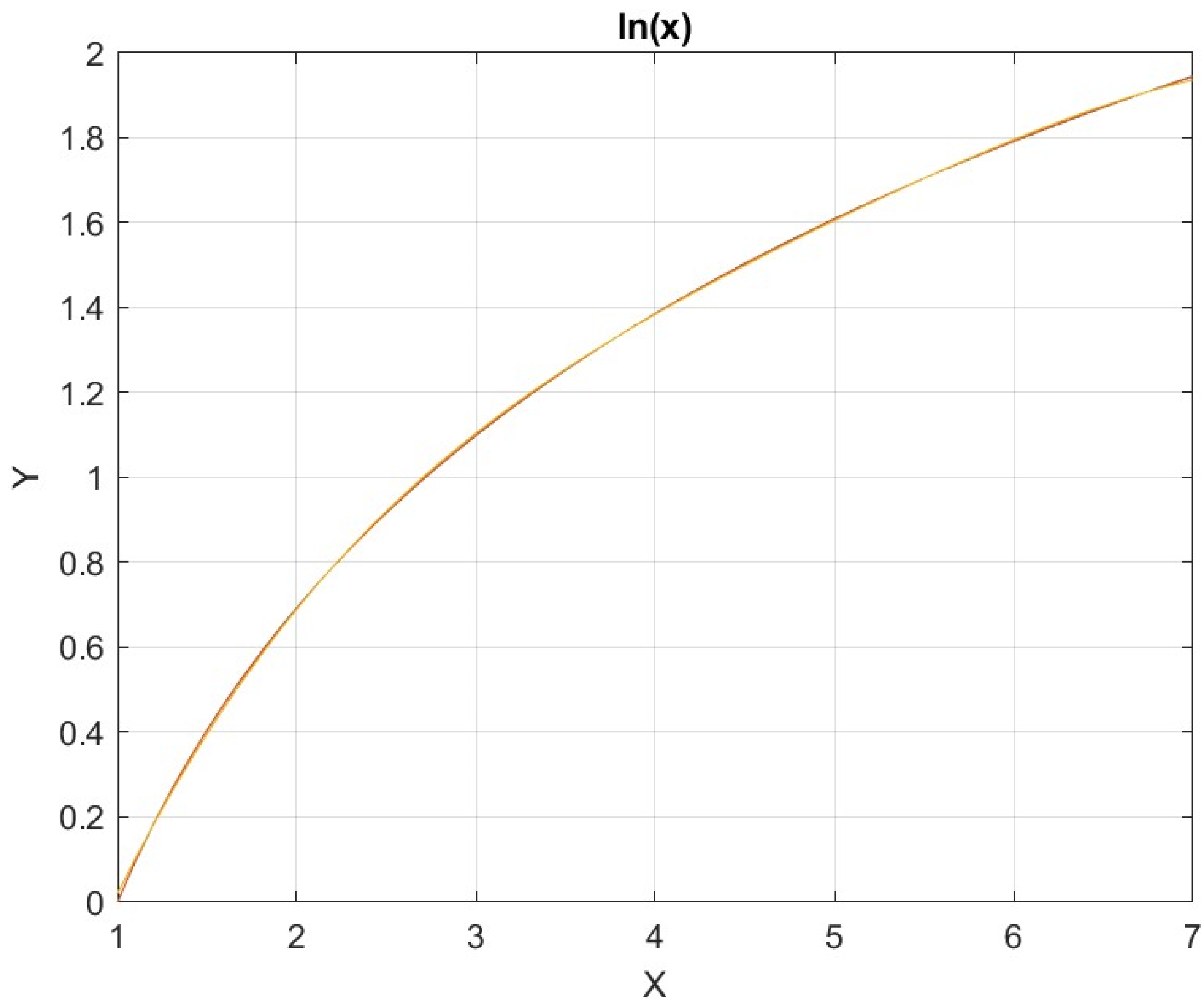

Figure 11. The graph from file ln_x_halton_rand.jpg

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

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

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

```
clc
clear
```

```
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "Ln_x_sobol_rand";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "ln(x)";
fprintf(sEqn);
fprintf("x=1:0.1:7\n")
xData0= 1:0.1:7;
xData0 = xData0';
n = length(xData0);
yData = log(xData0);
xData = xData0 - 1;
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.4);

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

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

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



```
exportgraphics(ax,gFile);

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + 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	
1.080320451	1.359033836	2.264753342	3.184459235	

QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
-0.000919835	1.747824814	-1.107690334	0.055834617	-0.002490127
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.019526836	0.850579688	-0.210226108	0.031956872	-0.001944825
r_sqr1	r_sqr2			
0.999999145	0.99989954			

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

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

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

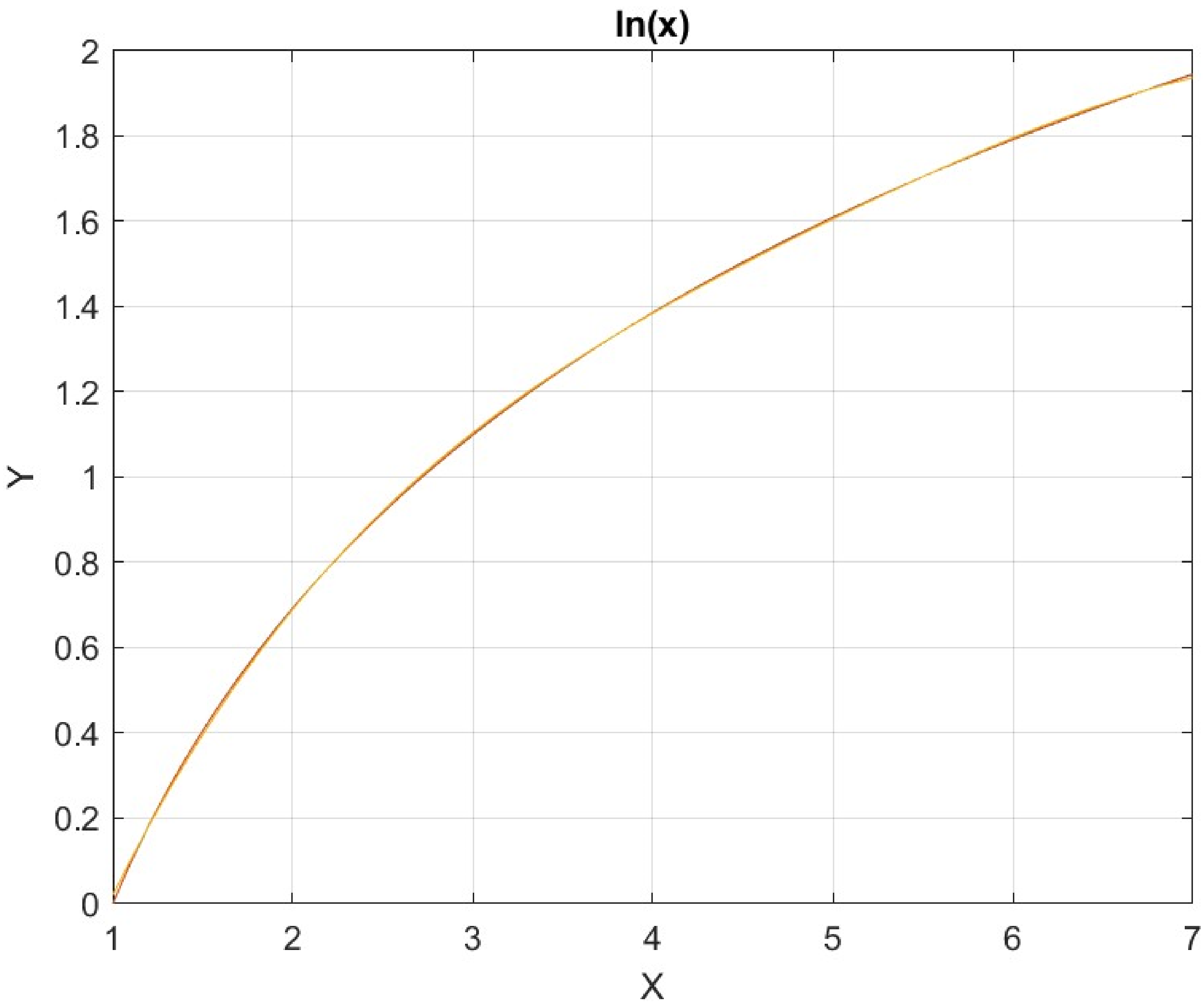

Figure 12. The graph from file ln_x_sobol_rand.jpg

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

Conclusion for Fitting the ln(x) Function

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

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

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

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

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "Right_GaussBell_x";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "exp(-x^2/2)/sqrt(2*pi)";
fprintf(sEqn);
fprintf("x=0:0.1:3\n")
xData= 0:0.1:3;
xData = xData';
n = length(xData);
yData = exp(-xData.^2/2)/sqrt(2*pi);
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.4);

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

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



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

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + 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	
1.393926502	2.399814573	2.753140127	3.501134201	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.398167858	0.02153673	-0.795788677	0.697814433	-0.080041643
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.397644494	0.028633101	-0.306018517	0.139989216	-0.018592075
r_sqr1	r_sqr2			
0.99997989	0.999967249			

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

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

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

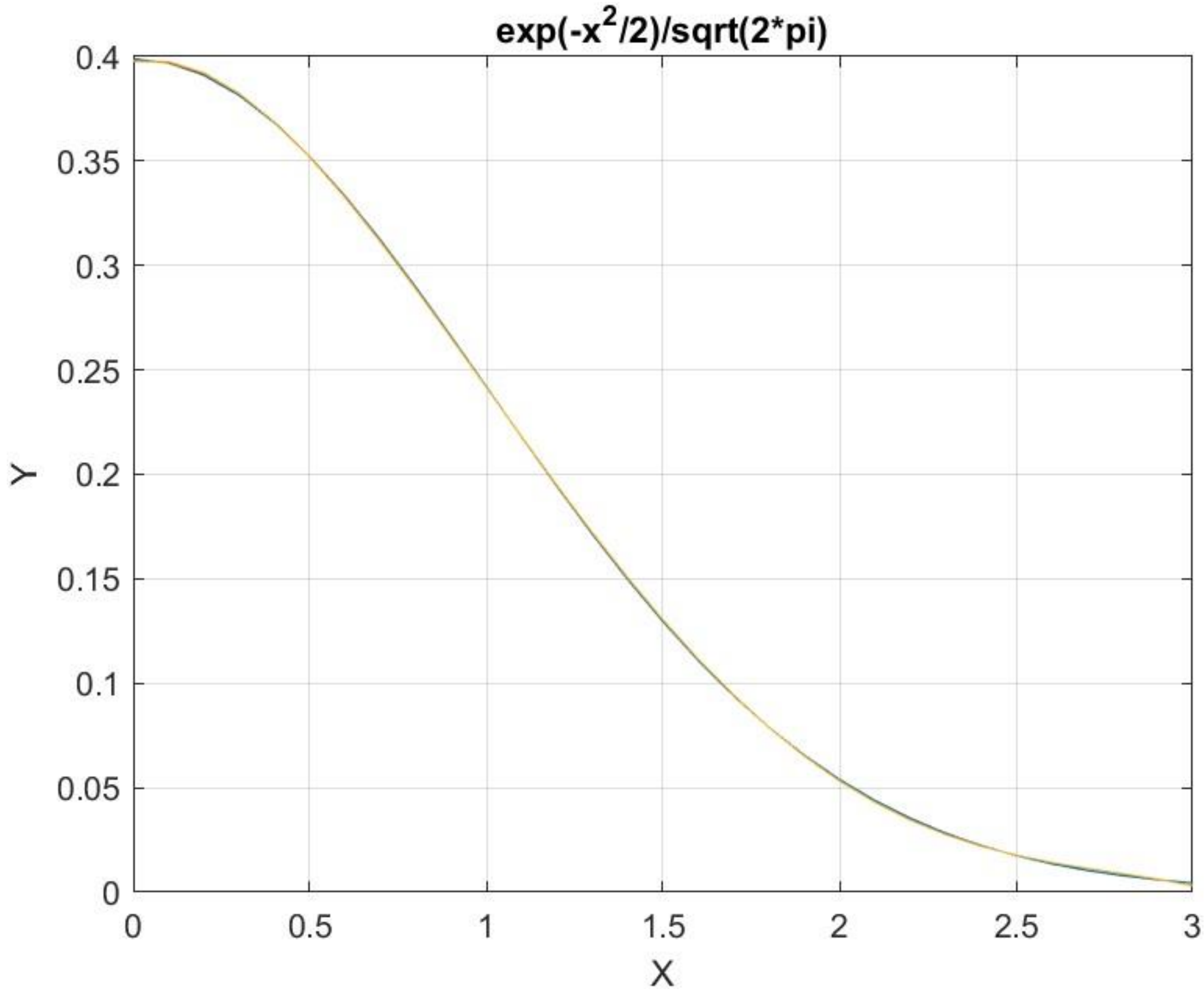

Figure 13. The graph from file Right_GaussBell_x.jpg.

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

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

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

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "Right_GaussBell_x_random";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "exp(-x^2/2)/sqrt(2*pi)";
fprintf(sEqn);
fprintf("x=0:0.1:3\n")
xData= 0:0.1:3;
xData = xData';
n = length(xData);
yData = exp(-xData.^2/2)/sqrt(2*pi);
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.4);

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

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



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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + 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	
1.529806485	2.606558356	2.790378427	3.157518428	

QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.398237661	0.014630588	-2.179238898	2.510018944	-0.501968366
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.397644494	0.028633101	-0.306018517	0.139989216	-0.018592075
r_sqr1	r_sqr2			
0.999982158	0.999967249			

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

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

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

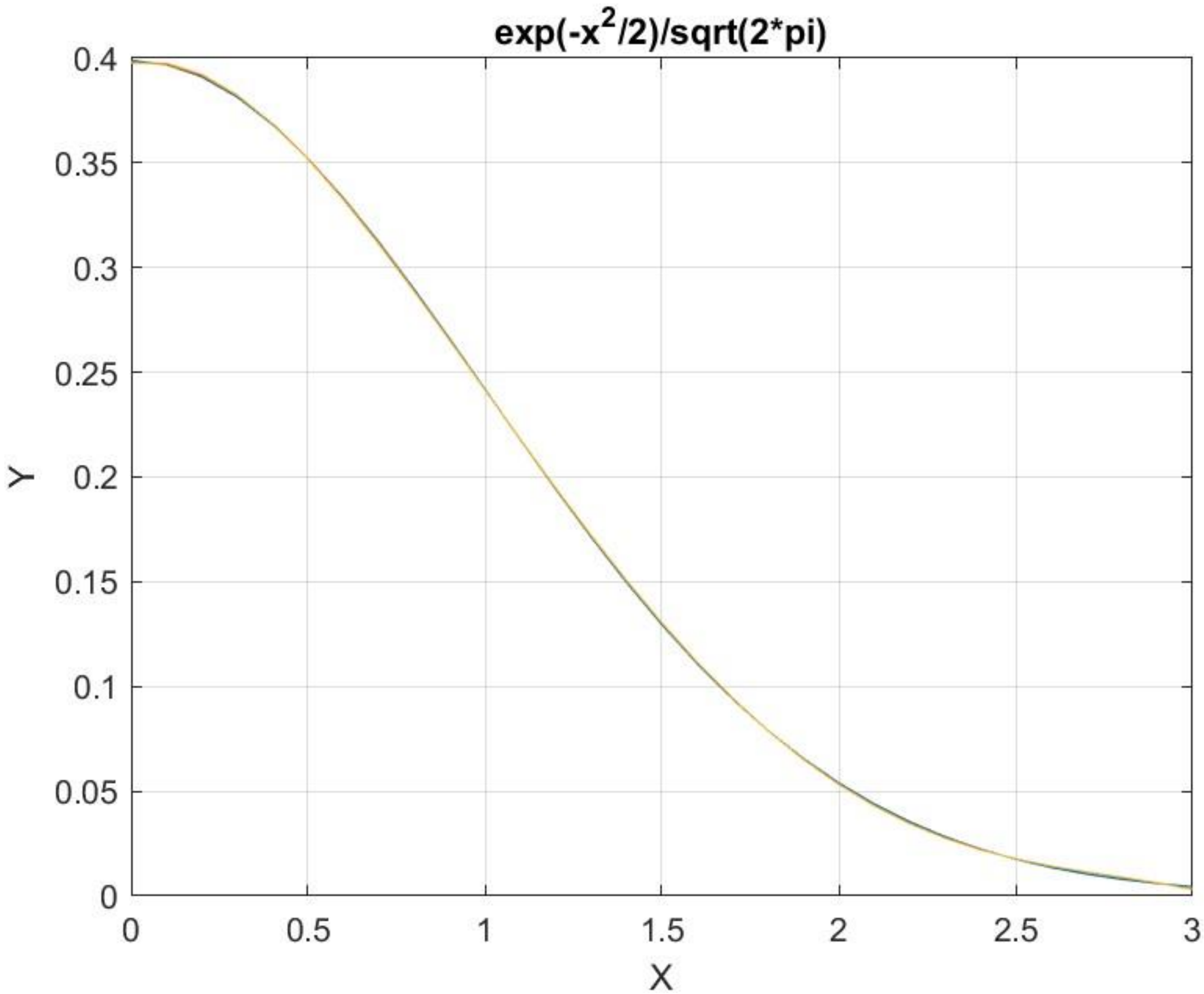

Figure 14. The graph from file Right_GaussBell_x_random.jpg.

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

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

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

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "Right_GaussBell_x_halton_random";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "exp(-x^2/2)/sqrt(2*pi)";
fprintf(sEqn);
fprintf("x=0:0.1:3\n")
xData= 0:0.1:3;
xData = xData';
n = length(xData);
yData = exp(-xData.^2/2)/sqrt(2*pi);
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.4);

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

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



```
exportgraphics(ax,gFile);

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + 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	
1.514658887	2.631929252	2.752323399	3.168725289	

QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.398228945	0.014349079	-3.286588632	3.56247719	-0.446790579
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.397644494	0.028633101	-0.306018517	0.139989216	-0.018592075
r_sqr1	r_sqr2			
0.999982146	0.999967249			

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

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

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

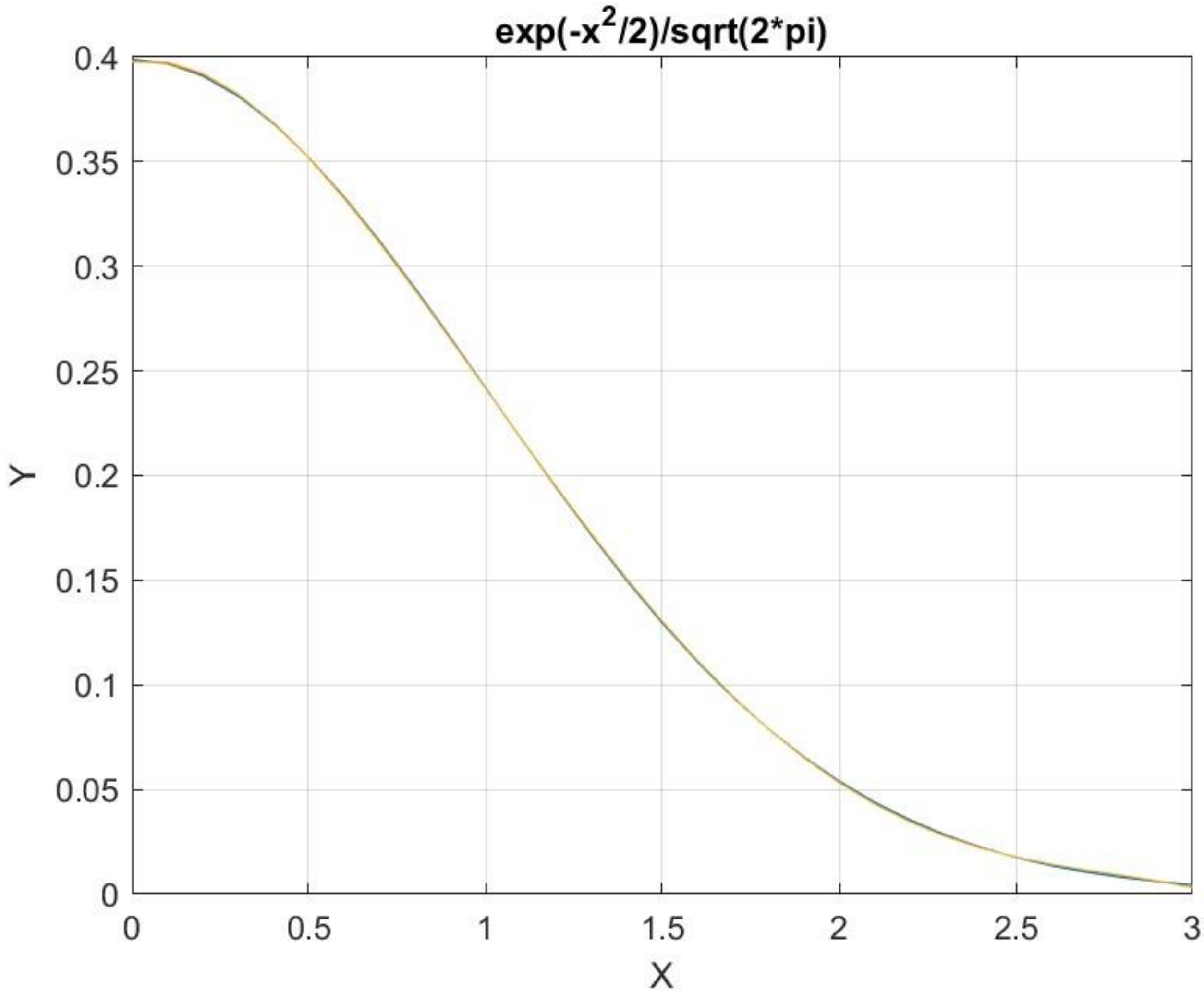

Figure 15. The graph from file Right_GaussBell_x_halton_random.jpg.

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

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

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


```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "Right_GaussBell_x_sobol_random";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "exp(-x^2/2)/sqrt(2*pi)";
fprintf(sEqn);
fprintf("x=0:0.1:3\n")
xData= 0:0.1:3;
xData = xData';
n = length(xData);
yData = exp(-xData.^2/2)/sqrt(2*pi);
order = 4;
[Lb,Ub] = makeLimits(order, 0.5, 1.4);

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

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

```
exportgraphics(ax,gFile);

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + 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	
1.522003875	2.598938314	2.788477382	3.160692568	

QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.398216639	0.015665411	-2.095319336	2.406124769	-0.483016053
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.397644494	0.028633101	-0.306018517	0.139989216	-0.018592075
r_sqr1	r_sqr2			
0.999982135	0.999967249			

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

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

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

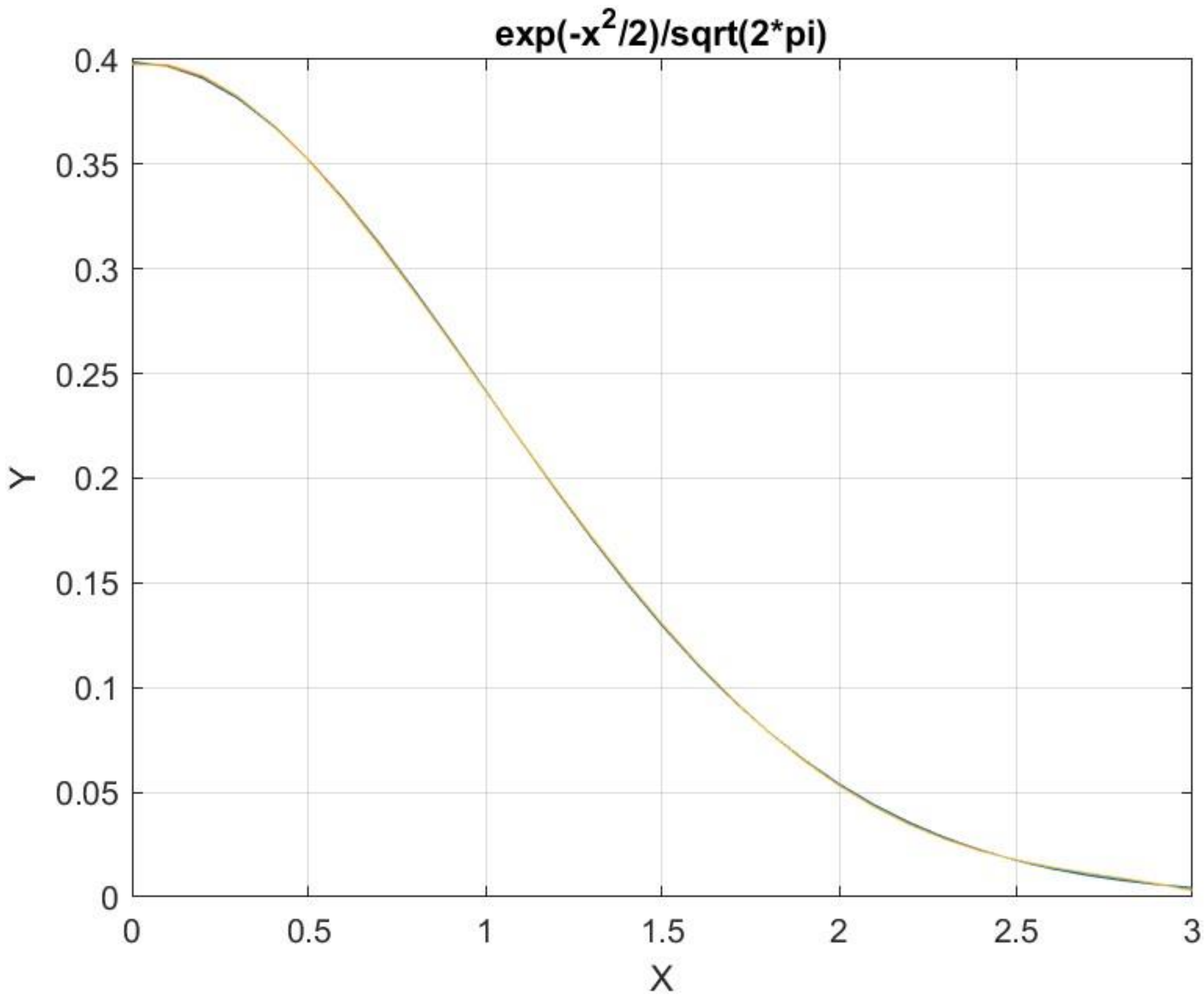

Figure 16. The graph from file Right_GaussBell_x_sobol_random.jpg.

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

Conclusion for Fitting the Right-Side Normal Gaussian Function

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

Conclusion for Part 1

The Quantum Shammas Polynomials 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 1B

Part 1B of this study looks at the Quantum Shammas Polynomials with wider ranges of random powers for most of the test cases presented in this part.

Document History

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

