Optimum Quantum Shammas Padé Polynomials
Part 2 of the Study

By
Namir Shammas

The better is the enemy of the good--Voltaire

Contents

Introduction

Quantum Shammas Polynomials are inspired by how quantum physics views the orbits of the electrons in an atom. These 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 and fixed orbits. The Heisenberg principle proposed that the orbits of the electrons in an atom are more probabilistic than fixed. This is the inspiration for Quantum Shammas Polynomials. While classical polynomials have the familiar fixed integer powers:

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

By contrast, the Quantum Shammas Polynomials have random powers that range between (I-1)+0.5 to I+0.4 where I is the term number. An example of a Quantum Shammas Polynomial is:

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

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.

The values of the random powers (r_i) are chosen to minimize the sum of errors squared between some observed values of y(x) and the ones calculated using equation (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 a Padé version of the Quantum Shammas Polynomials. A Quantum Shammas Padé Polynomials looks like:

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

Where the values for r_i and s_i follow the ranges of values that I discussed above. The study compares the best Quantum Shammas Padé Polynomials with classical Padé polynomials, expressed as:

$$\begin{aligned} y(x) &= P_c(x) / Q_c(x) \\ &= (c_0 + c_1*x + c_2*x^2 + \ldots + c_{np}*x^p) \ / \\ &\quad (1 + d_1*x + d_2*x^2 + \ldots + d_q*x^q) \end{aligned} \tag{4}$$

This way we proverbially compare apples and apples. P(x) and $P_c(x)$ are numerator polynomials. Q(x) and $Q_c(x)$ are denominator polynomials. Notice that the constant terms for polynomials Q(x) and $Q_c(x)$ are always 1.

☛This part attempts to avoid choosing arbitrary orders for the Quantum Shammas Padé Polynomials. Rather, it aims at determining the optimum orders of the P(x) and Q(x) polynomials to give the best least square fit of a tested function. These optimum polynomials are then compared to classical Padé polynomials of the same orders as polynomials P(x) and Q(x).

☛ 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.

The Padé Fit Function

The next MATLAB function (found in file padefit.m) performs least squares fit for classical Padé polynomials.

```
function [c,r2,yCalc] = padefit(xData,yData,orderP,orderQ)
%FOURIERFIT Summary of this function goes here
%   Detailed explanation goes here
  n = length(xData);
  X = [1+zeros(n,1)];
  for j=1:orderP
    X = [X xData.^j];
  end
  for j=1:orderQ
    X = [X -yData.*xData.^j];
  end
  [c] = regress(yData,X);
  SSE = 0;
  ymean = mean(yData);
  SStot = sum((yData - ymean).^2);
  yCalc = zeros(n,1);
  for i=1:n
    Px = c(1);
    for j=1:orderP
      Px = Px + c(j+1)*xData(i)^j;
    end
    Qx = 1;
    k=1+orderP;
    for j=1:orderQ
      Qx = Qx + c(k+j)*xData(i)^j;
    end
    yCalc(i) = Px / Qx;
    SSE = SSE + (yCalc(i) - yData(i))^2;
  end
  r2 = 1 - SSE / SStot;
end
```

The input parameters for the above function are:

- The xData parameter is the array of x values.
- The yData parameter is the array of y values.
- The orderP parameter is the value for the order of numerator polynomial P(x).
- The orderQ parameter is the value for the order of denominator polynomial Q(x).

The output parameters are:

- The c parameter is the array of coefficients for the polynomials P(x) and Q(x).
- The r2 parameter is the coefficient of determination.
- The yCalc parameter is the array of projected y values for the given array of x values in the input parameter xData.

The Quantum Shammas Padé Polynomial Function

The Quantum Shammas Padé Polynomial function (found in file quantShammasPadePoly.m) evaluates a Quantum Shammas Padé Polynomial. The MATLAB code for the function is:

```
function SSE = quantShammasPadePoly(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 takes one input parameter, the array of random powers pwr for the numerator and denominator polynomials. The global variables orderP and orderQ lets the function know the orders of the numerator and denominator polynomials. The function returns the sum of errors squared. The function builds the regression matrix using two for loops—one for the numerator polynomial and one for denominator polynomial. The terms for the latter polynomial include the factor –yData. The function then calls function regress() to obtain the regression coefficients. The function 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:

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

The output parameters are:

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

The Random Search Padé Function

The next function (found in file randomSearchPade.m) performs a random search optimization:

```
function [bestX,bestFx] =
randomSearchPade(fx,LbP,UbP,LbQ,UbQ,MaxIters)
%RANDOMSEARCH Summary of this function goes here
%   Detailed explanation goes here
  bestFx = 1e99;
  nP = length(LbP);
  nQ = length(LbQ);
  bestX = 1e+99+zeros(nP+nQ,1);
  for irun=1:2
    iter = 0;
    while iter < MaxIters
      iter = iter + 1;
      XP = LbP + (UbP - LbP).*rand(1,nP);
      XQ = LbQ + (UbQ - LbQ).*rand(1,nQ);
      X = [XP XQ];
      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

    bestXQ = bestX(1:nQ);
    bestXP = bestX(1+nQ:end);
    delta = 0.15;
    LbP = bestXP - delta;
    UbP = bestXP + delta;
    LbQ = bestXQ - delta;
    UbQ = bestXQ+ delta;
    MaxIters = fix(MaxIters/2);
  end
end
```

The function has the following input parameters:

- fx is the handle of the optimized function.
- LbP is the row array of low bound values for the numerator polynomial.

- UbP is the row array of upper bound values for the numerator polynomial.
- LbQ is the row array of low bound values for the denominator polynomial.
- UbQ is the row array of upper bound values for the denominator polynomial.
- MaxIters is the maximum number of iterations.

The output parameters are:

- bestX is the array of best solutions.
- 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 LbP, UbP, LbQ and UbQ) that are supplied to the function. The second pass narrows the values of arrays LbP, UbP, LbQ and UbQ to be around the best values of x obtained at the end of the first pass.

The Halton Quasi Random Search Padé Function

The next function (found in file haltonRandomSearchPade.m) performs random-search optimization using the Halton quasi-random sequences:

```
function [bestX,bestFx] =
haltonRandomSearchPade(fx,LbP,UbP,LbQ,UbQ,MaxIters)
%RANDOMSEARCH Summary of this function goes here
%   Detailed explanation goes here
  bestFx = 1e99;
  nP = length(LbP);
  nQ = length(LbQ);
  bestX = 1e+99+zeros(nP+nQ,1);

  for irun=1:2
    % set up halton sequences
    pP = haltonset(nP,'Skip',1e3,'Leap',1e2);
    pP = scramble(pP,'RR2');
    randoP = net(pP,MaxIters);
    pQ = haltonset(nQ,'Skip',1e3,'Leap',1e2);
    pQ = scramble(pQ,'RR2');
    randoQ = flip(net(pQ,MaxIters));
    iter = 0;
    while iter < MaxIters
      iter = iter + 1;
      for i=1:nP
        pwrP(i) = LbP(i) + (UbP(i) - LbP(i))*randoP(iter,i);
```



```
      end
      for i=1:nQ
        pwrQ(i) = LbQ(i) + (UbQ(i) - LbQ(i))*randoQ(iter,i);
      end

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

    bestXQ = bestX(1:nQ);
    bestXP = bestX(1+nQ:end);
    delta = 0.15;
    LbP = bestXP - delta;
    UbP = bestXP + delta;
    LbQ = bestXQ - delta;
    UbQ = bestXQ+ delta;
    MaxIters = fix(MaxIters/2);
  end
end
```

The above function has the same input and output parameters as the randomSearchPade() 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 matrices randoP and randoQ. The function then accesses the values in these matrices as needed.

The Sobol Quasi Random Search Padé Function

The next function (found in file sobolRandomSearchPade.m) performs random-search optimization using the Sobol quasi-random sequences:

```
function [bestX,bestFx] =
sobolRandomSearchPade(fx,LbP,UbP,LbQ,UbQ,MaxIters)
%RANDOMSEARCH Summary of this function goes here
%   Detailed explanation goes here
  bestFx = 1e99;
  nP = length(LbP);
  nQ = length(LbQ);
```



```
  bestX = 1e+99+zeros(nP+nQ,1);

  for irun=1:2
    % set up halton sequences
    pP = sobolset(nP,'Skip',1e3,'Leap',1e2);
    pP = scramble(pP,'MatousekAffineOwen');
    randoP = net(pP,MaxIters);
    pQ = sobolset(nQ,'Skip',1e3,'Leap',1e2);
    pQ = scramble(pQ,'MatousekAffineOwen');
    randoQ = flip(net(pQ,MaxIters));
    iter = 0;
    while iter < MaxIters
      iter = iter + 1;
      for i=1:nP
        pwrP(i) = LbP(i) + (UbP(i) - LbP(i))*randoP(iter,i);
      end
      for i=1:nQ
        pwrQ(i) = LbQ(i) + (UbQ(i) - LbQ(i))*randoQ(iter,i);
      end

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

    bestXQ = bestX(1:nQ);
    bestXP = bestX(1+nQ:end);
    delta = 0.15;
    LbP = bestXP - delta;
    UbP = bestXP + delta;
    LbQ = bestXQ - delta;
    UbQ = bestXQ+ delta;
    MaxIters = fix(MaxIters/2);
  end
end
```

The above function has the same input and output parameters as the randomSearchPade() 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 matrices randoP and randoQ. The function then accesses the values in these matrices as needed.

Testing Quantum Shammas Padé Polynomials

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

All of the testing to determine the best Optimum Quantum Shammas Padé Polynomials use nested loops that iterate over the orders of the numerator polynomial P(x) and denominator polynomial Q(x). These nested loops iterate over the ranges from 2 to 6 for each of P(x) and Q(x). The code retains the best adjusted coefficient of determination, best order for P(x), best order for Q(x), and other data that lead to the calculation of the best Quantum Shammas Padé Polynomials. The code then compares the results of the Optimum Quantum Shammas Padé Polynomials with the classical Padé polynomial that has the same order as the P(x) numerator polynomial and the Q(x) denominator polynomial.

Testing Bessel Function Fit with PSO-Run1

The next MATLAB script (found in file testPadeBessel1pso.m) tests fitting Bessel J(0, x) for x in the range (0, 5) and samples at 0.1 steps.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
global orderP orderQ
zFilename = "besselj_0_x_pade_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);
bestR2 = -1;
for orderP=2:6
  for orderQ=2:6
    fprintf("\norderP = %d, orderQ=%d\n\n", orderP,orderQ)
    [LbP,UbP] = makeLimits(orderP, 0.5, 1.4);
    [LbQ,UbQ] = makeLimits(orderQ, 0.5, 1.4);
    Lb = [LbP LbQ];
    Ub = [UbP UbQ];
    [bestX,bestFx] =
psox(@quantShammasPadePoly,Lb,Ub,1000,5000,true);

    SSE = quantShammasPadePoly(bestX);
    orderTot = orderP + orderQ;
    % calculate adjusted value of the coefficient of
determination
    glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-orderTot-1);
    if bestR2 < glbRsqr
      bestR2 = glbRsqr;
      bestP = orderP;
      bestQ = orderQ;
      bestQSPcoeff = QSPcoeff;
      zBestX = bestX;
      bestYCalc = yCalc;
    end
  end
end
fprintf("Best orderP = %d\n", bestP);
fprintf("Best orderQ = %d\n", bestQ);
fprintf("Adjusted Rsqr = %f\n", bestR2);
fprintf("Quantum Shammas Pade Polynomial Powers\n");
zBestX
fprintf("Quantum Shammas Pade Polynomial Coefficients\n");
QSPcoeff = bestQSPcoeff'
fprintf("\nRegular Pade polynomial fit\n");
[c,r,yPoly] = padefit(xData,yData,bestP,bestQ);
% calculate adjusted value of the coefficient of determination
orderTot = bestP + bestQ;
r = 1 - (1 - r)*(n-1)/(n-orderTot-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,bestYCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
```



```
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = zBestX;
Coeff = 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 = [bestR2 r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
padeOrderP = bestP;
padeOrderQ = bestQ;
orderP = [padeOrderP];
T = array2table(orderP);
writetable(T,xlFile,"Sheet","Sheet1","Range","D10");
orderQ = [padeOrderQ];
T = array2table(orderQ);
writetable(T,xlFile,"Sheet","Sheet1","Range","E10");
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
```

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
1.298317894	1.935706609	3.177943287	4.359882547	5.399999592

QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.999971056	0.009793132	-0.249667412	-0.009868938	0.018651209
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
1.000005356	-0.098317992	-0.231950891	0.022936247	0.012914014
r_sqr1	r_sqr2		orderP	orderQ
0.999999996	1		6	2

Table 1a. Summary of the results appearing in file besselj_0_x_pade_run1.xlsx.

QSPpwr6	QSPpwr7	QSPpwr8	
6.023203509	0.611373836	2.273241019	
QSPcoeff6	QSPcoeff7	QSPcoeff8	QSPcoeff9
-0.004323076	0.000673773	-4.17857E-07	1.45785E-07
Coeff6	Coeff7	Coeff8	Coeff9
-0.002421054	0.000112057	-0.098155261	0.017236385

Table 1b. Summary of the results appearing in file besselj_0_x_pade_run1.xlsx.

The second row shows the powers for the fitted Quantum Shammas Padé Polynomial. Notice that the second to the fourth powers have a common fractional part that begins with 0.39. The fifth row shows the intercept (below QSPcoeff1) and to its right the coefficients for the various Quantum Shammas Padé Polynomial. The eighth row shows the intercept and coefficients for the classical Padé polynomial. The cell under r_sqr1 shows the adjusted coefficient of determination for the fitted Quantum Shammas Padé Polynomial. The cell under r_sqr2 shows the adjusted coefficient of determination for the fitted classical Padé polynomial. The optimum orders for P(x) and Q(x) are 6 and 2, respectively. The calculations use these same orders for the classical Padé polynomial. The adjusted coefficient of determination for the fitted optimum Quantum Shammas Padé Polynomial (with a total of 8

polynomial terms) is slightly less than the one for the classical Padé polynomial of the same order. This condition indicates that the Quantum Shammas Padé Polynomial falls short by a small amount.

☛	
	To read the results tables in this study, follow these rules. 1. For the Padé Polynomial powers: Given n entries in a row. The first orderP entries are the powers of the P(x) numerator polynomial. The last orderQ entries are the powers of the Q(x) denominator polynomial. 2. For the Padé Polynomial coefficients: given n entries in a row. The first 1+orderP entries are the coefficients of the P(x) numerator polynomial, including the constant term. The last orderQ entries are the coefficients of the Q(x) denominator polynomial. Remember that constant term for the Q(x) denominator polynomial is 1.

Here is the graph (from file besselj_0_x_pade_run1.jpg) for the Bessel function and the two fitted polynomials:

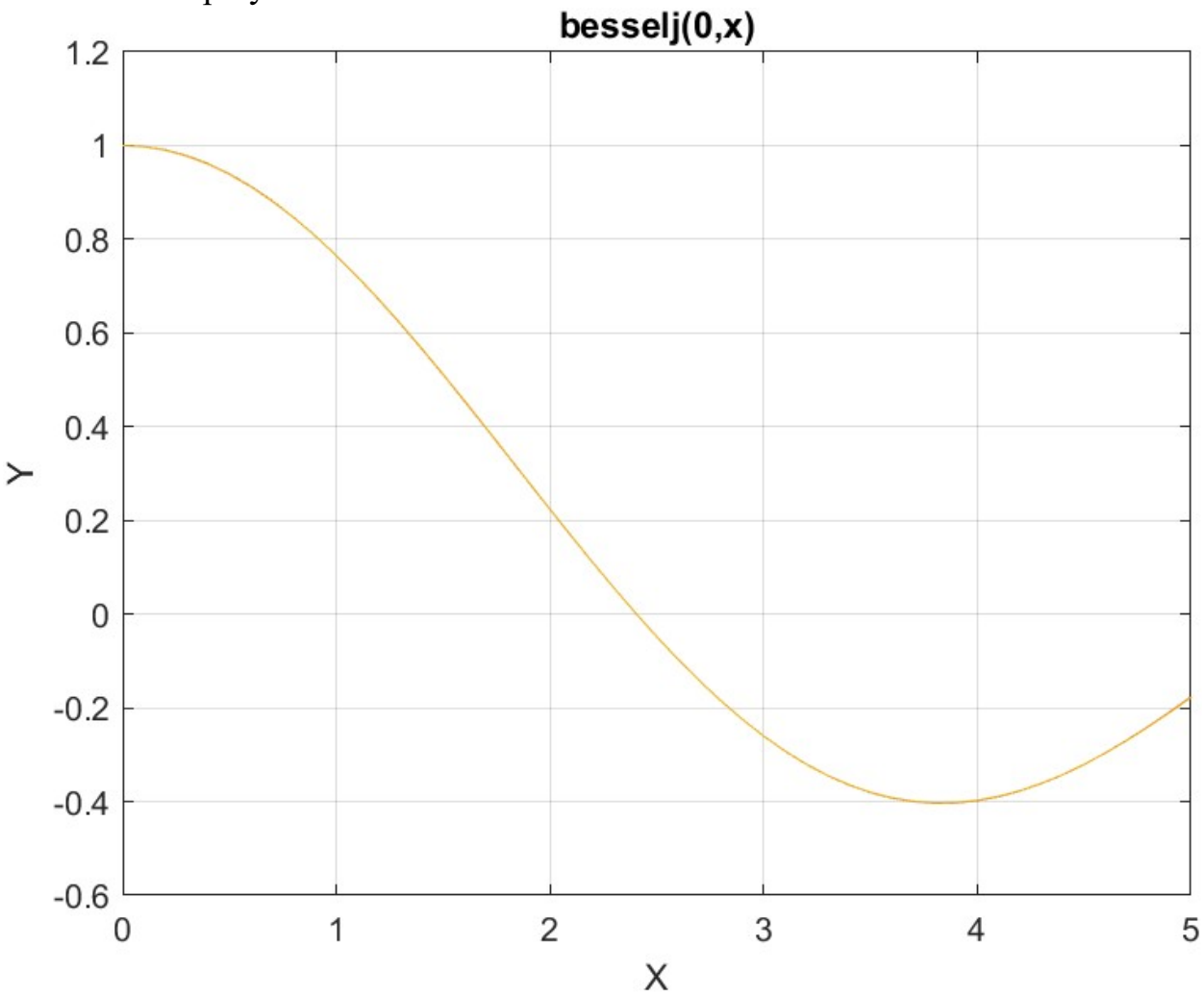

Figure 1. The graph from file besselj_0_x_pade_run1.jpg.

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

Testing Bessel Function Fit with PSO-Run2

The next MATLAB script (found in file testPadeBessel2pso.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps.

```
clc
clear
close all
```



```
global xData yData yCalc glbRsqr QSPcoeff
global orderP orderQ
zFilename = "besselj_0_x_pade_pso_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);

bestR2 = -1;
for orderP=2:6
  for orderQ=2:6
    fprintf("\norderP = %d, orderQ=%d\n\n", orderP,orderQ)
    [LbP,UbP] = makeLimits(orderP, 0.5, 1.4);
    [LbQ,UbQ] = makeLimits(orderQ, 0.5, 1.4);
    Lb = [LbP LbQ];
    Ub = [UbP UbQ];
    [bestX,bestFx] =
psox(@quantShammasPadePoly,Lb,Ub,1000,5000,true);

    SSE = quantShammasPadePoly(bestX);
    orderTot = orderP + orderQ;
    % calculate adjusted value of the coefficient of
determination
    glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-orderTot-1);
    if bestR2 < glbRsqr
      bestR2 = glbRsqr;
      bestP = orderP;
      bestQ = orderQ;
      bestQSPcoeff = QSPcoeff;
      zBestX = bestX;
      bestYCalc = yCalc;
    end
  end
end
fprintf("Best orderP = %d\n", bestP);
fprintf("Best orderQ = %d\n", bestQ);
fprintf("Adjusted Rsqr = %f\n", bestR2);
fprintf("Quantum Shammas Pade Polynomial Powers\n");
```



```
zBestX
fprintf("Quantum Shammas Pade Polynomial Coefficients\n");
QSPcoeff = bestQSPcoeff'
fprintf("\nRegular Pade polynomial fit\n");
[c,r,yPoly] = padefit(xData,yData,bestP,bestQ);
% calculate adjusted value of the coefficient of determination
orderTot = bestP + bestQ;
r = 1 - (1 - r)*(n-1)/(n-orderTot-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,bestYCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = zBestX;
Coeff = 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 = [bestR2 r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
padeOrderP = bestP;
padeOrderQ = bestQ;
orderP = [padeOrderP];
T = array2table(orderP);
writetable(T,xlFile,"Sheet","Sheet1","Range","D10");
orderQ = [padeOrderQ];
T = array2table(orderQ);
writetable(T,xlFile,"Sheet","Sheet1","Range","E10");
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
```

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
1.363230829	2.399295938	3.392974814	4.398466398	5.366457316
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
1.000231455	0.039591041	-0.407745936	0.162978281	-0.024615094
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.999702219	-0.301088582	-0.200176789	0.087991293	-0.011499866
r_sqr1	r_sqr2		orderP	orderQ
0.995853071	0.999999835		6	4

Table 2a. Summary of the results appearing in file besselj_0_x_pade_pso_run2.xlsx.

QSPpwr6	QSPpwr7	QSPpwr8	QSPpwr9	QSPpwr10	
6.348907089	0.652651608	2.383042018	3.21625426	3.852961147	
QSPcoeff6	QSPcoeff7	QSPcoeff8	QSPcoeff9	QSPcoeff10	QSPcoeff11
0.001761629	-4.31097E-05	0.028611265	-0.030784382	0.010288228	-0.001312341
Coeff6	Coeff7	Coeff8	Coeff9	Coeff10	Coeff11
0.000548373	-6.06546E-06	-0.305855713	0.064629629	-0.006867474	0.000353912

Table 2b. Summary of the results appearing in file besselj_0_x_pade_pso_run2.xlsx.

The second row shows the powers for the fitted Quantum Shammas Padé Polynomial. The fifth row shows the intercept (below QSPcoeff1) and to its right the coefficients for the various Quantum Shammas Padé 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 Padé Polynomial. The cell under r_sqr2 shows the adjusted coefficient of determination for the fitted classical polynomial. . The optimum orders for P(x) and Q(x) are 6 and 4, respectively. The adjusted coefficient of determination for the fitted Quantum Shammas Padé Polynomial is lower than the one for the classical Padé polynomial. This condition indicates that the Quantum Shammas Padé Polynomial did not perform as well for the above example with its extended x range of (0, 10).

Here is the graph (from file besselj_0_x_pade_pso_run2.jpg) for the Bessel function and the two fitted polynomials:

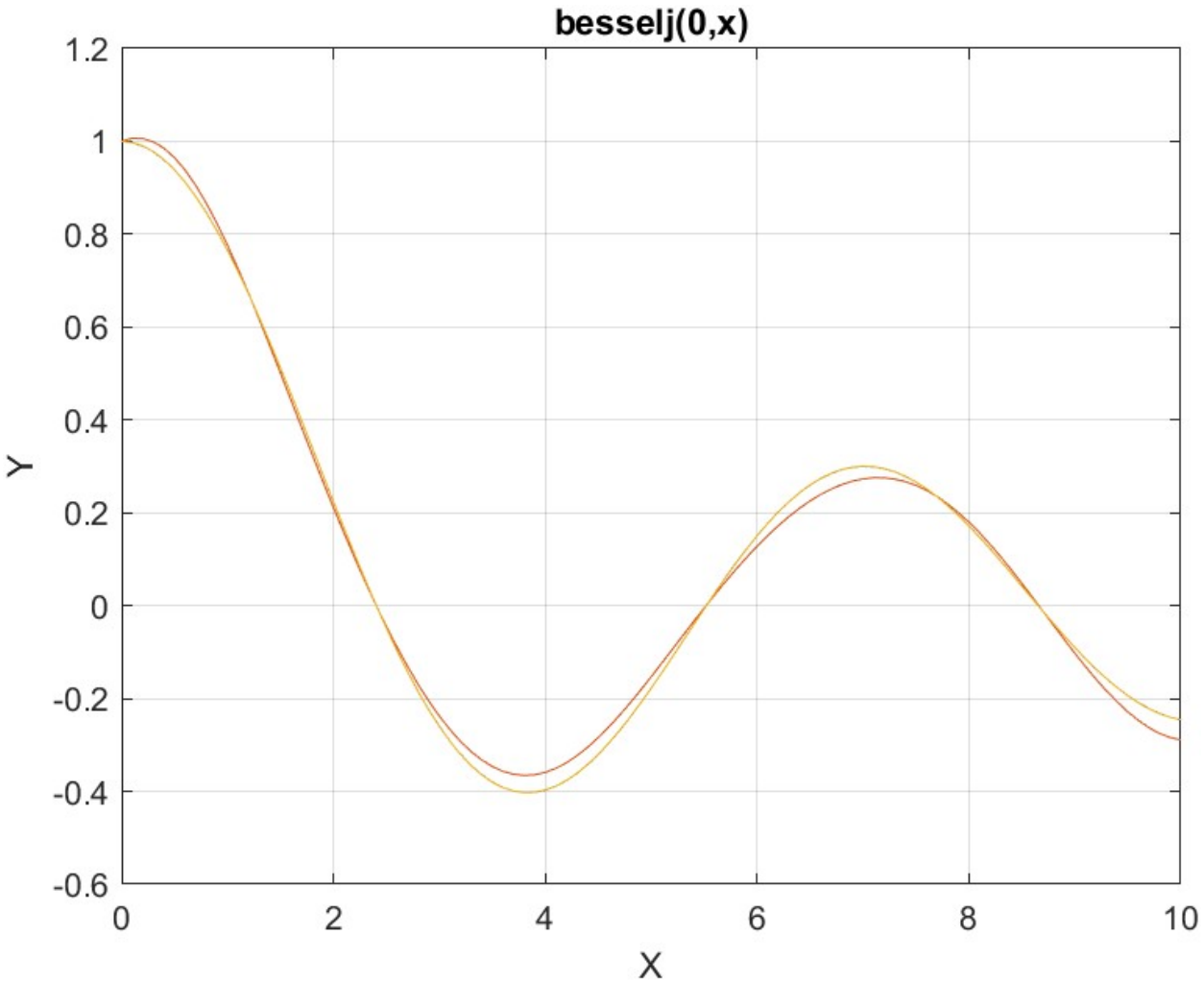

Figure 2. The graph from file besselj_0_x_pade_pso_run2.jpg.

The above graphs let you detect some deviations between the Bessel function and the Quantum Shammas Padé Polynomial. This is not unexpected since I have doubled the upper limit of the range of x from 5 to 10.

Testing Bessel Function Fit with Random Search Optimization-Run1

The next MATLAB script (found in file testPadeBessel1Random.m) tests fitting Bessel J(0, x) for x in the range (0, 5) and samples at 0.1 steps.

```
clc
clear
close all
```



```
global xData yData yCalc glbRsqr QSPcoeff
global orderP orderQ
zFilename = "besselj_0_x_pade_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);
bestR2 = -1;
for orderP=2:6
  for orderQ=2:6
    fprintf("\norderP = %d, orderQ=%d\n\n", orderP,orderQ)

  [LbP,UbP] = makeLimits(orderP, 0.5, 1.4);
  [LbQ,UbQ] = makeLimits(orderQ, 0.5, 1.4);
  [bestX,bestFx]
=randomSearchPade(@quantShammasPadePoly,LbP,UbP,LbQ,UbQ,2000000)
;

  SSE = quantShammasPadePoly(bestX);
  orderTot = orderP + orderQ;
  % calculate adjusted value of the coefficient of determination
  glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-orderTot-1);
    if bestR2 < glbRsqr
      bestR2 = glbRsqr;
      bestP = orderP;
      bestQ = orderQ;
      bestQSPcoeff = QSPcoeff;
      zBestX = bestX;
      bestYCalc = yCalc;
    end
  end
end
fprintf("Best orderP = %d\n", bestP);
fprintf("Best orderQ = %d\n", bestQ);
fprintf("Adjusted Rsqr = %f\n", bestR2);
fprintf("Quantum Shammas Pade Polynomial Powers\n");
zBestX
fprintf("Quantum Shammas Pade Polynomial Coefficients\n");
```



```
QSPcoeff = bestQSPcoeff'
fprintf("\nRegular Pade polynomial fit\n");
[c,r,yPoly] = padefit(xData,yData,bestP,bestQ);
% calculate adjusted value of the coefficient of determination
orderTot = bestP + bestQ;
r = 1 - (1 - r)*(n-1)/(n-orderTot-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,bestYCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = zBestX;
Coeff = 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 = [bestR2 r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
padeOrderP = bestP;
padeOrderQ = bestQ;
orderP = [padeOrderP];
T = array2table(orderP);
writetable(T,xlFile,"Sheet","Sheet1","Range","D10");
orderQ = [padeOrderQ];
T = array2table(orderQ);
writetable(T,xlFile,"Sheet","Sheet1","Range","E10");
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
```

The above script uses random search optimization by calling function randomSearchPade(). 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
1.390306124	1.770528854	2.96692345	3.936541494	5.393272519
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.999930459	0.056566885	-0.26632474	-0.055616915	0.033081313
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
1.000005356	-0.098317992	-0.231950891	0.022936247	0.012914014
r_sqr1	r_sqr2		orderP	orderQ
0.999999958	1		6	2

Table 3a. Summary of the results appearing in file besselj_0_x_pade_random_run1.xlsx.

QSPpwr6	QSPpwr7	QSPpwr8	
6.155723245	0.684728818	2.193382785	
QSPcoeff6	QSPcoeff7	QSPcoeff8	QSPcoeff9
-0.002592993	0.000302713	0.000125688	-1.41496E-05
Coeff6	Coeff7	Coeff8	Coeff9
-0.002421054	0.000112057	-0.098155261	0.017236385

Table 3b. Summary of the results appearing in file besselj_0_x_pade_random_run1.xlsx.

The above table shows that the adjusted coefficient of determination for the Quantum Shammas Padé Polynomial falls slightly short of that of the classical Padé polynomial. Both coefficients are good values.

Here is the graph (from file besselj_0_x_pade_random_run1.jpg) for the Bessel function and the two fitted polynomials:

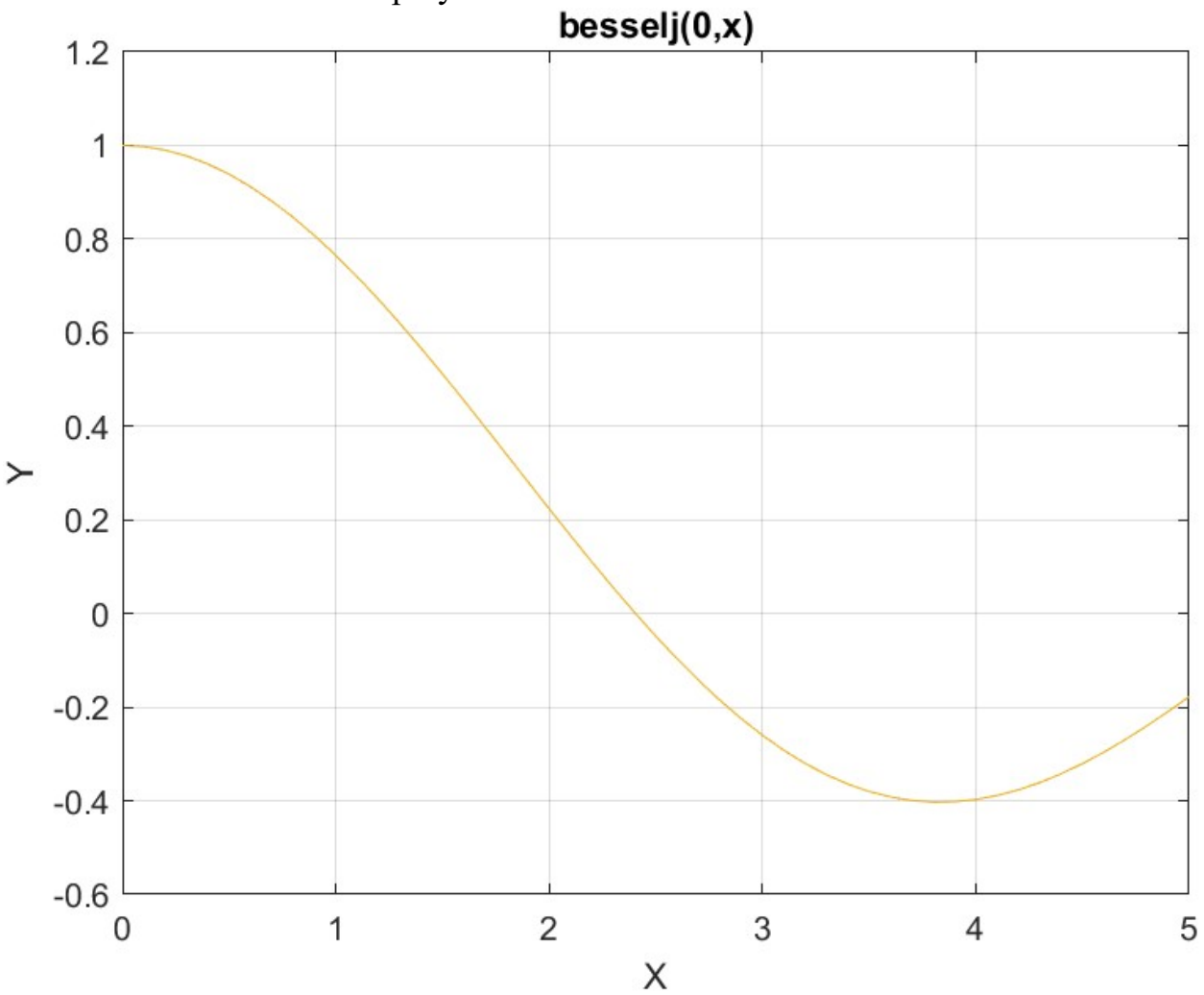

Figure 3. The graph from file besselj_0_x_pade_random_run1.jpg.

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

Testing Bessel Function Fit with Random Search Optimization-Run2

The next MATLAB script (found in file testPadeBessel2Random.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
global orderP orderQ
zFilename = "besselj_0_x_pade_rand_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);
bestR2 = -1;
for orderP=2:6
  for orderQ=2:6
    fprintf("\norderP = %d, orderQ=%d\n\n", orderP,orderQ)
    [LbP,UbP] = makeLimits(orderP, 0.5, 1.4);
    [LbQ,UbQ] = makeLimits(orderQ, 0.5, 1.4);
    [bestX,bestFx] =
randomSearchPade(@quantShammasPadePoly,LbP,UbP,LbQ,UbQ,1000000);

    SSE = quantShammasPadePoly(bestX);
    orderTot = orderP + orderQ;
    % calculate adjusted value of the coefficient of
determination
    glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-orderTot-1);
    if bestR2 < glbRsqr
      bestR2 = glbRsqr;
      bestP = orderP;
      bestQ = orderQ;
      bestQSPcoeff = QSPcoeff;
      zBestX = bestX;
      bestYCalc = yCalc;
    end
  end
end
fprintf("Best orderP = %d\n", bestP);
fprintf("Best orderQ = %d\n", bestQ);
fprintf("Quantum Shammas Pade Polynomial Powers\n");
zBestX
fprintf("Quantum Shammas Pade Polynomial Coefficients\n");
QSPcoeff = bestQSPcoeff'
fprintf("\nRegular Pade polynomial fit\n");
[c,r,yPoly] = padefit(xData,yData,bestP,bestQ);
% calculate adjusted value of the coefficient of determination
orderTot = bestP + bestQ;
```



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

QSPpwr = zBestX;
Coeff = 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 = [bestR2 r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
padeOrderP = bestP;
padeOrderQ = bestQ;
orderP = [padeOrderP];
T = array2table(orderP);
writetable(T,xlFile,"Sheet","Sheet1","Range","D10");
orderQ = [padeOrderQ];
T = array2table(orderQ);
writetable(T,xlFile,"Sheet","Sheet1","Range","E10");
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
```


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
1.336492876	2.378417564	3.201722897	4.309615241	5.285016117
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
1.000836962	0.058982337	-0.487040509	0.215890353	-0.022839035
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.999702219	-0.301088582	-0.200176789	0.087991293	-0.011499866
r_sqr1	r_sqr2		orderP	orderQ
0.994049228	0.999999835		6	4

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

QSPpwr6	QSPpwr7	QSPpwr8	QSPpwr9	QSPpwr10	
6.303030899	0.547493375	2.169152217	3.023661526	3.97790015	
QSPcoeff6	QSPcoeff7	QSPcoeff8	QSPcoeff9	QSPcoeff10	QSPcoeff11
0.001559455	-3.20626E-05	0.030021153	-0.038832478	-0.009962765	-0.000479351
Coeff6	Coeff7	Coeff8	Coeff9	Coeff10	Coeff11
0.000548373	-6.06546E-06	-0.305855713	0.064629629	-0.006867474	0.000353912

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

The above table shows the adjusted coefficient of determination for the Quantum Shammas Padé Polynomial is reasonably fine but less than that of the 6^{th} order classical polynomial.

Here is the graph (from file besselj_0_x_pade_rand_run2.jpg) for the Bessel function and the two fitted polynomials:

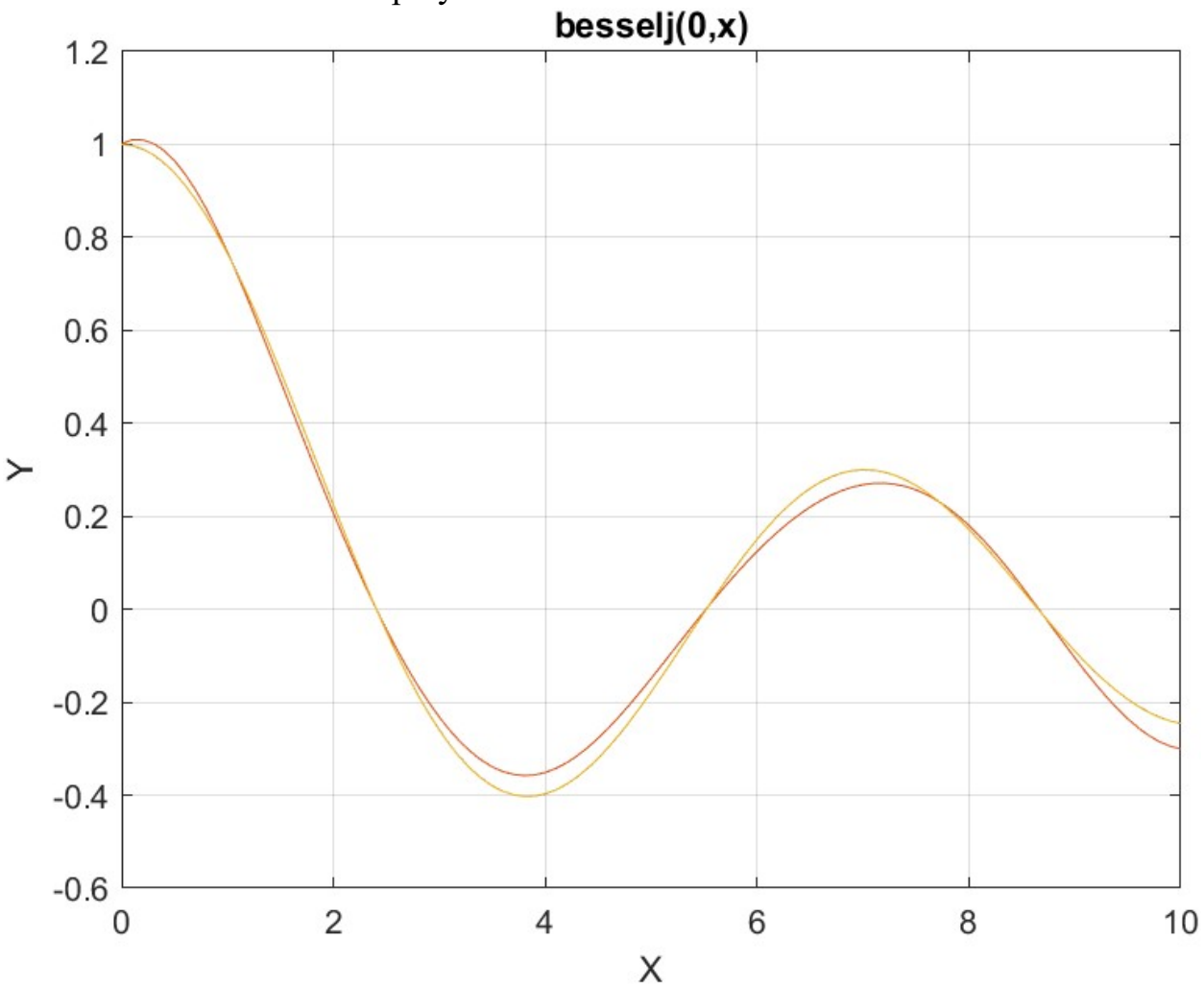

Figure 4. The graph from file besselj_0_x_pade_rand_run2.jpg.

The above graphs let you detect some deviations between the fitted Quantum Shammas Padé polynomial and the Bessel function curve.

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.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
```



```
global orderP orderQ
zFilename = "besselj_0_x_pade_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);
bestR2 = -1;
for orderP=2:6
  for orderQ=2:6
    fprintf("\norderP = %d, orderQ=%d\n\n", orderP,orderQ)
    [LbP,UbP] = makeLimits(orderP, 0.5, 1.4);
    [LbQ,UbQ] = makeLimits(orderQ, 0.5, 1.4);
    [bestX,bestFx] =
haltonRandomSearchPade(@quantShammasPadePoly,LbP,UbP,LbQ,UbQ,200
0000);

    SSE = quantShammasPadePoly(bestX);
    orderTot = orderP + orderQ;
    % calculate adjusted value of the coefficient of
determination
    glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-orderTot-1);
    if bestR2 < glbRsqr
      bestR2 = glbRsqr;
      bestP = orderP;
      bestQ = orderQ;
      bestQSPcoeff = QSPcoeff;
      zBestX = bestX;
      bestYCalc = yCalc;
    end
  end
end
fprintf("Best orderP = %d\n", bestP);
fprintf("Best orderQ = %d\n", bestQ);
fprintf("Adjusted Rsqr = %f\n", bestR2);
fprintf("Quantum Shammas Pade Polynomial Powers\n");
zBestX
fprintf("Quantum Shammas Pade Polynomial Coefficients\n");
QSPcoeff = bestQSPcoeff'
```



```
fprintf("\nRegular Pade polynomial fit\n");
[c,r,yPoly] = padefit(xData,yData,bestP,bestQ);
% calculate adjusted value of the coefficient of determination
orderTot = bestP + bestQ;
r = 1 - (1 - r)*(n-1)/(n-orderTot-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,bestYCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = zBestX;
Coeff = 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 = [bestR2 r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
padeOrderP = bestP;
padeOrderQ = bestQ;
orderP = [padeOrderP];
T = array2table(orderP);
writetable(T,xlFile,"Sheet","Sheet1","Range","D10");
orderQ = [padeOrderQ];
T = array2table(orderQ);
writetable(T,xlFile,"Sheet","Sheet1","Range","E10");

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
```

The above code is like the one in first random search optimization program. The main difference is that the above code uses functions that involve the Halton quasi-random sequence. Running the above code produces the following Excel table:

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5
1.390306124	1.770528854	2.96692345	3.936541494	5.393272519
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.999930459	0.056566885	-0.26632474	-0.055616915	0.033081313
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
1.000005356	-0.098317992	-0.231950891	0.022936247	0.012914014
r_sqr1	r_sqr2		orderP	orderQ
0.999999958	1		6	2

Table 5a. Summary of the results appearing in file besselj_0_x_pade_halton_random_run1.xlsx.

QSPpwr6	QSPpwr7	QSPpwr8	
6.155723245	0.684728818	2.193382785	
QSPcoeff6	QSPcoeff7	QSPcoeff8	QSPcoeff9
-0.002592993	0.000302713	0.000125688	-1.41496E-05
Coeff6	Coeff7	Coeff8	Coeff9
-0.002421054	0.000112057	-0.098155261	0.017236385

Table 5b. Summary of the results appearing in file besselj_0_x_pade_halton_random_run1.xlsx.

The above table shows the adjusted coefficient of determination for the Quantum Shammas Padé Polynomial is slightly less than that for the classical Padé polynomial. Both coefficients have very good values. Using the Halton sequence gives surprisingly good results.

Here is the graph (from file besselj_0_x_pade_halton_random_run1.jpg) for the Bessel function and the two fitted polynomials:

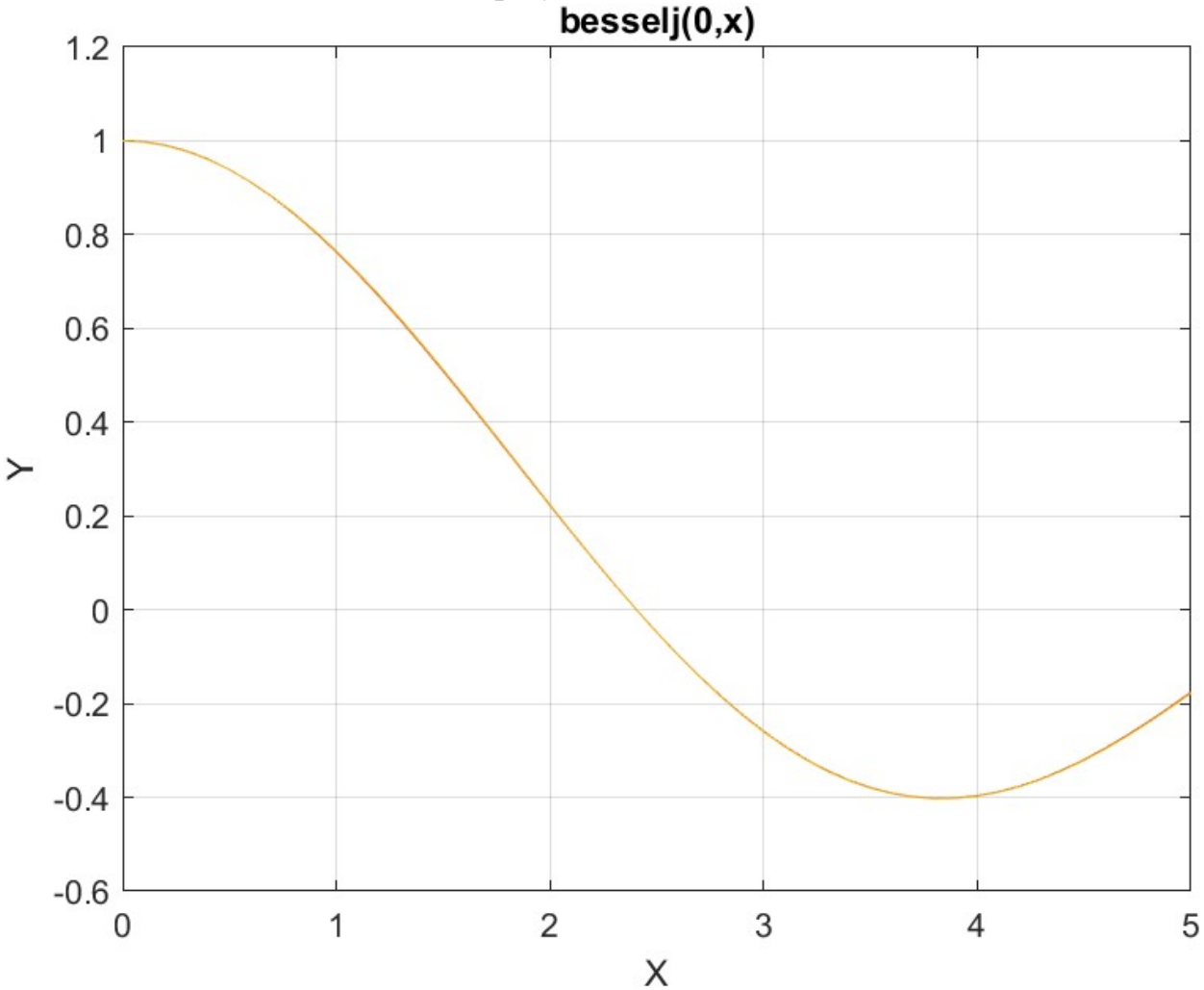

Figure 5. The graph from file besselj_0_x_pade_halton_random_run1.jpg.

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.


```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
global orderP orderQ
zFilename = "besselj_0_x_pade_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);
bestR2 = -1;
for orderP=2:6
  for orderQ=2:6
    fprintf("\norderP = %d, orderQ=%d\n\n", orderP,orderQ)
    [LbP,UbP] = makeLimits(orderP, 0.5, 1.4);
    [LbQ,UbQ] = makeLimits(orderQ, 0.5, 1.4);
    [bestX,bestFx] =
haltonRandomSearchPade(@quantShammasPadePoly,LbP,UbP,LbQ,UbQ,200
0000);

    SSE = quantShammasPadePoly(bestX);
    orderTot = orderP + orderQ;
    % calculate adjusted value of the coefficient of
determination
    glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-orderTot-1);
    if bestR2 < glbRsqr
      bestR2 = glbRsqr;
      bestP = orderP;
      bestQ = orderQ;
      bestQSPcoeff = QSPcoeff;
      zBestX = bestX;
      bestYCalc = yCalc;
    end
  end
end
fprintf("Best orderP = %d\n", bestP);
fprintf("Best orderQ = %d\n", bestQ);
```



```
fprintf("Adjusted Rsqr = %f\n", bestR2);
fprintf("Quantum Shammas Pade Polynomial Powers\n");
zBestX
fprintf("Quantum Shammas Pade Polynomial Coefficients\n");
QSPcoeff = bestQSPcoeff'
fprintf("\nRegular Pade polynomial fit\n");
[c,r,yPoly] = padefit(xData,yData,bestP,bestQ);
% calculate adjusted value of the coefficient of determination
orderTot = bestP + bestQ;
r = 1 - (1 - r)*(n-1)/(n-orderTot-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,bestYCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = zBestX;
Coeff = 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 = [bestR2 r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
padeOrderP = bestP;
padeOrderQ = bestQ;
orderP = [padeOrderP];
T = array2table(orderP);
writetable(T,xlFile,"Sheet","Sheet1","Range","D10");
orderQ = [padeOrderQ];
T = array2table(orderQ);
writetable(T,xlFile,"Sheet","Sheet1","Range","E10");

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
```

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
1.365728088	2.034561706	3.396616124	4.210982716	5.397496554
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.998944114	0.220249344	-0.548033523	0.129153115	-0.027570666
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.999702219	-0.301088582	-0.200176789	0.087991293	-0.011499866
r_sqr1	r_sqr2		orderP	orderQ
0.993448058	0.999999835		6	4

Table 6a. Summary of the results appearing in file besselj_0_x_pade_halton_random_run2.xlsx.

QSPpwr6	QSPpwr7	QSPpwr8	QSPpwr9	QSPpwr10	
6.260899353	0.72085473	1.944214989	2.741042714	4.380151161	
QSPcoeff6	QSPcoeff7	QSPcoeff8	QSPcoeff9	QSPcoeff10	QSPcoeff11
0.000995209	-3.80536E-05	0.058323203	-0.062682354	0.013925476	-8.78517E-05
Coeff6	Coeff7	Coeff8	Coeff9	Coeff10	Coeff11

0.000548373	-6.06546E-06	-0.305855713	0.064629629	-0.006867474	0.000353912

Table 6b. Summary of the results appearing in file besselj_0_x_pade_halton_random_run2.xlsx.

The above table shows the adjusted coefficient of determination for the Quantum Shammas Padé Polynomial is less than that for the classical Padé polynomial.

Here is the graph (from file besselj_0_x_pade_halton_random_run2.jpg) for the Bessel function and the two fitted polynomials:

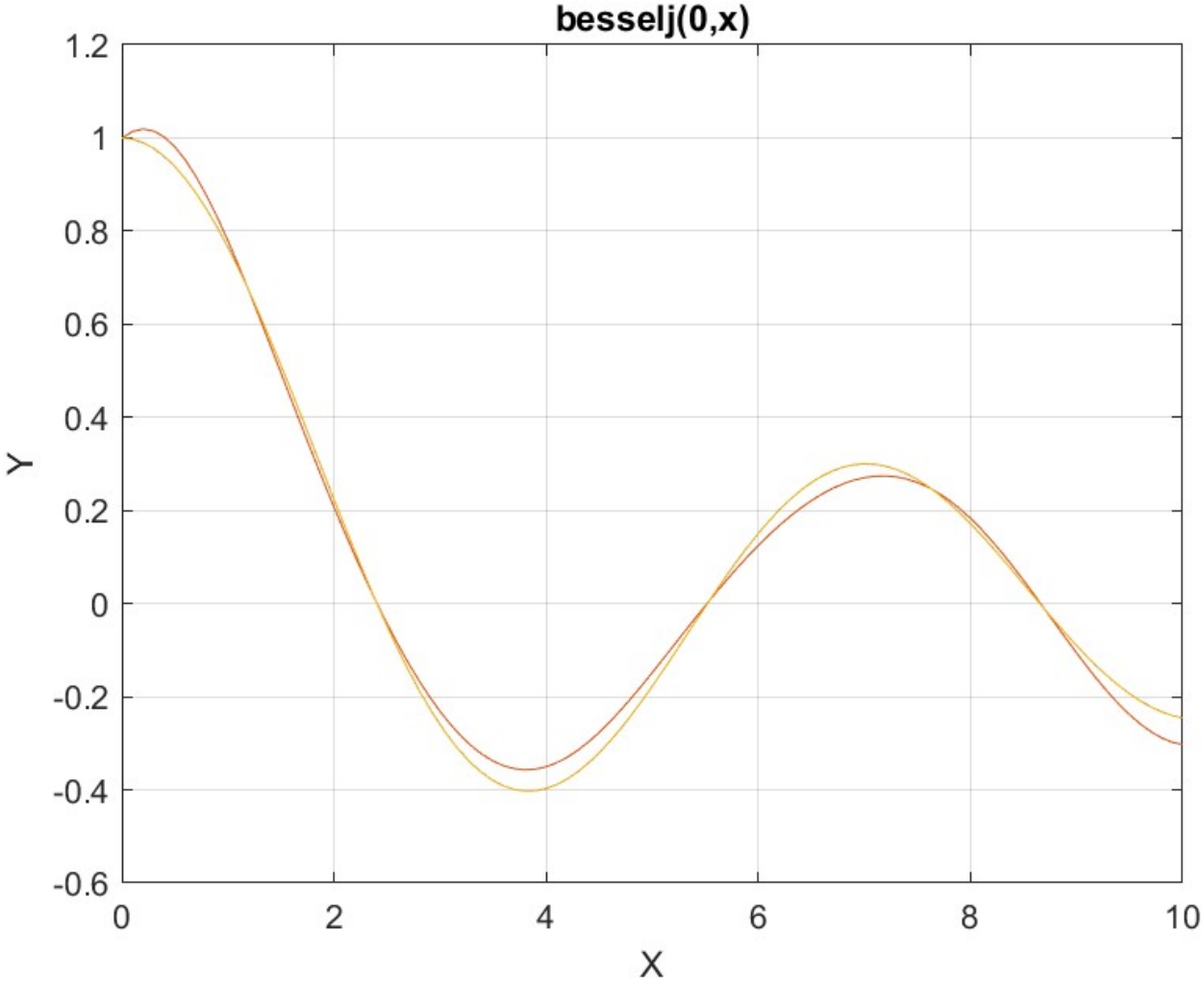

Figure 6. The graph from file besselj_0_x_pade_halton_random_run2.jpg.

The curves in the above figure show some deviations between the Quantum Shammas Padé Polynomial and 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.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
global orderP orderQ
```



```
zFilename = "besselj_0_x_pade_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);
bestR2 = -1;
for orderP=2:6
  for orderQ=2:6
    fprintf("\norderP = %d, orderQ=%d\n\n", orderP,orderQ)
  [LbP,UbP] = makeLimits(orderP, 0.5, 1.4);
  [LbQ,UbQ] = makeLimits(orderQ, 0.5, 1.4);
  [bestX,bestFx] =
sobolRandomSearchPade(@quantShammasPadePoly,LbP,UbP,LbQ,UbQ,2000
000);

  SSE = quantShammasPadePoly(bestX);
  orderTot = orderP + orderQ;
  % calculate adjusted value of the coefficient of determination
  glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-orderTot-1);
    if bestR2 < glbRsqr
      bestR2 = glbRsqr;
      bestP = orderP;
      bestQ = orderQ;
      bestQSPcoeff = QSPcoeff;
      zBestX = bestX;
      bestYCalc = yCalc;
    end
  end
end
fprintf("Best orderP = %d\n", bestP);
fprintf("Best orderQ = %d\n", bestQ);
fprintf("Adjusted Rsqr = %f\n", bestR2);
fprintf("Quantum Shammas Pade Polynomial Powers\n");
zBestX
fprintf("Quantum Shammas Pade Polynomial Coefficients\n");
QSPcoeff = bestQSPcoeff'
fprintf("\nRegular Pade polynomial fit\n");
[c,r,yPoly] = padefit(xData,yData,bestP,bestQ);
```



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

QSPpwr = zBestX;
Coeff = 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 = [bestR2 r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
padeOrderP = bestP;
padeOrderQ = bestQ;
orderP = [padeOrderP];
T = array2table(orderP);
writetable(T,xlFile,"Sheet","Sheet1","Range","D10");
orderQ = [padeOrderQ];
T = array2table(orderQ);
writetable(T,xlFile,"Sheet","Sheet1","Range","E10");
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
```


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	QSPpwr5
1.153894665	1.780472026	3.137612519	4.31838467	4.917837508
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.999725758	0.030190233	-0.244364973	-0.048238113	0.04342938
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
1.000005356	-0.098317992	-0.231950891	0.022936247	0.012914014
r_sqr1	r_sqr2		orderP	orderQ
0.999999944	1		6	2

Table 7a. Summary of the results appearing in file besselj_0_x_pade_sobol_random_run1.xlsx.

QSPpwr6	QSPpwr7	QSPpwr8	
5.717753613	1.025201805	2.288174538	
QSPcoeff6	QSPcoeff7	QSPcoeff8	QSPcoeff9
-0.016593335	0.001064939	-7.97575E-06	-1.48801E-05
Coeff6	Coeff7	Coeff8	Coeff9
-0.002421054	0.000112057	-0.098155261	0.017236385

Table 7b. Summary of the results appearing in file besselj_0_x_pade_sobol_random_run1.xlsx.

The above table shows that the adjusted coefficient of determination for the Quantum Shammas Padé Polynomial is a bit less than that for the classical Padé

polynomial. Using the Sobol sequence gives surprisingly good results (but still falling a tiny bit below the classical Padé polynomial). I also suspect using two million iterations has something to do with it.

Here is the graph (from file besselj_0_x_pade_sobol_random_run1.jpg) for the Bessel function and the two fitted polynomials:

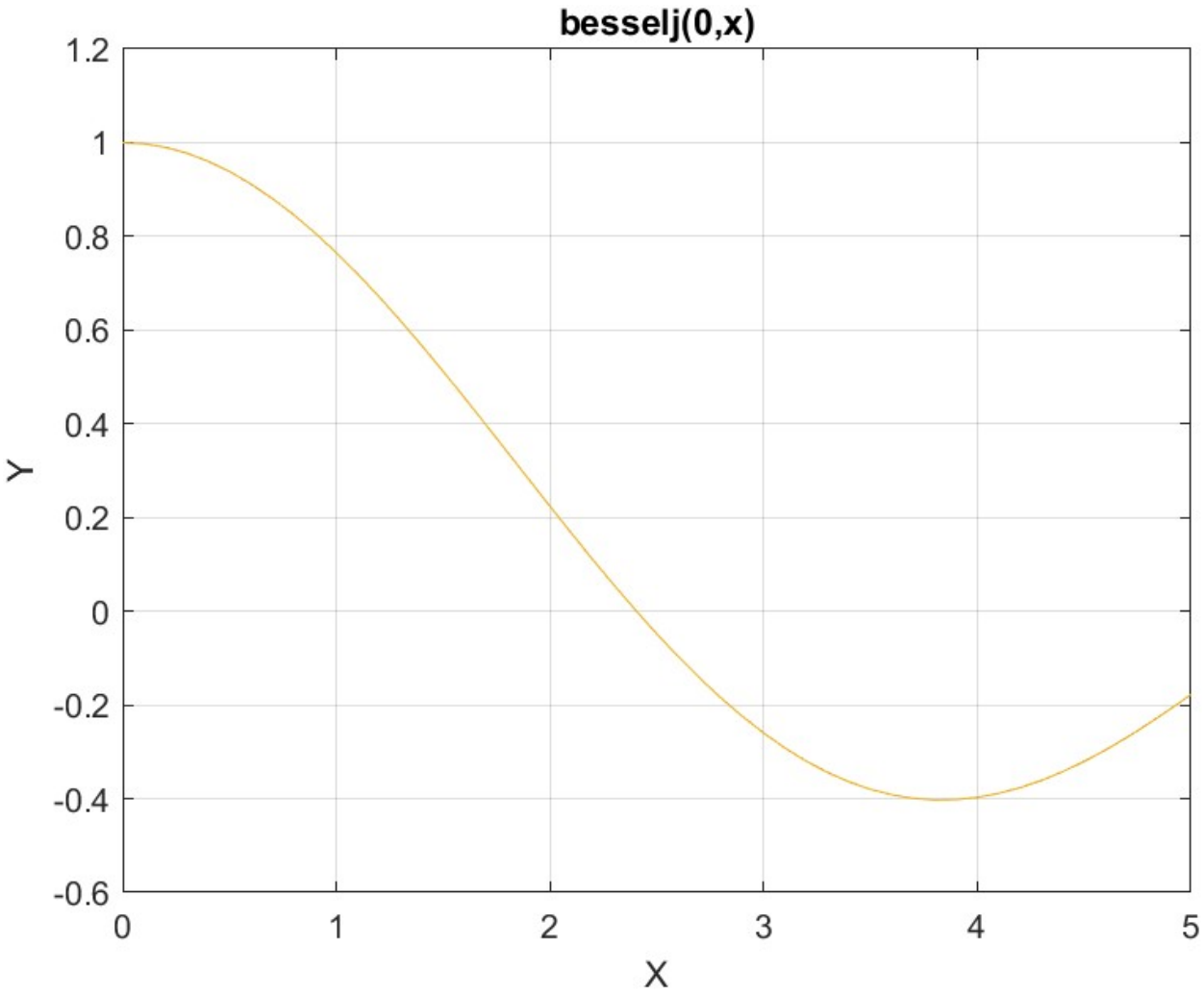

Figure 7. The graph from file besselj_0_x_pade_sobol_random_run1.jpg.

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

Testing Bessel Function Fit with Sobol Random Search Optimization-Run2

The next MATLAB script (found in file testBessel1Sobo2.m) tests fitting Bessel J(0, x) for x in the range (0, 5) and samples at 0.1 steps.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
```

```
global orderP orderQ
zFilename = "besselj_0_x_pade_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);
bestR2 = -1;
for orderP=2:6
  for orderQ=2:6
    fprintf("\norderP = %d, orderQ=%d\n\n", orderP,orderQ);
    [LbP,UbP] = makeLimits(orderP, 0.5, 1.4);
    [LbQ,UbQ] = makeLimits(orderQ, 0.5, 1.4);
    [bestX,bestFx] =
sobolRandomSearchPade(@quantShammasPadePoly,LbP,UbP,LbQ,UbQ,2000
000);

    SSE = quantShammasPadePoly(bestX);
    orderTot = orderP + orderQ;
    % calculate adjusted value of the coefficient of
determination
    glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-orderTot-1);
    if bestR2 < glbRsqr
      bestR2 = glbRsqr;
      bestP = orderP;
      bestQ = orderQ;
      bestQSPcoeff = QSPcoeff;
      zBestX = bestX;
      bestYCalc = yCalc;
    end
  end
end
fprintf("Best orderP = %d\n", bestP);
fprintf("Best orderQ = %d\n", bestQ);
fprintf("Adjusted Rsqr = %f\n", bestR2);
fprintf("Quantum Shammas Pade Polynomial Powers\n");
zBestX
fprintf("Quantum Shammas Pade Polynomial Coefficients\n");
QSPcoeff = bestQSPcoeff'
```



```
fprintf("\nRegular Pade polynomial fit\n");
[c,r,yPoly] = padefit(xData,yData,bestP,bestQ);
% calculate adjusted value of the coefficient of determination
orderTot = bestP + bestQ;
r = 1 - (1 - r)*(n-1)/(n-orderTot-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,bestYCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = zBestX;
Coeff = 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 = [bestR2 r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
padeOrderP = bestP;
padeOrderQ = bestQ;
orderP = [padeOrderP];
T = array2table(orderP);
writetable(T,xlFile,"Sheet","Sheet1","Range","D10");
orderQ = [padeOrderQ];
T = array2table(orderQ);
writetable(T,xlFile,"Sheet","Sheet1","Range","E10");
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
```

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
1.167578274	2.394169049	3.327742594	4.322646743	5.309386266
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.999328741	0.026884337	-0.414412032	0.176873298	-0.025503539
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.999702219	-0.301088582	-0.200176789	0.087991293	-0.011499866
r_sqr1	r_sqr2		orderP	orderQ
0.994354415	0.999999835		6	4

Table 8a. Summary of the results appearing in file besselj_0_x_pade_sobol_random_run2.xlsx.

QSPpwr6	QSPpwr7	QSPpwr8	QSPpwr9	QSPpwr10	
6.246165644	0.610454044	2.313615873	3.06117802	3.770744714	
QSPcoeff6	QSPcoeff7	QSPcoeff8	QSPcoeff9	QSPcoeff10	QSPcoeff11
0.001671082	-4.52886E-05	0.022296393	-0.036390356	-0.012966551	-0.001205296
Coeff6	Coeff7	Coeff8	Coeff9	Coeff10	Coeff11
0.000548373	-6.06546E-06	-0.305855713	0.064629629	-0.006867474	0.000353912

Table 8b. Summary of the results appearing in file besselj_0_x_pade_sobol_random_run2.xlsx.

As expected, the adjusted coefficient of determination for the Quantum Shammas Padé Polynomial is lower than the one for classical Padé polynomial.

Here is the graph (from file besselj_0_x_pade_sobol_random_run2.jpg) for the Bessel function and the two fitted polynomials:

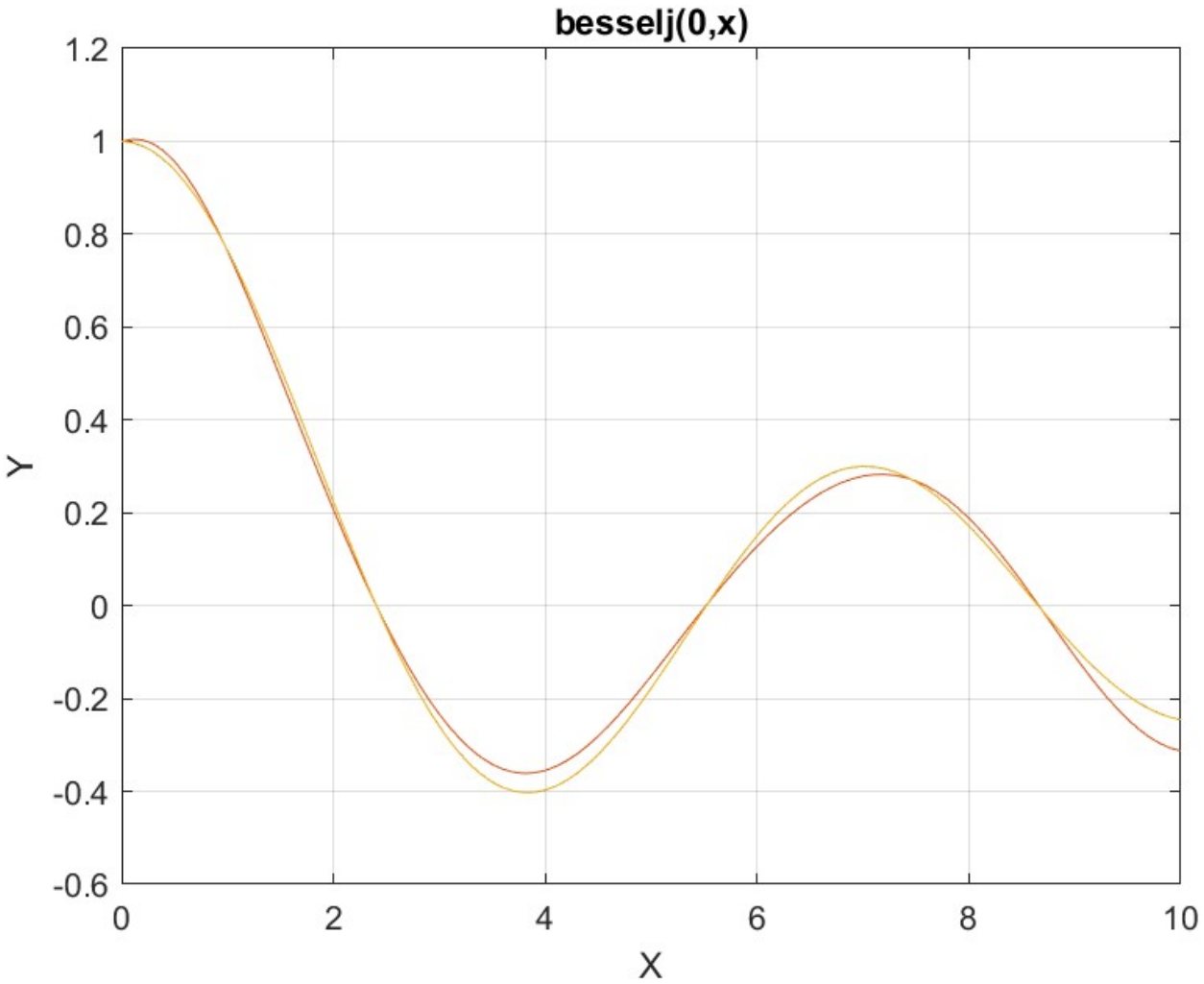

Figure 8. The graph from file besselj_0_x_pade_sobol_random_run2.jpg.

Again, the above curves show some deviations of fitted Quantum Padé Shammas Polynomial with 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 slightly inferior fittings for the Quantum Shammas Padé Polynomial than the classical Padé polynomials for the same orders of the numerator and denominator polynomials.

The next four subsections look at the curve fitting of ln(x) with values of (x-1) in the range of (1, 8).

Testing ln(x) Function Fit with PSO

The next MATLAB script (found in file testPadeLog1pso.m) tests fitting ln(x) vs (x-1) for x in the range (1, 8) and samples at 0.1 steps, and using the PSO method.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
global orderP orderQ
zFilename = "Ln_x_pade_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("%s\n", sEqn);
fprintf("x=1:0.1:8\n")
xData0= 1:0.1:8;
xData0 = xData0';
n = length(xData0);
yData = log(xData0);
xData = xData0 - 1;
bestR2 = -1;
for orderP=2:6
  for orderQ=2:6
    fprintf("\norderP = %d, orderQ=%d\n\n", orderP,orderQ)

    [LbP,UbP] = makeLimits(orderP, 0.5, 1.4);
    [LbQ,UbQ] = makeLimits(orderQ, 0.5, 1.4);
    Lb = [LbP LbQ];
    Ub = [UbP UbQ];
    [bestX,bestFx] =
psox(@quantShammasPadePoly,Lb,Ub,1000,5000,true);

    SSE = quantShammasPadePoly(bestX);
    orderTot = orderP + orderQ;
    % calculate adjusted value of the coefficient of
determination
    glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-orderTot-1);
    if bestR2 < glbRsqr
```



```
      bestR2 = glbRsqr;
      bestP = orderP;
      bestQ = orderQ;
      bestQSPcoeff = QSPcoeff;
      zBestX = bestX;
      bestYCalc = yCalc;
    end
  end
end
fprintf("Best orderP = %d\n", bestP);
fprintf("Best orderQ = %d\n", bestQ);
fprintf("Adjusted Rsqr = %f\n", bestR2);
fprintf("Quantum Shammas Pade Polynomial Powers\n");
zBestX
fprintf("Quantum Shammas Pade Polynomial Coefficients\n");
QSPcoeff = bestQSPcoeff'
fprintf("\nRegular Pade polynomial fit\n");
[c,r,yPoly] = padefit(xData,yData,bestP,bestQ);
% calculate adjusted value of the coefficient of determination
orderTot = bestP + bestQ;
r = 1 - (1 - r)*(n-1)/(n-orderTot-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData0,yData,xData0,bestYCalc,xData0,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = zBestX;
Coeff = 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 = [bestR2 r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
padeOrderP = bestP;
padeOrderQ = bestQ;
orderP = [padeOrderP];
T = array2table(orderP);
writetable(T,xlFile,"Sheet","Sheet1","Range","D10");
```



```
orderQ = [padeOrderQ];
T = array2table(orderQ);
writetable(T,xlFile,"Sheet","Sheet1","Range","E10");

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
```

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	QSPpwr5	QSPpwr6	
0.901873428	1.521923228	3.074908719	3.721794661	0.751341903	1.864273779	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
-0.008747042	0.923460085	-0.229931401	0.006460984	-0.00098337	-0.001003338	0.000117908
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
7.36623E-05	0.998443883	0.414032292	0.008299306	-0.000145407	0.907416923	0.142173842
r_sqr1	r_sqr2		orderP	orderQ		
0.999942529	0.999999999		4	2		

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

The adjusted coefficient of determination for the Quantum Shammas Padé Polynomial is fairly good but less than the one for the classical Padé polynomial.

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

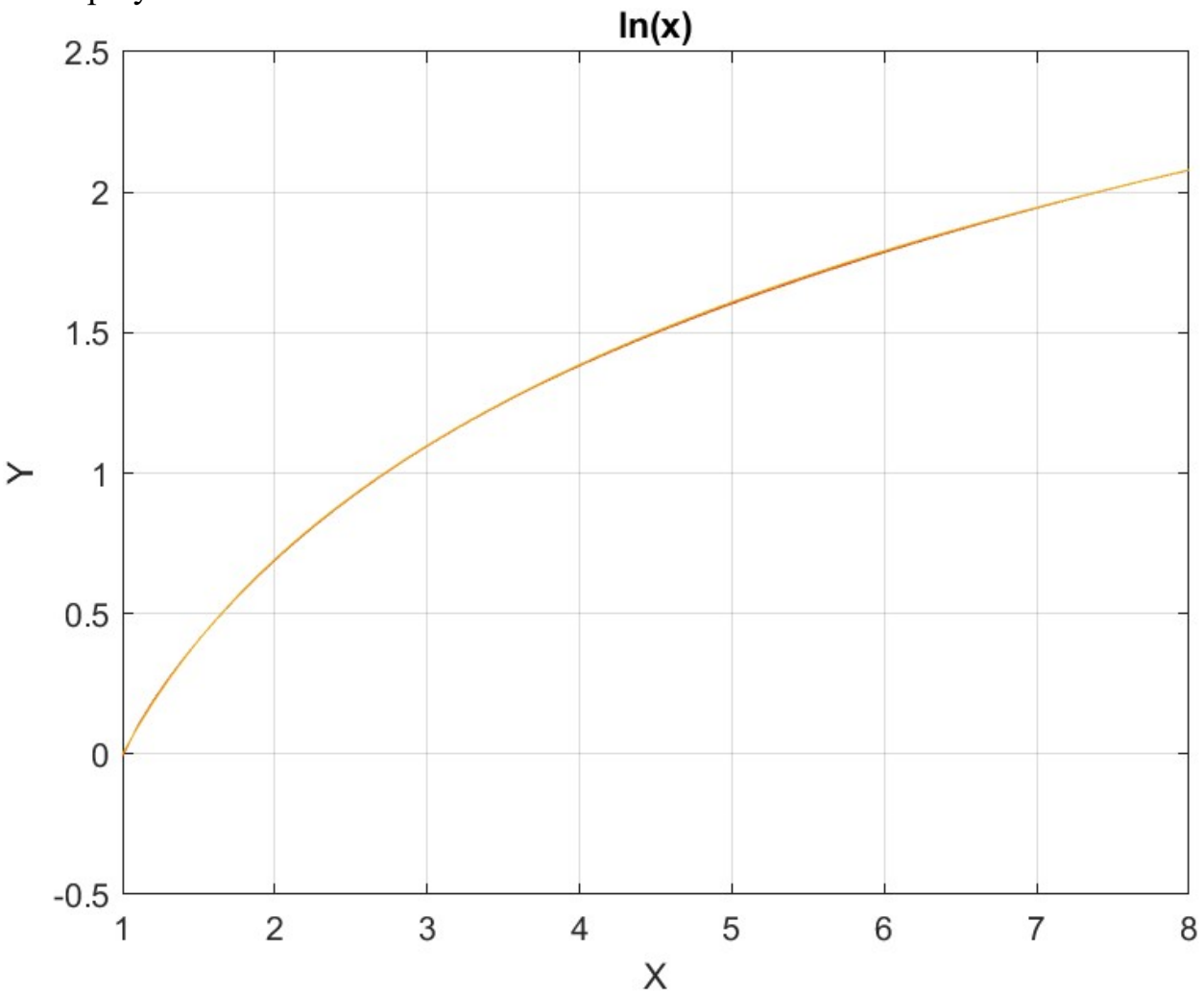

Figure 9. The graph from file Ln_x_pade_pso.jpg.

The above curves show that the two fitted polynomials appear close to the curve of the ln(x) function. The classical Padé polynomial fits the curve for the ln(x) function better.

Testing ln(x) Function Fit with Random Search Optimization

The next MATLAB script (found in file testPadeLog1Random.m) tests fitting ln(x) vs (x-1) for x in the range (1, 8) and samples at 0.1 steps, and using the random search optimization.

```
clc
clear
close all
```



```
global xData yData yCalc glbRsqr QSPcoeff
global orderP orderQ
zFilename = "Ln_x_pade_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=1:0.1:8\n")
xData0= 1:0.1:8;
xData0 = xData0';
n = length(xData0);
yData = log(xData0);
xData = xData0 - 1;
bestR2 = -1;
for orderP=2:6
  for orderQ=2:6
    fprintf("\norderP = %d, orderQ=%d\n\n", orderP,orderQ)
    [LbP,UbP] = makeLimits(orderP, 0.5, 1.4);
    [LbQ,UbQ] = makeLimits(orderQ, 0.5, 1.4);
    [bestX,bestFx]
=randomSearchPade(@quantShammasPadePoly,LbP,UbP,LbQ,UbQ,2000000)
;

    SSE = quantShammasPadePoly(bestX);
    orderTot = orderP + orderQ;
    % calculate adjusted value of the coefficient of
determination
    glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-orderTot-1);
    if bestR2 < glbRsqr
      bestR2 = glbRsqr;
      bestP = orderP;
      bestQ = orderQ;
      bestQSPcoeff = QSPcoeff;
      zBestX = bestX;
      bestYCalc = yCalc;
    end
  end
end
fprintf("Best orderP = %d\n", bestP);
fprintf("Best orderQ = %d\n", bestQ);
fprintf("Adjusted Rsqr = %f\n", bestR2);
fprintf("Quantum Shammas Pade Polynomial Powers\n");
```



```
zBestX
fprintf("Quantum Shammas Pade Polynomial Coefficients\n");
QSPcoeff = bestQSPcoeff'
fprintf("\nRegular Pade polynomial fit\n");
[c,r,yPoly] = padefit(xData,yData,bestP,bestQ);
% calculate adjusted value of the coefficient of determination
orderTot = bestP + bestQ;
r = 1 - (1 - r)*(n-1)/(n-orderTot-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData0,yData,xData0,bestYCalc,xData0,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = zBestX;
Coeff = 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 = [bestR2 r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
padeOrderP = bestP;
padeOrderQ = bestQ;
orderP = [padeOrderP];
T = array2table(orderP);
writetable(T,xlFile,"Sheet","Sheet1","Range","D10");
orderQ = [padeOrderQ];
T = array2table(orderQ);
writetable(T,xlFile,"Sheet","Sheet1","Range","E10");

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
```

The above code is similar to testPadeLog1pso.m except it uses different output filenames and calls the randomSearchPade() function for the curve fit optimization. The above code generates the following summary Excel table:

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	
0.616558708	1.816432877	1.044198314	1.783241833	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
-0.085684413	0.797396403	-0.013974731	-0.000548513	0.00036115
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.002100533	0.979757131	0.1660485	0.617667098	0.038547232
r_sqr1	r_sqr2		orderP	orderQ
0.998144944	0.999999557		2	2

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

The adjusted coefficient of determination for the Quantum Shammas Padé Polynomial is significantly less than the one for classical Padé polynomial.

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

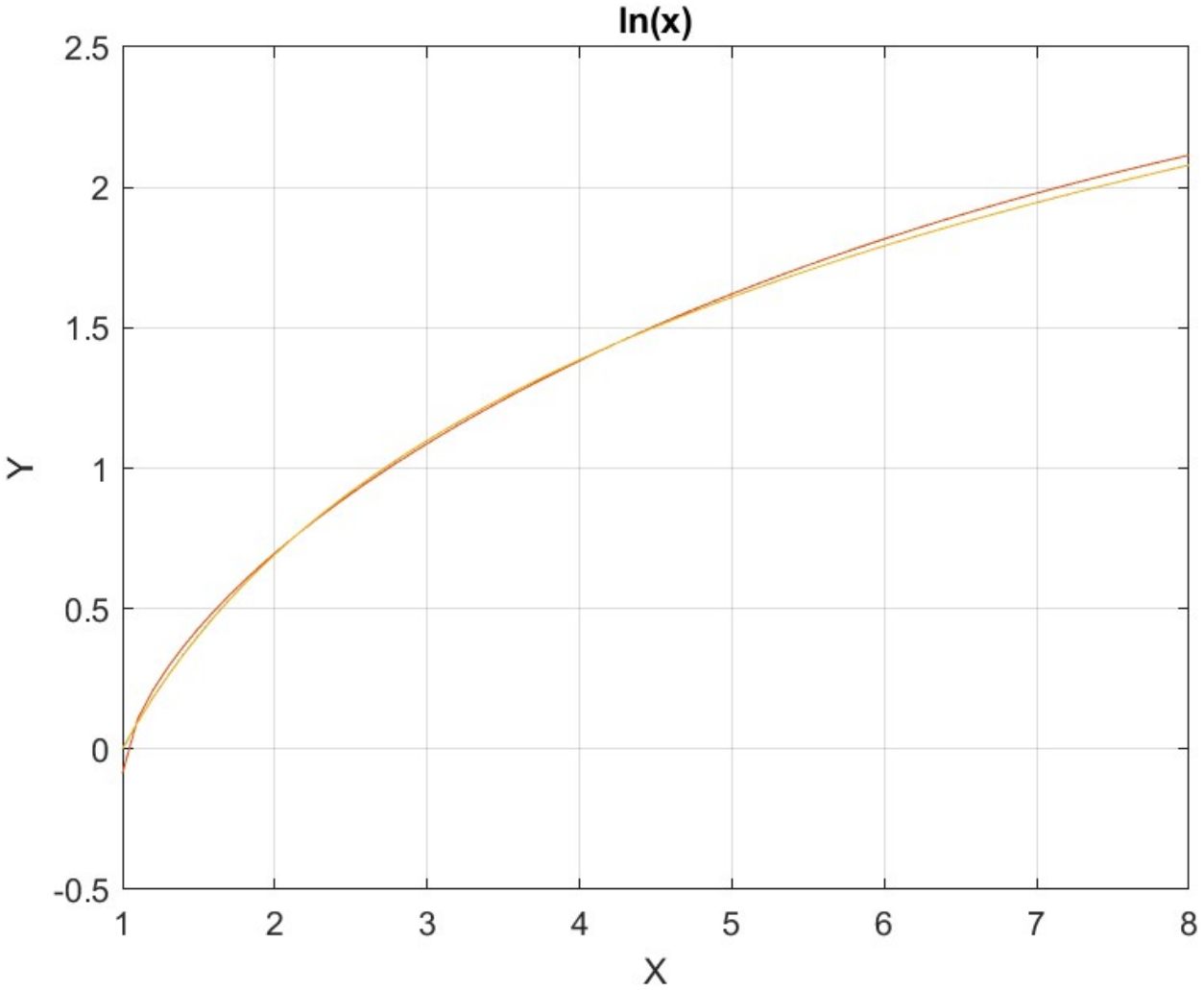

Figure 10. The graph from file Ln_x_pade_rand.jpg.

The above graph shows that the Quantum Shammas Padé Polynomial shows a significant deviation from the ln(x) curve!

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

The next MATLAB script (found in file testPadeLog1Halton.m) tests fitting ln(x) vs (x-1) for x in the range (1, 8) and samples at 0.1 steps, and using the Halton quasi-random search optimization.

```
clc
clear
close all
```



```
global xData yData yCalc glbRsqr QSPcoeff
global orderP orderQ
zFilename = "Ln_x_pade_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=1:0.1:8\n")
xData0= 1:0.1:8;
xData0 = xData0';
n = length(xData0);
yData = log(xData0);
xData = xData0 - 1;
bestR2 = -1;
for orderP=2:6
  for orderQ=2:6
    fprintf("\norderP = %d, orderQ=%d\n\n", orderP,orderQ)

    [LbP,UbP] = makeLimits(orderP, 0.5, 1.4);
    [LbQ,UbQ] = makeLimits(orderQ, 0.5, 1.4);
    [bestX,bestFx] =
haltonRandomSearchPade(@quantShammasPadePoly,LbP,UbP,LbQ,UbQ,200
0000);

    SSE = quantShammasPadePoly(bestX);
    orderTot = orderP + orderQ;
    % calculate adjusted value of the coefficient of
determination
    glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-orderTot-1);
    if bestR2 < glbRsqr
      bestR2 = glbRsqr;
      bestP = orderP;
      bestQ = orderQ;
      bestQSPcoeff = QSPcoeff;
      zBestX = bestX;
      bestYCalc = yCalc;
    end
  end
end
fprintf("Best orderP = %d\n", bestP);
fprintf("Best orderQ = %d\n", bestQ);
fprintf("Adjusted Rsqr = %f\n", bestR2);
fprintf("Quantum Shammas Pade Polynomial Powers\n");
```



```
zBestX
fprintf("Quantum Shammas Pade Polynomial Coefficients\n");
QSPcoeff = bestQSPcoeff'
fprintf("\nRegular Pade polynomial fit\n");
[c,r,yPoly] = padefit(xData,yData,bestP,bestQ);
% calculate adjusted value of the coefficient of determination
orderTot = bestP + bestQ;
r = 1 - (1 - r)*(n-1)/(n-orderTot-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData0,yData,xData0,bestYCalc,xData0,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = zBestX;
Coeff = 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 = [bestR2 r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
padeOrderP = bestP;
padeOrderQ = bestQ;
orderP = [padeOrderP];
T = array2table(orderP);
writetable(T,xlFile,"Sheet","Sheet1","Range","D10");
orderQ = [padeOrderQ];
T = array2table(orderQ);
writetable(T,xlFile,"Sheet","Sheet1","Range","E10");
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
```

The above file generates the following Excel table summary.

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	
0.766528247	1.676079885	2.626385714	1.035615327	2.224202861	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6
-0.032263692	0.800174259	-0.079278758	0.003839794	0.002007904	-0.000136969
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6
0.00036765	0.994634046	0.299758257	0.002692028	0.78225683	0.089388814
r_sqr1	r_sqr2		orderP	orderQ	
0.998315873	0.999999986		3	2	

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

The adjusted coefficient of determination for the Quantum Shammas Padé Polynomial is very disappointing compared to the one for the classical Padé polynomial.

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

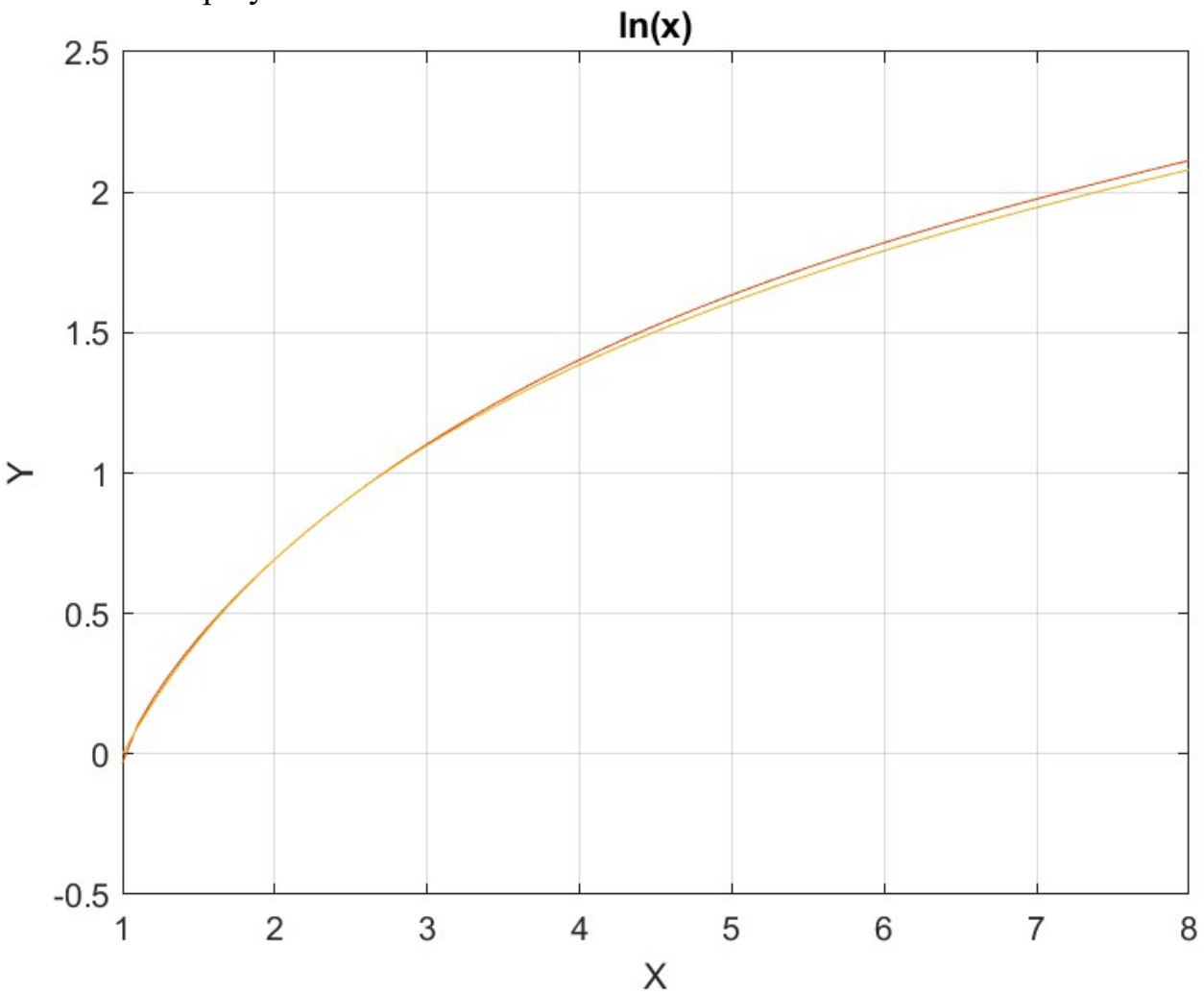

Figure 11. The graph from file Ln_x_pade_halton_rand.jpg.

The above graph shows the low quality of fit for the Quantum Shammas Pade Polynomial.

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, 8) and samples at 0.1 steps, and using the Sobol quasi-random search optimization.

```
clc
clear
close all
```



```
global xData yData yCalc glbRsqr QSPcoeff
global orderP orderQ
zFilename = "Ln_x_pade_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=1:0.1:8\n")
xData0= 1:0.1:8;
xData0 = xData0';
n = length(xData0);
yData = log(xData0);
xData = xData0 - 1;
bestR2 = -1;
for orderP=2:6
  for orderQ=2:6
    fprintf("\norderP = %d, orderQ=%d\n\n", orderP,orderQ)

    [LbP,UbP] = makeLimits(orderP, 0.5, 1.4);
    [LbQ,UbQ] = makeLimits(orderQ, 0.5, 1.4);
    [bestX,bestFx] =
sobolRandomSearchPade(@quantShammasPadePoly,LbP,UbP,LbQ,UbQ,2000
000);

    SSE = quantShammasPadePoly(bestX);
    orderTot = orderP + orderQ;
    % calculate adjusted value of the coefficient of
determination
    glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-orderTot-1);
    if bestR2 < glbRsqr
      bestR2 = glbRsqr;
      bestP = orderP;
      bestQ = orderQ;
      bestQSPcoeff = QSPcoeff;
      zBestX = bestX;
      bestYCalc = yCalc;
    end
  end
end

fprintf("Best orderP = %d\n", bestP);
fprintf("Best orderQ = %d\n", bestQ);
fprintf("Adjusted Rsqr = %f\n", bestR2);
```



```
fprintf("Quantum Shammas Pade Polynomial Powers\n");
zBestX
fprintf("Quantum Shammas Pade Polynomial Coefficients\n");
QSPcoeff = bestQSPcoeff'
fprintf("\nRegular Pade polynomial fit\n");
[c,r,yPoly] = padefit(xData,yData,bestP,bestQ);
% calculate adjusted value of the coefficient of determination
orderTot = bestP + bestQ;
r = 1 - (1 - r)*(n-1)/(n-orderTot-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData0,yData,xData0,bestYCalc,xData0,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = zBestX;
Coeff = 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 = [bestR2 r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
padeOrderP = bestP;
padeOrderQ = bestQ;
orderP = [padeOrderP];
T = array2table(orderP);
writetable(T,xlFile,"Sheet","Sheet1","Range","D10");
orderQ = [padeOrderQ];
T = array2table(orderQ);
writetable(T,xlFile,"Sheet","Sheet1","Range","E10");

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
```

The above file generates the following Excel table summary.

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	
0.642915068	1.644619351	0.956465739	1.733651534	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
-0.075742172	0.799864229	-0.026496794	0.001346314	-0.000281711
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.002100533	0.979757131	0.1660485	0.617667098	0.038547232
r_sqr1	r_sqr2		orderP	orderQ
0.99931731	0.999999557		2	2

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

The adjusted coefficient of determination for the Quantum Shammas Padé Polynomial is less than the one for classical polynomials. The results of using the Sobol sequence are not as poor as using the Halton sequence.

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

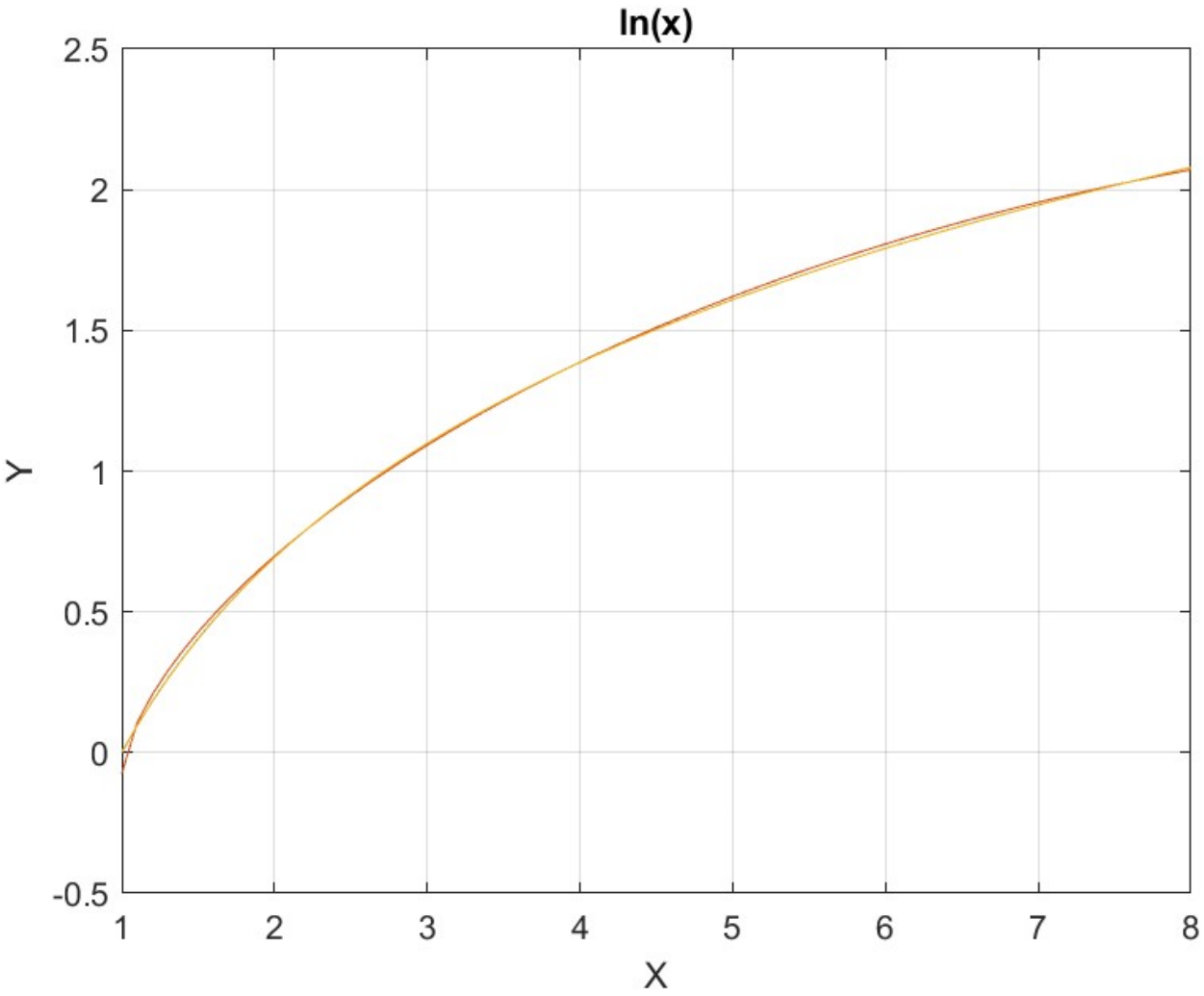

Figure 12. The graph from file ln_x_sobol_rand.jpg

The above graph shows the low quality of fit for the Quantum Shammas Pade Polynomial.

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, 8) using the Quantum Shammas Padé Polynomial was NOT a success.

The next four subsections in Part 2 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.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
global orderP orderQ
zFilename = "Opt_Right_GaussBell_x_pade_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 = "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);
bestR2 = -1;
for orderP=2:6
  for orderQ=2:6
    fprintf("\norderP = %d, orderQ=%d\n\n", orderP,orderQ)
    [LbP,UbP] = makeLimits(orderP, 0.5, 1.4);
    [LbQ,UbQ] = makeLimits(orderQ, 0.5, 1.4);
    Lb = [LbP LbQ];
    Ub = [UbP UbQ];
    [bestX,bestFx] =
psox(@quantShammasPadePoly,Lb,Ub,1000,5000,true);

    SSE = quantShammasPadePoly(bestX);
    orderTot = orderP + orderQ;
    % calculate adjusted value of the coefficient of
determination
    glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-orderTot-1);
    if bestR2 < glbRsqr
      bestR2 = glbRsqr;
      bestP = orderP;
      bestQ = orderQ;
      bestQSPcoeff = QSPcoeff;
      zBestX = bestX;
```



```
      bestYCalc = yCalc;
    end
  end
end
fprintf("Best orderP = %d\n", bestP);
fprintf("Best orderQ = %d\n", bestQ);
fprintf("Adjusted Rsqr = %f\n", bestR2);
fprintf("Quantum Shammas Pade Polynomial Powers\n");
zBestX
fprintf("Quantum Shammas Pade Polynomial Coefficients\n");
QSPcoeff = bestQSPcoeff'
fprintf("\nRegular Pade polynomial fit\n");
[c,r,yPoly] = padefit(xData,yData,bestP,bestQ);
% calculate adjusted value of the coefficient of determination
orderTot = bestP + bestQ;
r = 1 - (1 - r)*(n-1)/(n-orderTot-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,bestYCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = zBestX;
Coeff = 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 = [bestR2 r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
padeOrderP = bestP;
padeOrderQ = bestQ;
orderP = [padeOrderP];
T = array2table(orderP);
writetable(T,xlFile,"Sheet","Sheet1","Range","D10");
orderQ = [padeOrderQ];
T = array2table(orderQ);
writetable(T,xlFile,"Sheet","Sheet1","Range","E10");
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
```

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	QSPpwr5
0.701360693	2.398663049	3.095285278	3.730938601	4.777260308
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.399144909	-0.006744056	-0.359864215	0.214485846	0.015699747
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.398939415	-0.07212055	-0.138484347	0.040979147	0.012501147
r_sqr1	r_sqr2		orderP	orderQ
0.999999002	0.999999999		6	2

Table 13a. Summary of the results appearing in file Right_GaussBell_x_pade_pso.xlsx.

QSPpwr6	QSPpwr7	QSPpwr8	
5.537785146	0.725239287	2.05667251	

QSPcoeff6	QSPcoeff7	QSPcoeff8	QSPcoeff9
-0.025867287	0.004976886	-0.000300035	0.000227279
Coeff6	Coeff7	Coeff8	Coeff9
-0.006409313	0.000700281	-0.181300897	0.157063193

Table 13b. Summary of the results appearing in file Right_GaussBell_x_pade_pso.xlsx.

The adjusted coefficient of determination for the Quantum Shammas Padé Polynomial is slightly less than the one for classical Padé polynomial.

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

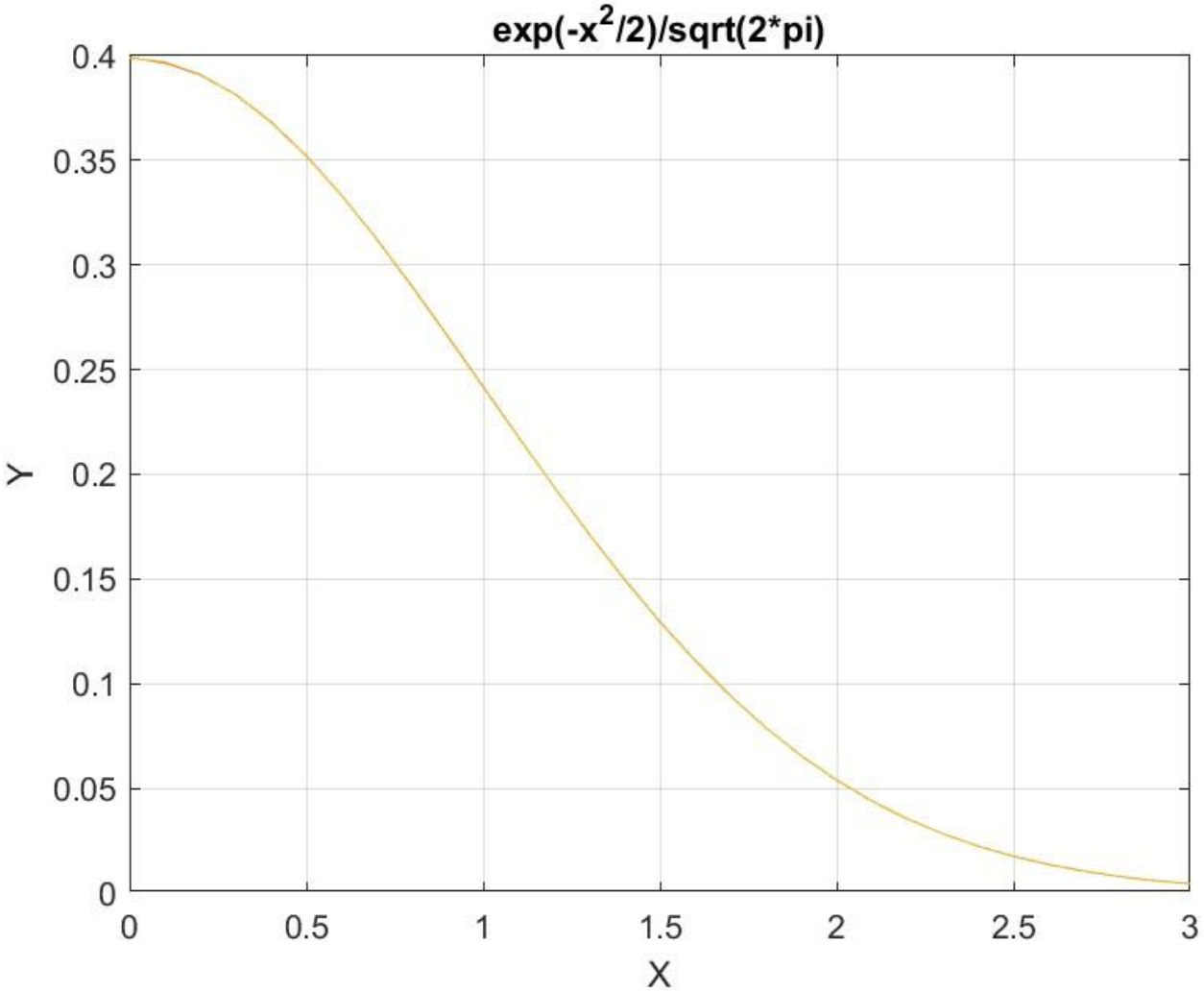

Figure 13. The graph from file Right_GaussBell_x_pade_pso.jpg.

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

```
clc
```



```
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
global orderP orderQ
zFilename = "Opt_Right_GaussBell_x_pade_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);
bestR2 = -1;
for orderP=2:6
  for orderQ=2:6
    fprintf("\norderP = %d, orderQ=%d\n\n", orderP,orderQ)
  [LbP,UbP] = makeLimits(orderP, 0.5, 1.4);
  [LbQ,UbQ] = makeLimits(orderQ, 0.5, 1.4);
  [bestX,bestFx]
=randomSearchPade(@quantShammasPadePoly,LbP,UbP,LbQ,UbQ,2000000)
;

  SSE = quantShammasPadePoly(bestX);
  orderTot = orderP + orderQ;
  % calculate adjusted value of the coefficient of determination
  glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-orderTot-1);
    if bestR2 < glbRsqr
      bestR2 = glbRsqr;
      bestP = orderP;
      bestQ = orderQ;
      bestQSPcoeff = QSPcoeff;
      zBestX = bestX;
      bestYCalc = yCalc;
    end
  end
end
fprintf("Best orderP = %d\n", bestP);
fprintf("Best orderQ = %d\n", bestQ);
fprintf("Adjusted Rsqr = %f\n", bestR2);
fprintf("Quantum Shammas Pade Polynomial Powers\n");
```



```
zBestX
fprintf("Quantum Shammas Pade Polynomial Coefficients\n");
QSPcoeff = bestQSPcoeff'
fprintf("\nRegular Pade polynomial fit\n");
[c,r,yPoly] = padefit(xData,yData,bestP,bestQ);
% calculate adjusted value of the coefficient of determination
orderTot = bestP + bestQ;
r = 1 - (1 - r)*(n-1)/(n-orderTot-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,bestYCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = zBestX;
Coeff = 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 = [bestR2 r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
padeOrderP = bestP;
padeOrderQ = bestQ;
orderP = [padeOrderP];
T = array2table(orderP);
writetable(T,xlFile,"Sheet","Sheet1","Range","D10");
orderQ = [padeOrderQ];
T = array2table(orderQ);
writetable(T,xlFile,"Sheet","Sheet1","Range","E10");
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
```

The above code generates the following summary Excel table:

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5
0.696383988	2.391682266	3.340539594	4.391228872	5.285658856
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.399149498	-0.007108881	-0.32698618	0.225058833	-0.05444248
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.398918701	-0.111274224	-0.133835654	0.079069264	-0.014953101
r_sqr1	r_sqr2		orderP	orderQ
0.999995242	0.999999985		5	2

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

QSPpwr6	QSPpwr7	
1.381293535	2.073445839	
QSPcoeff6	QSPcoeff7	QSPcoeff8
0.006284392	-0.001948838	0.002591535
Coeff6	Coeff7	Coeff8
0.000921217	-0.2820541	0.1864216

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

The adjusted coefficient of determination for the Quantum Shammas Padé Polynomial is less (by a proverbial hair) than the one for classical Padé polynomial.

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

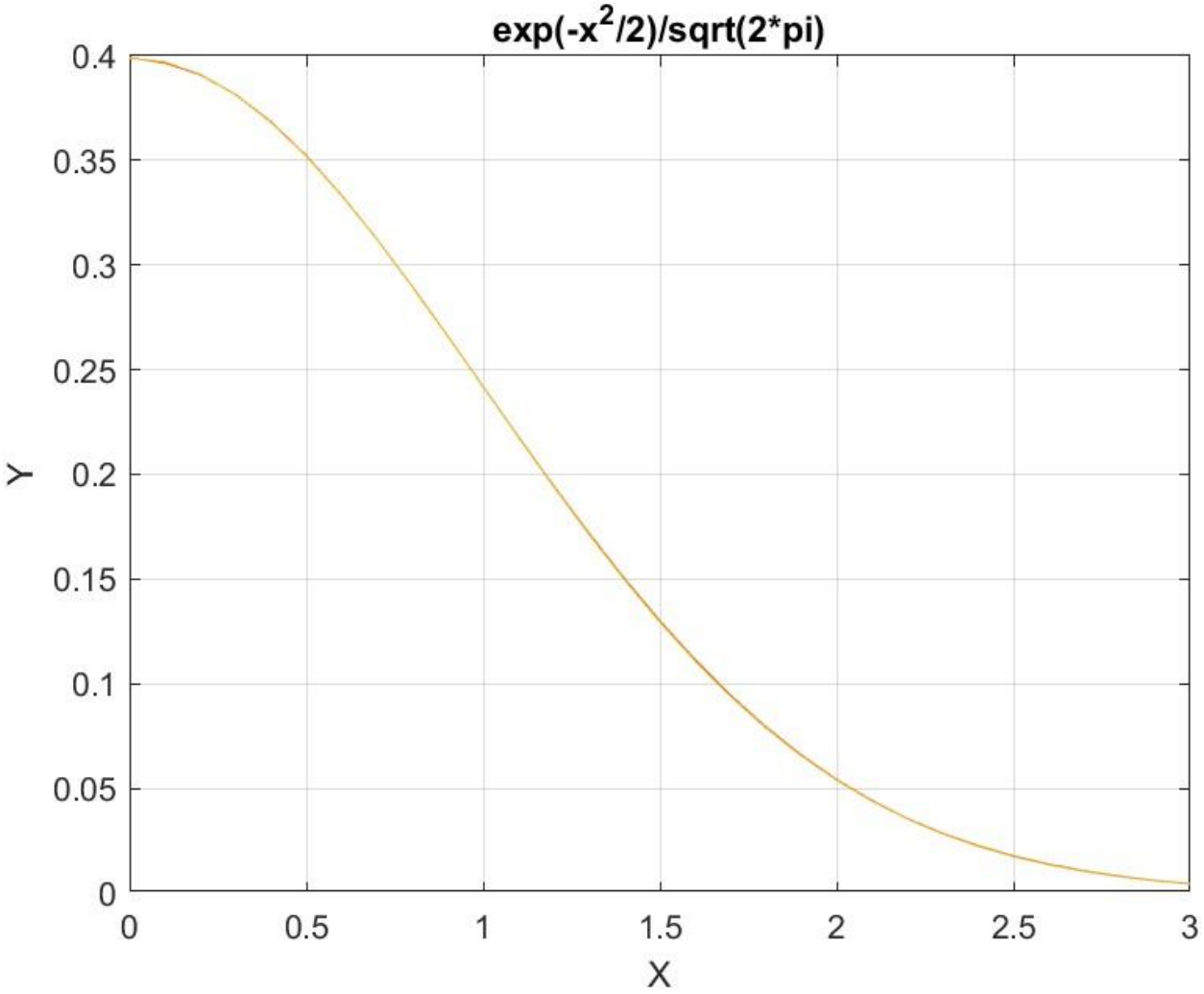

Figure 14. The graph from file Right_GaussBell_x_random.jpg.

The above graph shows that the 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 testPadeGauss1Halton.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.

```
clc
```



```
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
global orderP orderQ
zFilename = "Opt_Ln_x_pade_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=1:0.1:8\n")
xData0= 1:0.1:8;
xData0 = xData0';
n = length(xData0);
yData = log(xData0);
xData = xData0 - 1;
bestR2 = -1;
for orderP=2:6
  for orderQ=2:6
    fprintf("\norderP = %d, orderQ=%d\n\n", orderP,orderQ)

    [LbP,UbP] = makeLimits(orderP, 0.5, 1.4);
    [LbQ,UbQ] = makeLimits(orderQ, 0.5, 1.4);
    [bestX,bestFx] =
haltonRandomSearchPade(@quantShammasPadePoly,LbP,UbP,LbQ,UbQ,200
0000);

    SSE = quantShammasPadePoly(bestX);
    orderTot = orderP + orderQ;
    % calculate adjusted value of the coefficient of
determination
    glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-orderTot-1);
    if bestR2 < glbRsqr
      bestR2 = glbRsqr;
      bestP = orderP;
      bestQ = orderQ;
      bestQSPcoeff = QSPcoeff;
      zBestX = bestX;
      bestYCalc = yCalc;
    end
  end
end
fprintf("Best orderP = %d\n", bestP);
```



```
fprintf("Best orderQ = %d\n", bestQ);
fprintf("Adjusted Rsqr = %f\n", bestR2);
fprintf("Quantum Shammas Pade Polynomial Powers\n");
zBestX
fprintf("Quantum Shammas Pade Polynomial Coefficients\n");
QSPcoeff = bestQSPcoeff'
fprintf("\nRegular Pade polynomial fit\n");
[c,r,yPoly] = padefit(xData,yData,bestP,bestQ);
% calculate adjusted value of the coefficient of determination
orderTot = bestP + bestQ;
r = 1 - (1 - r)*(n-1)/(n-orderTot-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData0,yData,xData0,bestYCalc,xData0,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = zBestX;
Coeff = 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 = [bestR2 r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
padeOrderP = bestP;
padeOrderQ = bestQ;
orderP = [padeOrderP];
T = array2table(orderP);
writetable(T,xlFile,"Sheet","Sheet1","Range","D10");
orderQ = [padeOrderQ];
T = array2table(orderQ);
writetable(T,xlFile,"Sheet","Sheet1","Range","E10");
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
```

The above code generates the following summary Excel table:

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5
1.120342538	2.285698867	3.372993345	3.511421392	4.628391968
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.398103079	0.009172122	-0.370143238	0.92755725	-0.728550069
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.398918701	-0.111274224	-0.133835654	0.079069264	-0.014953101
r_sqr1	r_sqr2		orderP	orderQ
0.999983219	0.999999985		5	2

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

QSPpwr6	QSPpwr7	
0.935914795	1.614929736	
QSPcoeff6	QSPcoeff7	QSPcoeff8
0.005611556	0.000702911	1.40828E-05
Coeff6	Coeff7	Coeff8
0.000921217	-0.2820541	0.1864216

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

The adjusted coefficient of determination for the Quantum Shammas Padé Polynomial is less than the one for classical Padé polynomial.

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

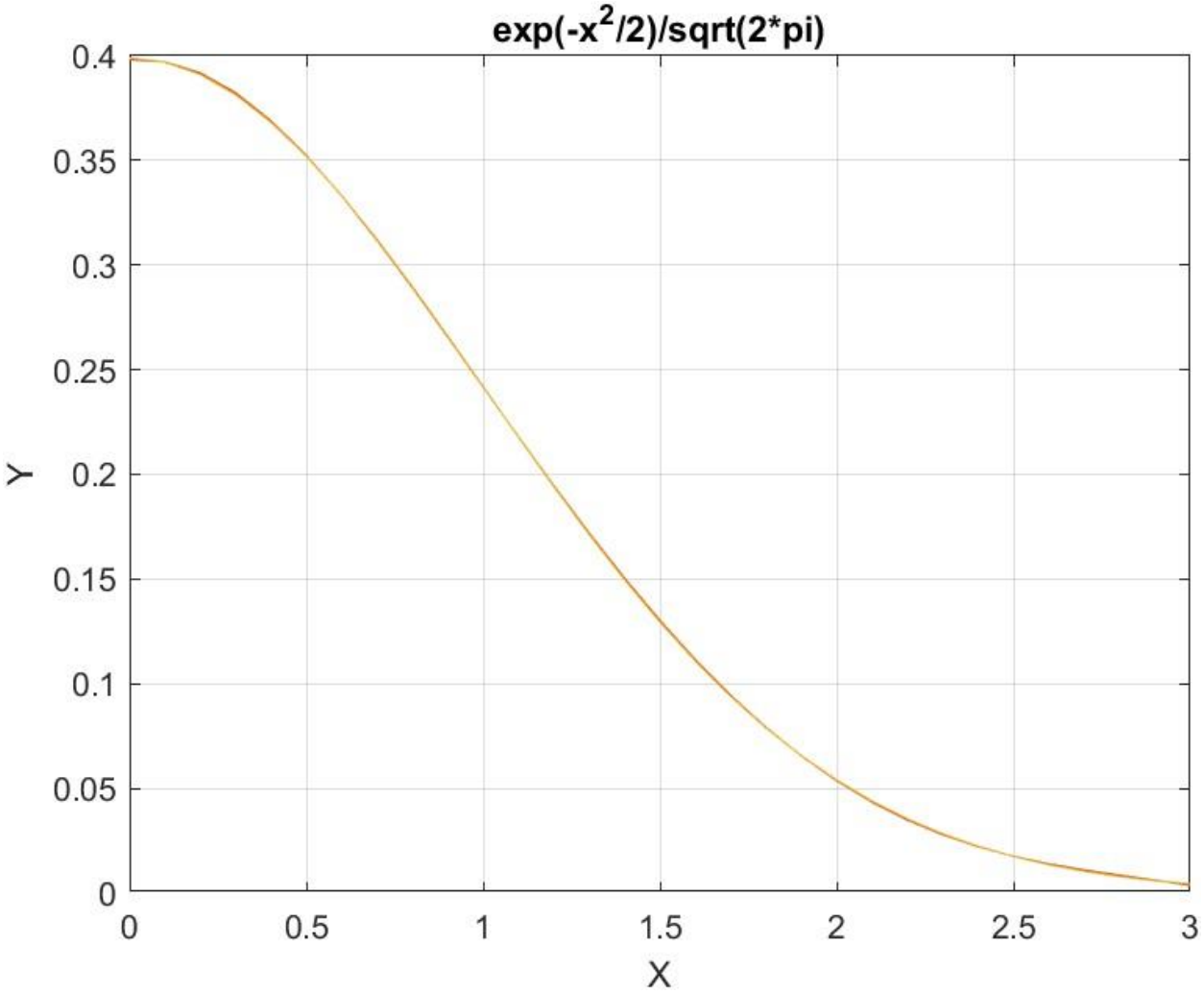

Figure 15. The graph from file Right_GaussBell_x_halton_random.jpg.

The above graph shows that the 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 testGauss1Random.m) tests fitting the normal N(0, 1) function for x in the range (0, 3) and samples at 0.1 steps, and using the Sobol quasi-random search optimization.

```
clc
```



```
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
global orderP orderQ
zFilename = "Opt_Right_GaussBell_x_pade_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);
bestR2 = -1;
for orderP=2:6
  for orderQ=2:6
    fprintf("\norderP = %d, orderQ=%d\n\n", orderP,orderQ)

    [LbP,UbP] = makeLimits(orderP, 0.5, 1.4);
    [LbQ,UbQ] = makeLimits(orderQ, 0.5, 1.4);
    [bestX,bestFx] =
sobolRandomSearchPade(@quantShammasPadePoly,LbP,UbP,LbQ,UbQ,2000
000);

    SSE = quantShammasPadePoly(bestX);
    orderTot = orderP + orderQ;
    % calculate adjusted value of the coefficient of
determination
    glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-orderTot-1);
    if bestR2 < glbRsqr
      bestR2 = glbRsqr;
      bestP = orderP;
      bestQ = orderQ;
      bestQSPcoeff = QSPcoeff;
      zBestX = bestX;
      bestYCalc = yCalc;
    end
  end
end
fprintf("Best orderP = %d\n", bestP);
fprintf("Best orderQ = %d\n", bestQ);
```



```
fprintf("Adjusted Rsqr = %f\n", bestR2);
fprintf("Quantum Shammas Pade Polynomial Powers\n");
zBestX
fprintf("Quantum Shammas Pade Polynomial Coefficients\n");
QSPcoeff = bestQSPcoeff'
fprintf("\nRegular Pade polynomial fit\n");
[c,r,yPoly] = padefit(xData,yData,bestP,bestQ);
% calculate adjusted value of the coefficient of determination
orderTot = bestP + bestQ;
r = 1 - (1 - r)*(n-1)/(n-orderTot-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,bestYCalc,xData,yPoly);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

QSPpwr = zBestX;
Coeff = 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 = [bestR2 r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A10");
padeOrderP = bestP;
padeOrderQ = bestQ;
orderP = [padeOrderP];
T = array2table(orderP);
writetable(T,xlFile,"Sheet","Sheet1","Range","D10");
orderQ = [padeOrderQ];
T = array2table(orderQ);
writetable(T,xlFile,"Sheet","Sheet1","Range","E10");
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
```

The above code generates the following summary Excel table:

QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5
0.537918691	2.391085243	3.396326092	4.329440284	4.946769424
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
0.39910017	-0.004332468	-0.332977309	0.250908836	-0.088264405
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
0.398918701	-0.111274224	-0.133835654	0.079069264	-0.014953101
r_sqr1	r_sqr2		orderP	orderQ
0.999994816	0.999999985		5	2

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

QSPpwr6	QSPpwr7	
0.935788797	1.981997913	
QSPcoeff6	QSPcoeff7	QSPcoeff8
0.017516667	0.002699022	-0.0018495
Coeff6	Coeff7	Coeff8
0.000921217	-0.2820541	0.1864216

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

The adjusted coefficient of determination for the Quantum Shammas Padé Polynomial is less than the one for classical Padé polynomial.

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

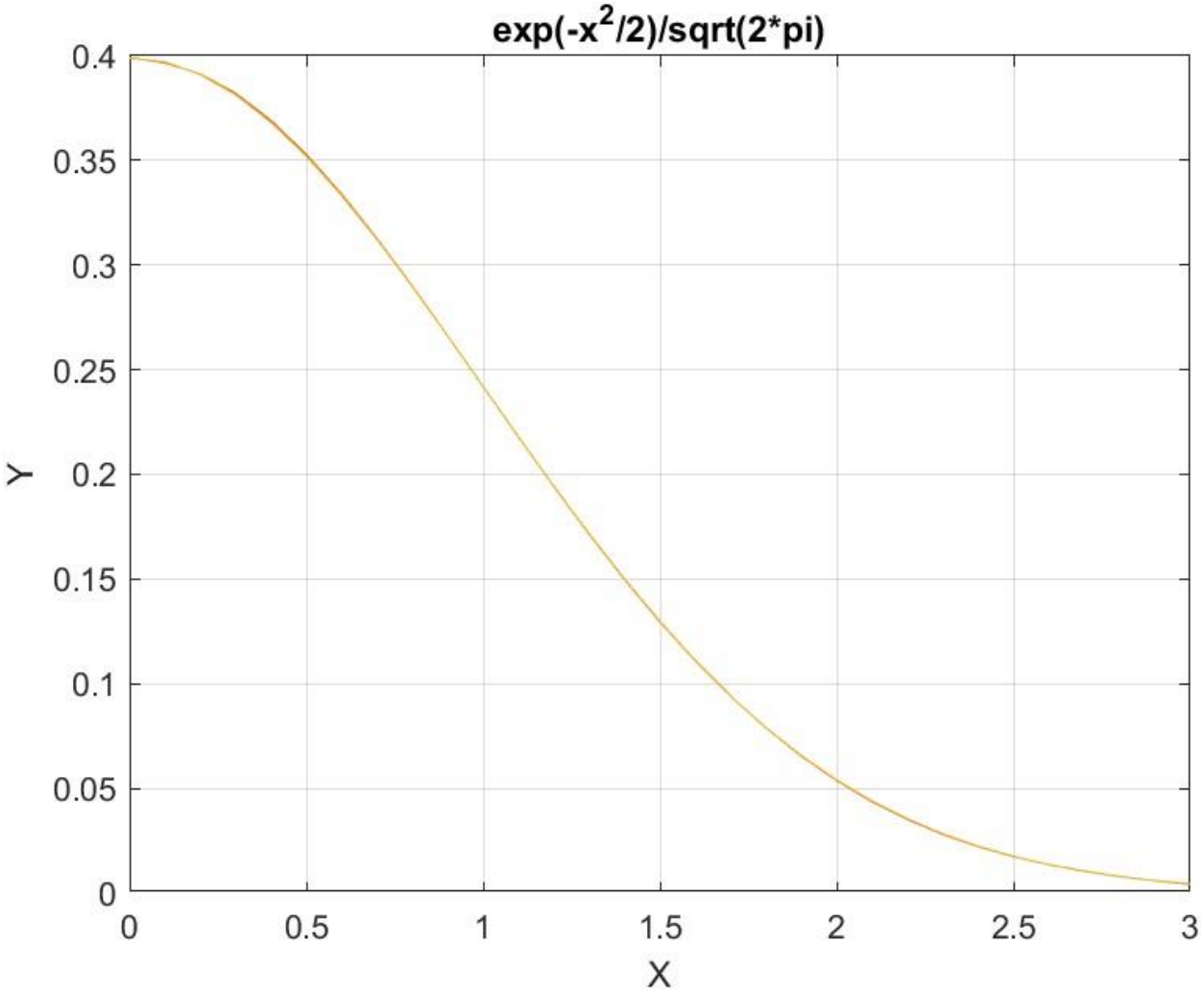

Figure 14. The graph from file Right_GaussBell_x_sobol_random.jpg.

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 Padé Polynomial was not the expected success These polynomials yield adjusted coefficients of determination that are slightly less than the corresponding classical Padé polynomials.

Conclusion for Part 2

The Quantum Shammas Padé Polynomials had adjusted coefficients of determination that were always below those of the classical Padé polynomials. This tiny lag is still disappointing given the fact that more computational efforts went for the Quantum Shammas Padé Polynomials. I have even run different sets of calculations with different schemes for the random powers of the Quantum Shammas Padé Polynomials. The results are pretty much the same, with classical Padé polynomials being consistently in the lead! I also included the Shammas Padé Polynomials in the calculations (which simply used assigned powers and require no optimization). These polynomials did better than Quantum Shammas Padé Polynomials! The powers for the Shammas Padé Polynomials were (LbP(i)+(UbP(i)-LbP(i))/2 for i=1 to the order of numerator polynomial P(x), and LbQ(i)+(UbP(i)-LbQ(i))/2 for i=1 to the order of denominator polynomial Q(x)). The orders of the numerator and denominator polynomials for the classical Padé polynomials and the Shammas Padé Polynomials were the optimum values obtained for the Quantum Shammas Padé Polynomials.

Given this observation, here is the MATALB code for the bestpadfit() function that obtains the best classical Padé fit in the orders that range between 2 and an upper user-specified limits for the numerator and denominator polynomials.

```
function [c,r2,yCalc,orderP,orderQ] =
bestpadefit(xData,yData,maxP,maxQ)
% BESTPADEFIT performs best fit for a Pade pollynomial.
  n = length(xData);
  bestR2 = -1;
  for orderP=2:maxP
    for orderQ=2:maxQ
      X = [1+zeros(n,1)];
      for j=1:orderP
        X = [X xData.^j];
      end
      for j=1:orderQ
        X = [X -yData.*xData.^j];
      end
      [c] = regress(yData,X);
      SSE = 0;
      ymean = mean(yData);
      SStot = sum((yData - ymean).^2);
      yCalc = zeros(n,1);
      for i=1:n
```



```
        Px = c(1);
        for j=1:orderP
          Px = Px + c(j+1)*xData(i)^j;
        end
        Qx = 1;
        k=1+orderP;
        for j=1:orderQ
          Qx = Qx + c(k+j)*xData(i)^j;
        end
        yCalc(i) = Px / Qx;
        SSE = SSE + (yCalc(i) - yData(i))^2;
      end
      r2 = 1 - SSE / SStot;

      if r2 > bestR2
        bestR2 = r2;
        bestP = orderP;
        bestQ = orderQ;
        bestYCalc = yCalc;
        bestCoeff = c;
      end
    end
  end

  c = bestCoeff;
  r2 = bestR2;
  orderP = bestP;
  orderQ = bestQ;
  yCalc = bestYCalc;
end
```

The function has the following input parameters:

- The xData parameter is the array of x values.
- The yData parameter is the array of y values.
- The maxP parameter is the maximum order for the numerator polynomial.
- The maxQ parameter is the maximum order for the denominator polynomial.

The function has the following output paramaters:

- The parameter c is the array of polynomial coefficients for the best Padé polynomial.
- The parameter r2 is the coefficient of determination of the best Padé polynomial.

- The parameter yCalc is the array of the calculated y values for the xData values.
- The parameter orderP is the best order of the numerator polynomial.
- The parameter orderQ is the best order of the denominator polynomial.

Next is Part 3

Part 3 of this study looks at the Quantum Shammas Fourier Series for the same test cases presented in Part 1.

Document History

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

