Quantum Shammas Fourier Series Part 3 of the Study

By
Namir Shammas

Contents

Introduction

Part 1 of this study introduces you to Quantum Shammas Polynomials and the notion of using random powers that fall within specific ranges. This part of the study looks at Quantum Shammas Fourier Series where the coefficients inside the sine and cosine terms are not fixed integers but are random values that fall within specific ranges. A Quantum Shammas Fourier Series looks like:

$$y(x) = a_0 + a_1 * \sin(s_1 * \pi * x) + b_1 * \cos(c_1 * \pi * x) + \ldots + a_n * \sin(s_n * \pi * x) + b_n * \cos(c_n * \pi * x) \qquad (1)$$

Where the values for s_i and c_i fall the ranges $0.5 <= s_1 <= 1.4$, $1.5 <= s_2 <= 2.4$, …, and $(n-1)+0.5 <= s_n < n+0.4$ and , $0.5 <= c_1 <= 1.4$, $1.5 <= c_2 <= 2.4$, …, and $(n-1)+0.5 <= c_n < n+0.4$, respectively.

The classical Fourier series are of course:

$$\begin{aligned} y(x) = a_0 &+ a_1 * \sin(\pi * x) + b_1 * \cos(\pi * x) + \ldots + \\ &a_2 * \sin(2 * \pi * x) + b_2 * \cos(2 * \pi * x) + \ldots + \\ &a_n * \sin(n * \pi * x) + b_n * \cos(n * \pi * x) \end{aligned} \qquad (2)$$

The Quantum Shammas Fourier Series Function

The Quantum Shammas Fourier Series function (found in file quantShammasFourierPoly.m) in MATLAB is:

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

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

The PSO Function

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

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

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

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

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

  % pso loop
  for iter = 1:MaxIters

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

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

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

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

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

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

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

The function has the following input parameters:

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

- The parameter MaxPop is the maximum population of swarm.
- The parameter MaxIters is the maximum number of iterations.
- The parameter bShow is the Boolean flag to request viewing intermediate results.

The output parameters are:

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

The Random Search Function

The next function performs a random search optimization:

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

  bestFx = 1e99;
  n = length(Lb);
  bestX = 1e+99+zeros(n,1);
  for irun=1:2
    for iter = 1:MaxIters
      X = Lb + (Ub - Lb).*rand(1,n);
      f = fx(X);
      if f < bestFx
        bestFx = f;
        bestX = X;
        k = iter + (irun-1) *MaxIters;
        fprintf("%7i: Fx = %e, X=[", k, bestFx);
        fprintf("%f, ", X)
        fprintf("]\n");
      end
    end

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

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

The function has the following input parameters:

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

The output parameters are:

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

The above function is easy to code and works well with Quasi Shammas Polynomials since the range of each power is relatively small (<1).

The Halton Quasi Random Search Function

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

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

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

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

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

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

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

The Sobol Quasi Random Search Function

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

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

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

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

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

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

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

The Fourier Fit Function

The Fourier fit function (found in file fourerfit.m) is the MATLAB script that performs regular Fourier series least-squares curve fitting:

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

The above has the following parameters:

1. The xData parameter is the array of x values.
2. The yData parameter is the array of y values.
3. The order parameter specifies the number of sine and cosine pairs used in the Fourier series fitting.

The output parameters are:

1. The c parameter is the array of Fourier series coefficients.
2. The r2 parameter is the coefficient of determination.
3. The yCalc parameter is the array of projected y values.

The Test Functions

This part of the study tests the following three functions using the various methods:

1. The function $y_1(x)$ = 2+(sin(x)+2*sin(x)^2)*log(1+x) for $0 \leq x \leq \pi$.
2. The function $y_2(x)$ = (sin(2*x)^2)*sinh(x/5) for $0 \leq x \leq \pi$.
3. The function $y_3(x)$ = 1+J_0(x)*sin(x) for $0 \leq x \leq \pi$.
4. The function $y_4(x)$ = 1 + (sin(x) - 2*sin(2*x-pi) + 3*sin(3*x-pi)*(cos(x) - 2*cos(2*x-pi) + 3*cos(3*x-pi) for $0 \leq x \leq \pi$.

I will refer to the above functions as function 1, function 2, function 3, and function 4, respectively.

Testing Quantum Shammas Fourier Series

The next subsections show examples of using the Quasi Shammas Fourier Series to fit the three functions that I mentioned in the last section. The results of the Quasi Shammas Fourier Series are compared with those of classical Fourier series. The adjusted coefficient of determinations are good indicators of how the two types of polynomial stack up against each other.

Testing Function 1 Fit with PSO

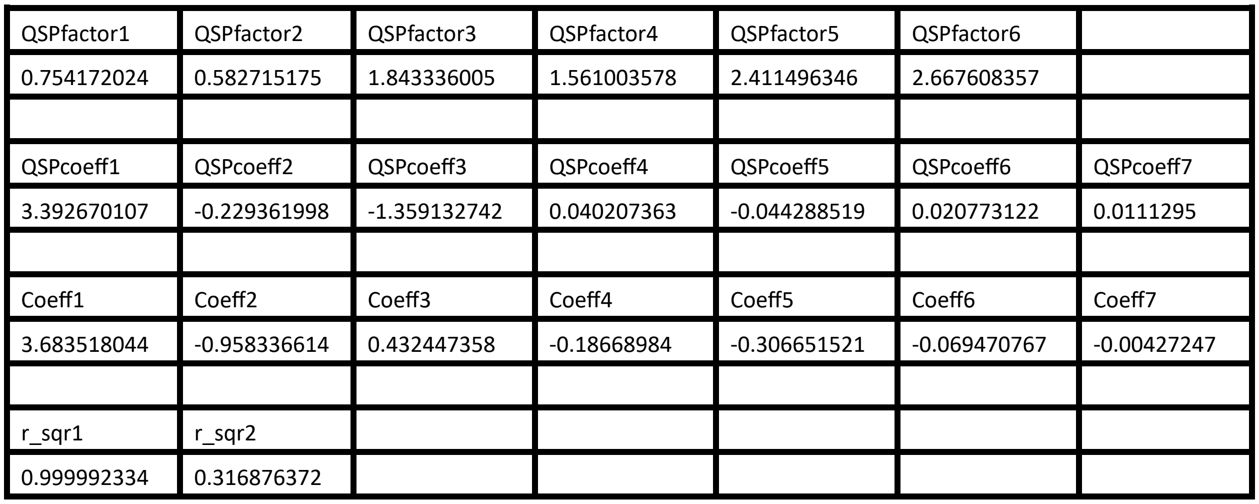
The next MATLAB script (found in file testFourierFx1pso.m) tests fitting *function 1* for x in the range (0, π) and samples at 0.1 steps. The curve fit uses the third order Quantum Shammas Fourier Series and a third order classical Fourier series.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "sin_by_sqr_sin_fourier_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 = "2+(sin(x)+2*sin(x).^2).*log(1+x)";
fprintf("%s\n", sEqn);
xData = 0:.1:pi;
xData = xData';
n=length(xData);
```

```
yData = 2+(sin(xData)+2*sin(xData).^2).*log(1+xData);
fprintf("x=0:.1:pi\n")
order = 3;
[LbS,UbS] = makeLimits(order, 0.5, 1.4);
Lb = [];
Ub = [];
for i=1:order
  Lb = [Lb LbS(i) LbS(i)];
  Ub = [Ub UbS(i) UbS(i)];
end
[bestX,bestFx] =
psox(@quantShammasFourierPoly,Lb,Ub,1000,5000,true);

SSE = quantShammasFourierPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Fourier Series factors\n");
bestX
fprintf("Quantum Shammas Fourier Series Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular Fourier fit\n");
sincos_factors = [];
for i=1:order
  sincos_factors =[sincos_factors i i];
end
sincos_factors
[c,r,yFourier] = fourierfit(xData,yData,order);
c = c'
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,yCalc,xData,yFourier);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

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

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + maxPwr;
  end
end
```

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



QSPfactor1	QSPfactor2	QSPfactor3	QSPfactor4	QSPfactor5	QSPfactor6	
0.754172024	0.582715175	1.843336005	1.561003578	2.411496346	2.667608357	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
3.392670107	-0.229361998	-1.359132742	0.040207363	-0.044288519	0.020773122	0.0111295
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
3.683518044	-0.958336614	0.432447358	-0.18668984	-0.306651521	-0.069470767	-0.00427247
r_sqr1	r_sqr2					
0.999992334	0.316876372					

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

The second row shows the powers for the fitted Quantum Shammas Fourier Series. The fifth row shows the intercept (below QSPcoeff1) and to its right the coefficients for the various Quantum Shammas Fourier Series. 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 Fourier

Series. The cell under r_sqr2 shows the adjusted coefficient of determination for the fitted classical polynomial. The adjusted coefficient of determination for the fitted Quantum Shammas Fourier Series is well higher than the one for the regular Fourier series. This condition indicates that the Quantum Shammas Fourier Series performs a better fit for the above example.

Here is the graph for the test function and the two fitted polynomials:

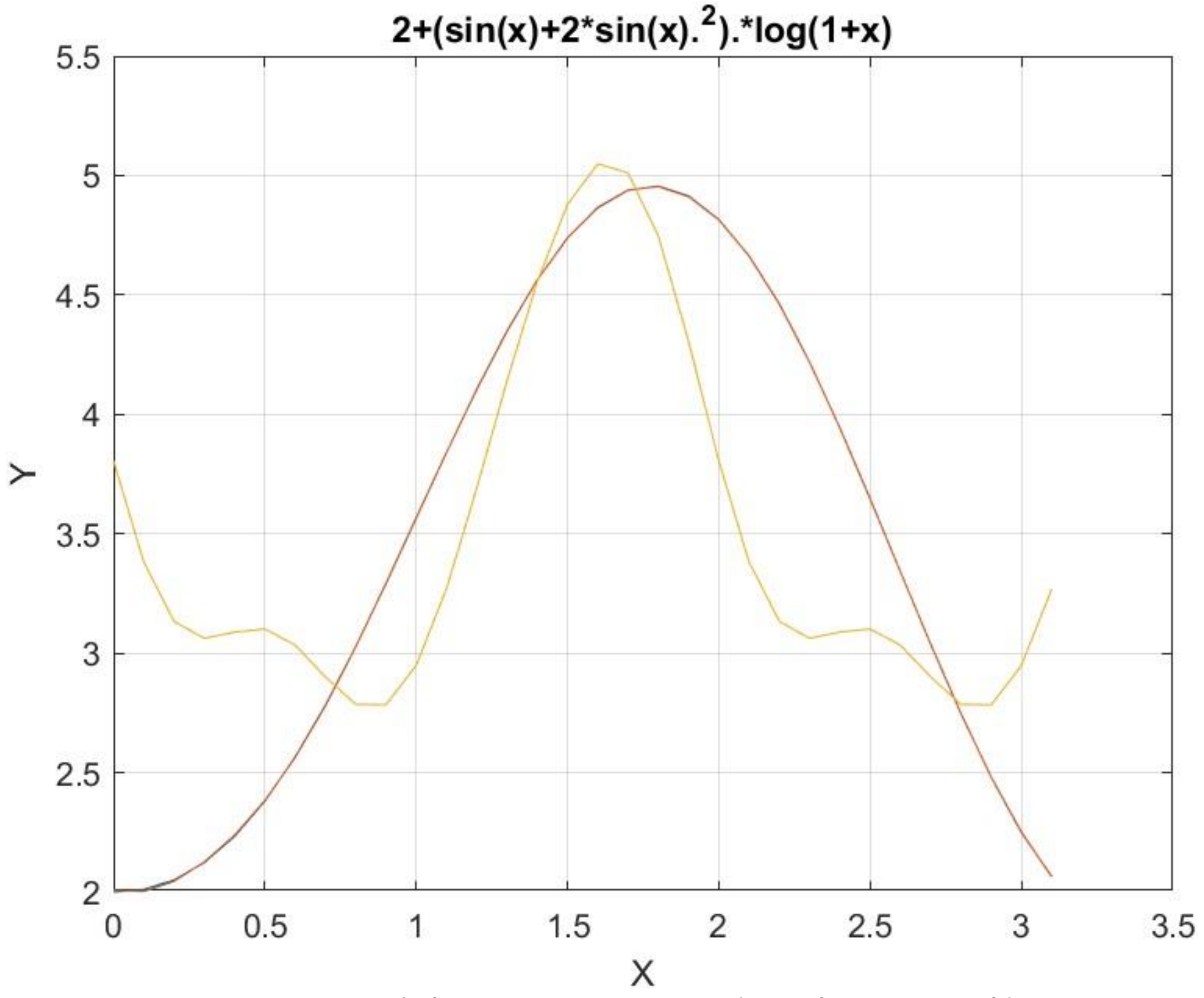


Figure 1. Graph for Fourier Series and test function in file sin_by_sqr_sin_fourier_pso.jpg.

The above figure shows that the Quantum Shammas Fourier Series fit *function 1* well and much better than the regular Fourier series.

Testing Function 1 Fit with Random Search

The next MATLAB script (found in file found in file testFourierFx1Random.m) tests fitting *function 1* for x in the range (0, π) and samples at 0.1 steps. The curve fit uses the third order Quantum Shammas Fourier Series and a third order classical Fourier series.

```
clc
clear
```

```
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "sin_by_sqr_sin_rand_fourier";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "2+(sin(x)+2*sin(x).^2).*log(1+x)";
fprintf("%s\n", sEqn);
xData = 0:.1:pi;
xData = xData';
n = length(xData);
yData = 2+(sin(xData)+2*sin(xData).^2).*log(1+xData);
fprintf("x=0:.1:pi\n")
order = 3;
[LbS,UbS] = makeLimits(order, 0.5, 1.4);
Lb = [];
Ub = [];
for i=1:order
  Lb = [Lb LbS(i) LbS(i)];
  Ub = [Ub UbS(i) UbS(i)];
end
[bestX,bestFx] =
randomSearch(@quantShammasFourierPoly,Lb,Ub,1000000);

SSE = quantShammasFourierPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Fourier Series factors\n");
bestX
fprintf("Quantum Shammas Fourier Series Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular Fourier fit\n");
sincos_factors = [];
for i=1:order
  sincos_factors =[sincos_factors i i];
end
sincos_factors
[c,r,yFourier] = fourierfit(xData,yData,order);
c = c'
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", r);
```

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

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + maxPwr;
  end
end
```

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

QSPfactor1	QSPfactor2	QSPfactor3	QSPfactor4	QSPfactor5	QSPfactor6	
0.777352642	0.580052809	1.748116186	1.483288107	2.815825387	2.57071601	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
3.395706469	-0.236810906	-1.340694091	0.026744821	-0.034520213	0.009288427	-0.011714747

Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
3.683518044	-0.958336614	0.432447358	-0.18668984	-0.306651521	-0.069470767	-0.00427247
r_sqr1	r_sqr2					
0.999987333	0.316876372					

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

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

Here is the graph for the test function and the two fitted polynomials:

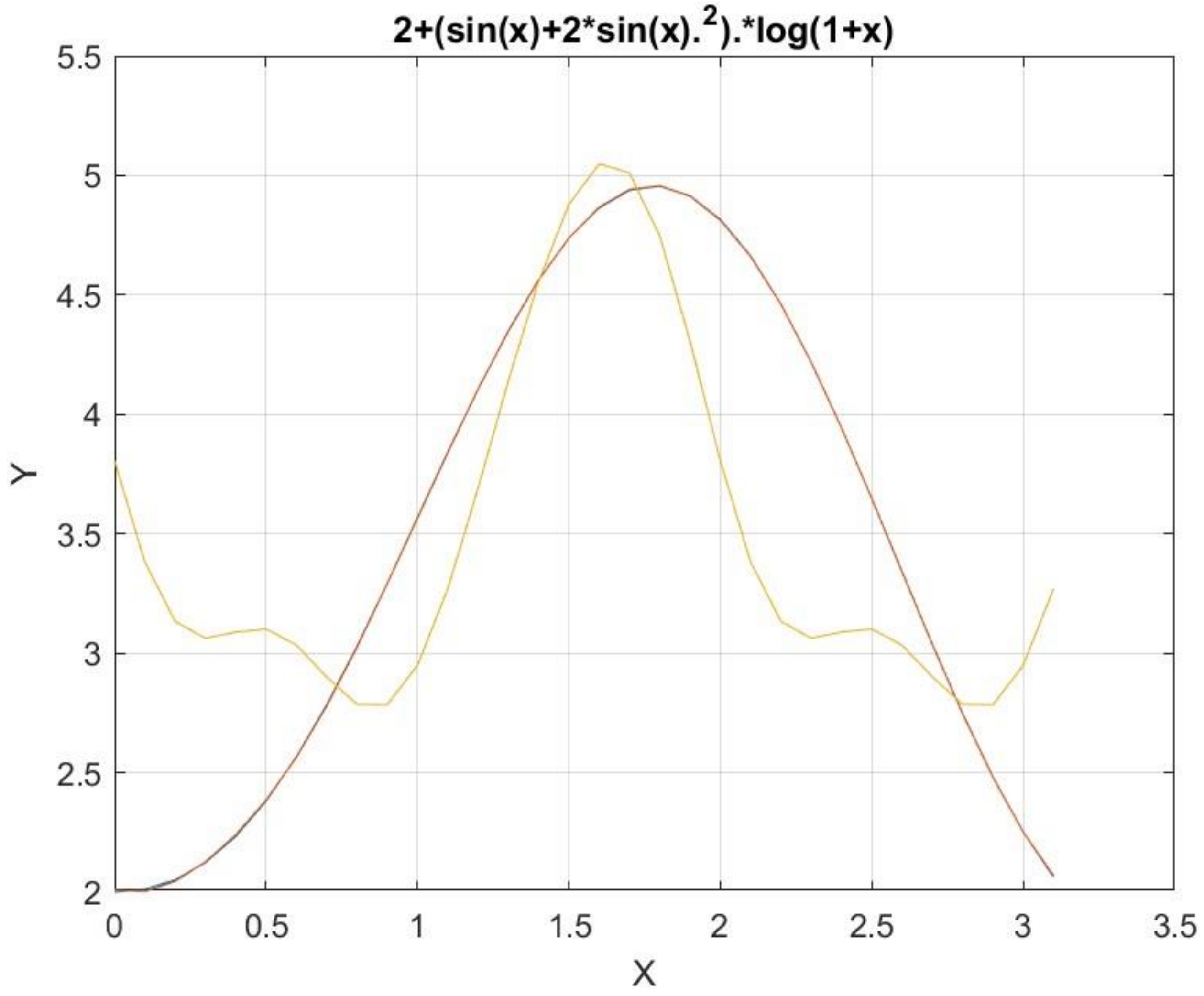


Figure 2. Graph for Fourier Series and test function in file sin_by_sqr_sin_rand_fourier.jpg.

The above figure shows that the Quantum Shammas Fourier Series fit *function 1* well and much better than the regular Fourier series.

Testing Function 1 Fit with Halton Random Search

The next MATLAB script (found in file testFourierFx1Halton.m) tests fitting *function 1* for x in the range (0, π) and samples at 0.1 steps. The curve fit uses the third order Quantum Shammas Fourier Series and a third order classical Fourier series.

```
clc
clear
close all
```

```
global xData yData yCalc glbRsqr QSPcoeff
zFilename = "sin_by_sqr_sin_halton_rand_fourier";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "2+(sin(x)+2*sin(x).^2).*log(1+x)";
fprintf("%s\n", sEqn);
xData = 0:.1:pi;
xData = xData';
n = length(xData);
yData = 2+(sin(xData)+2*sin(xData).^2).*log(1+xData);
fprintf("x=0:.1:pi\n")
order = 3;
[LbS,UbS] = makeLimits(order, 0.5, 1.4);
Lb = [];
Ub = [];
for i=1:order
  Lb = [Lb LbS(i) LbS(i)];
  Ub = [Ub UbS(i) UbS(i)];
end
[bestX,bestFx] =
haltonRandomSearch(@quantShammasFourierPoly,Lb,Ub,1000000);

SSE = quantShammasFourierPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Fourier Series factors\n");
bestX
fprintf("Quantum Shammas Fourier Series Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular Fourier fit\n");
sincos_factors = [];
for i=1:order
  sincos_factors =[sincos_factors i i];
end
sincos_factors
[c,r,yFourier] = fourierfit(xData,yData,order);
c = c'
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
```

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

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + maxPwr;
  end
end
```

The above code generates the following Excel table of results.

QSPfactor1	QSPfactor2	QSPfactor3	QSPfactor4	QSPfactor5	QSPfactor6	
0.777352642	0.580052809	1.748116186	1.483288107	2.815825387	2.57071601	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
3.395706469	-0.236810906	-1.340694091	0.026744821	-0.034520213	0.009288427	-0.011714747
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
3.683518044	-0.958336614	0.432447358	-0.18668984	-0.306651521	-0.069470767	-0.00427247

r_sqr1	r_sqr2					
0.999987333	0.316876372					

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

Again, the adjusted coefficient of determination for the fitted Quantum Shammas Fourier Series is well higher than the one for the regular Fourier series. This condition indicates that the Quantum Shammas Fourier Series performs a better fit for the above example.

Here is the graph for the test function and the two fitted polynomials:

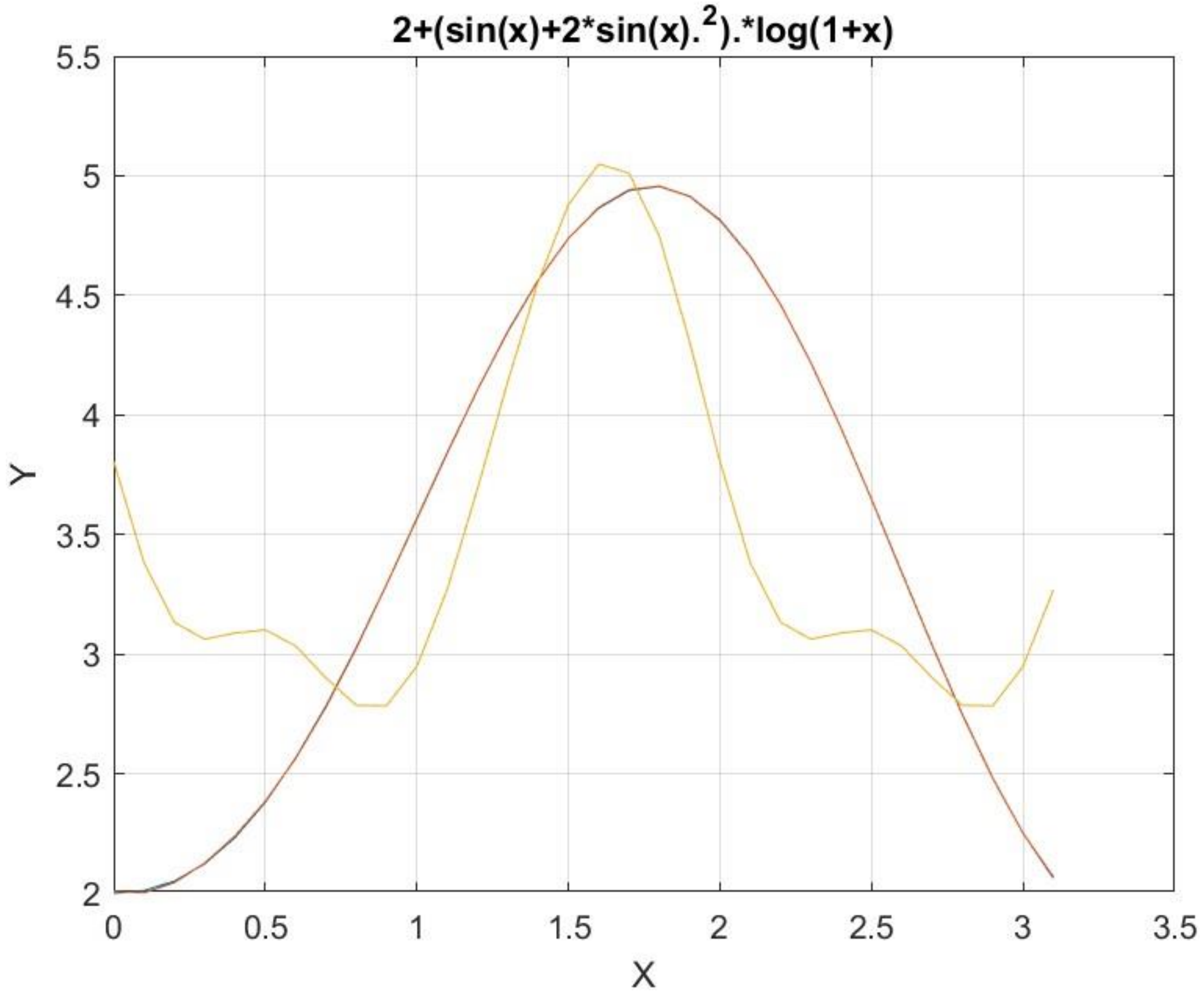


Figure 3. Graph for Fourier Series and test function in file sin_by_sqr_sin_halton_rand_fourier.jpg.

The above figure shows that the Quantum Shammas Fourier Series fit *function 1* well and much better than the regular Fourier series.

Testing Function 1 Fit with Sobol Random Search

The next MATLAB script (found in file found in file testFourierFx1Sobol.m) tests fitting *function 1* for x in the range (0, π) and samples at 0.1 steps. The curve fit uses the third order Quantum Shammas Fourier Series and a third order classical Fourier series.

```
clc
```

```
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "sin_by_sqr_sin_sobol_rand_fourier";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "2+(sin(x)+2*sin(x).^2).*log(1+x)";
fprintf("%s\n", sEqn);
xData = 0:.1:pi;
xData = xData';
n = length(xData);
yData = 2+(sin(xData)+2*sin(xData).^2).*log(1+xData);
fprintf("x=0:.1:pi\n")
order = 3;
[LbS,UbS] = makeLimits(order, 0.5, 1.4);
Lb = [];
Ub = [];
for i=1:order
  Lb = [Lb LbS(i) LbS(i)];
  Ub = [Ub UbS(i) UbS(i)];
end
[bestX,bestFx] =
sobolRandomSearch(@quantShammasFourierPoly,Lb,Ub,1000000);

SSE = quantShammasFourierPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Fourier Series factors\n");
bestX
fprintf("Quantum Shammas Fourier Series Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular Fourier fit\n");
sincos_factors = [];
for i=1:order
  sincos_factors =[sincos_factors i i];
end
sincos_factors
[c,r,yFourier] = fourierfit(xData,yData,order);
c = c'
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-2*order-1);
```

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

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + maxPwr;
  end
end
```

The above code generates the following Excel table of results.

QSPfactor1	QSPfactor2	QSPfactor3	QSPfactor4	QSPfactor5	QSPfactor6	
0.716296786	0.591576988	1.834246752	1.562118971	2.947699456	2.696088809	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
3.406199939	-0.268592635	-1.352121722	0.024608255	-0.033980554	0.010065469	-0.011252968

Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
3.683518044	-0.958336614	0.432447358	-0.18668984	-0.306651521	-0.069470767	-0.00427247
r_sqr1	r_sqr2					
0.999981839	0.316876372					

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

Again, the adjusted coefficient of determination for the fitted Quantum Shammas Fourier Series is well higher than the one for the regular Fourier series. This condition indicates that the Quantum Shammas Fourier Series performs a better fit for the above example.

Here is the graph for the test function and the two fitted polynomials:

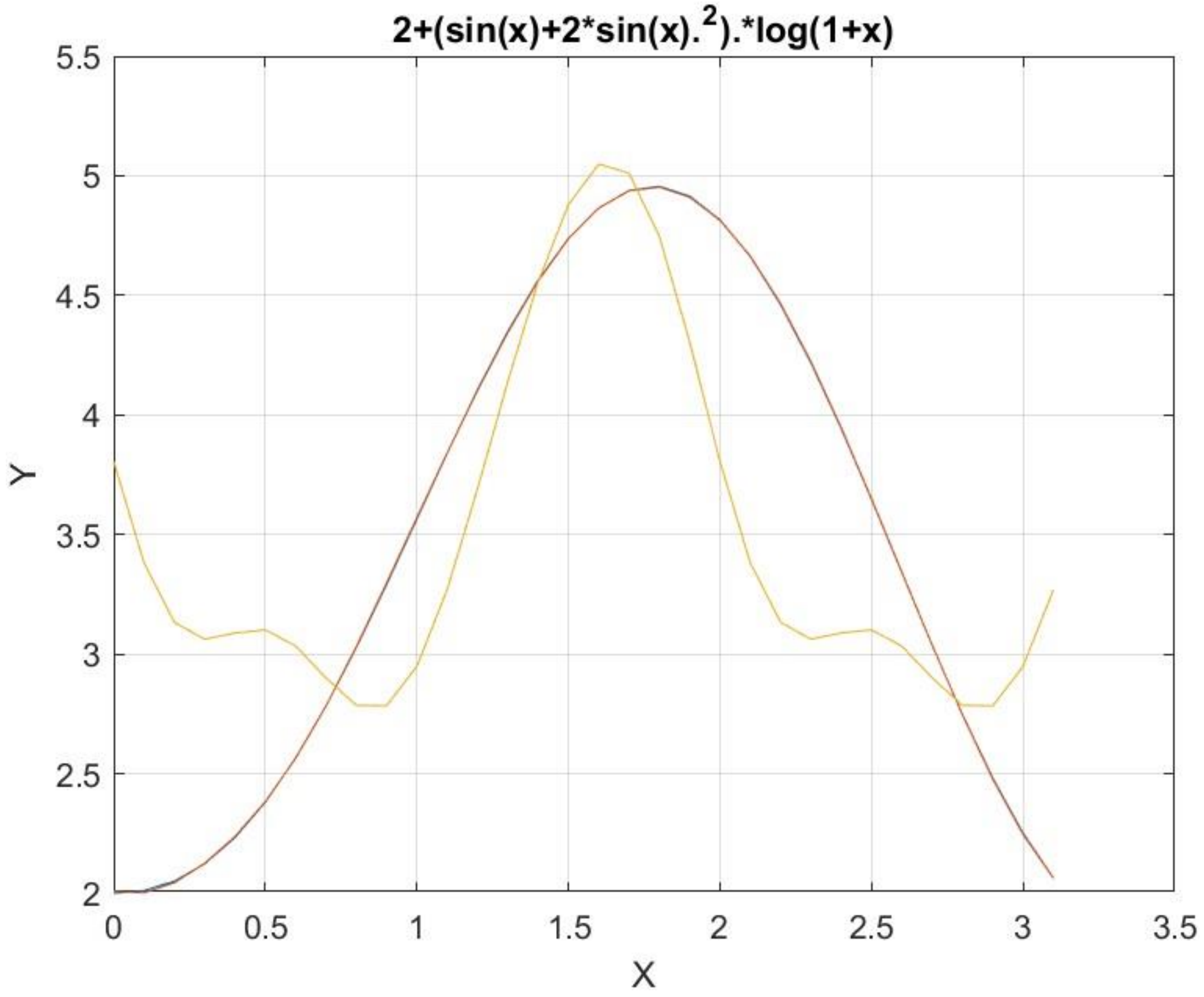


Figure 4. Graph for Fourier Series and test function in file sin_by_sqr_sin_sobol_rand_fourier.jpg.

The above figure shows that the Quantum Shammas Fourier Series fit *function 1* well and much better than the regular Fourier series.

Conclusion For Fitting Function 1

The above four sections showed that the Quantum Shammas Fourier Series fitted the test function much better than using regular Fourier series.

Testing Function 2 Fit with PSO

The next MATLAB script (found in file found in file testFourierFx2pso.m) tests fitting *function 1* for x in the range $(0, \pi)$ and samples at 0.1 steps. The curve fit uses the third order Quantum Shammas Fourier Series and a third order classical Fourier series.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "sqr_sin_by_sinh_fourier_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 = "(sin(2*x).^2).*sinh(x/5)";
fprintf("%s\n", sEqn);
xData = 0:.1:pi;
xData = xData';
n = length(xData);
yData = (sin(2*xData).^2).*sinh(xData/5);
fprintf("x=0:.1:pi\n")
order = 3;
[LbS,UbS] = makeLimits(order, 0.5, 1.4);
Lb = [];
Ub = [];
for i=1:order
  Lb = [Lb LbS(i) LbS(i)];
  Ub = [Ub UbS(i) UbS(i)];
end
[bestX,bestFx] =
psox(@quantShammasFourierPoly,Lb,Ub,1000,5000,true);

SSE = quantShammasFourierPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Fourier Series factors\n");
bestX
fprintf("Quantum Shammas Fourier Series Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular Fourier fit\n");
```

```
sincos_factors = [];
for i=1:order
  sincos_factors =[sincos_factors i i];
end
sincos_factors
[c,r,yFourier] = fourierfit(xData,yData,order);
c = c'
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,yCalc,xData,yFourier);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + maxPwr;
  end
end
```

The above code generates the following Excel table of results.

QSPfactor1	QSPfactor2	QSPfactor3	QSPfactor4	QSPfactor5	QSPfactor6	
0.662839342	1.215823022	1.984110805	2.359647517	2.505758423	2.972764724	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
0.155098876	-0.148916484	-0.174799882	0.036661986	0.015134066	0.004054788	0.004471241
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.13694724	0.123188494	0.003034555	0.007604924	-0.010429409	-0.010799934	0.009984818
r_sqr1	r_sqr2					
0.999912519	0.121879049					

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

Again, the adjusted coefficient of determination for the fitted Quantum Shammas Fourier Series is well higher than the one for the regular Fourier series. This condition indicates that the Quantum Shammas Fourier Series performs a better fit for the above example.

Here is the graph for the test function and the two fitted polynomials:

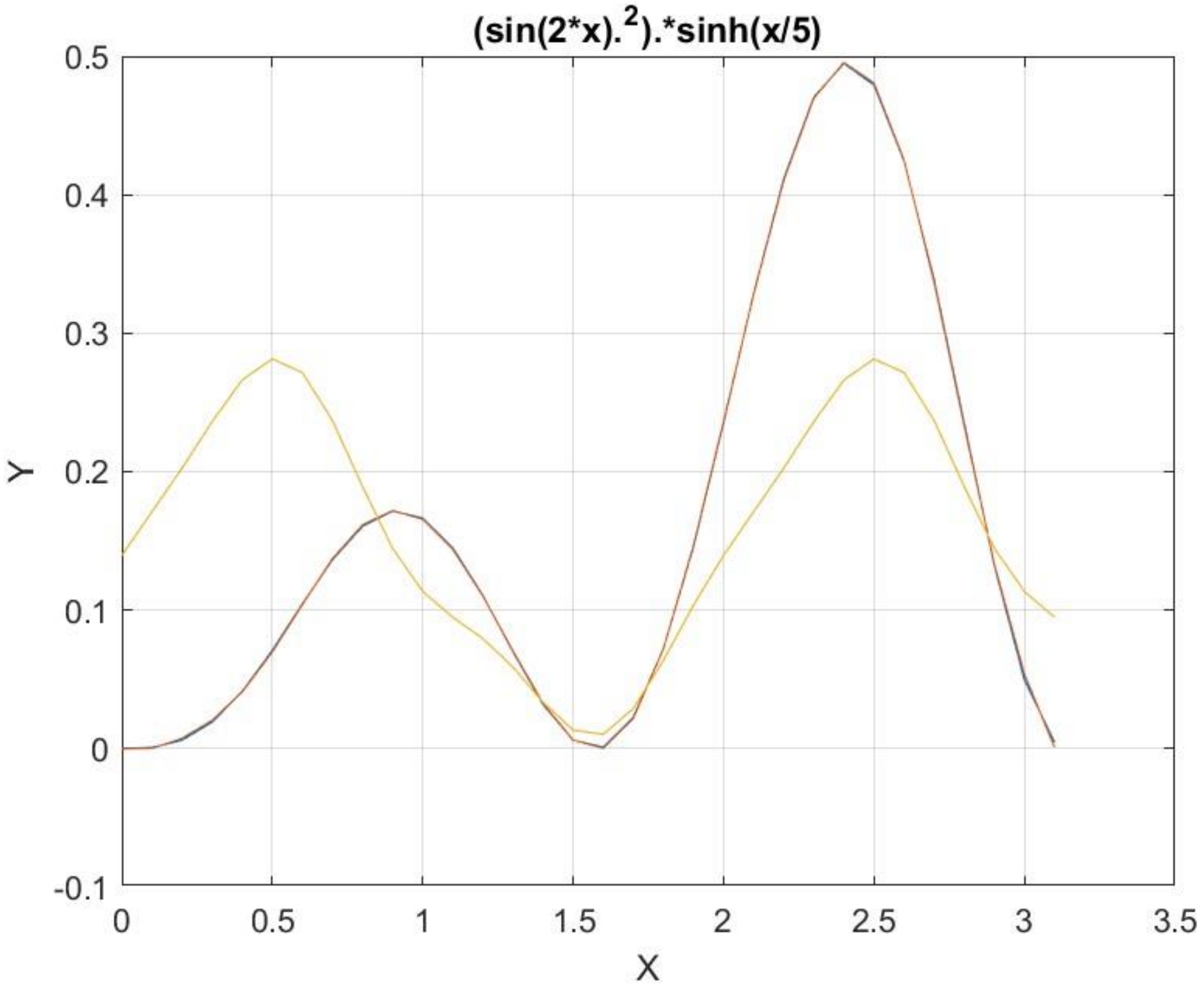


Figure 5. Graph for Fourier Series and test function in file sqr_sin_by_sinh_fourier_pso.jpg.

The above figure shows that the Quantum Shammas Fourier Series fit *function 2* well and much better than the regular Fourier series.

Testing Function 2 Fit with Random Search

The next MATLAB script (found in file found in file testFourierFx2Random.m) tests fitting *function 2* for x in the range (0, π) and samples at 0.1 steps. The curve fit uses the third order Quantum Shammas Fourier Series and a third order classical Fourier series.

```
clc
```

```
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "sqr_sin_by_sinh_rand_fourier";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "(sin(2*x).^2).*sinh(x/5)";
fprintf("%s\n", sEqn);
xData = 0:.1:pi;
xData = xData';
n = length(xData);
yData = (sin(2*xData).^2).*sinh(xData/5);
fprintf("x=0:.1:pi\n")
order = 3;
[LbS,UbS] = makeLimits(order, 0.5, 1.4);
Lb = [];
Ub = [];
for i=1:order
  Lb = [Lb LbS(i) LbS(i)];
  Ub = [Ub UbS(i) UbS(i)];
end
[bestX,bestFx] =
randomSearch(@quantShammasFourierPoly,Lb,Ub,1000000);

SSE = quantShammasFourierPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Fourier Series factors\n");
bestX
fprintf("Quantum Shammas Fourier Series Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular Fourier fit\n");
sincos_factors = [];
for i=1:order
  sincos_factors =[sincos_factors i i];
end
sincos_factors
[c,r,yFourier] = fourierfit(xData,yData,order);
c = c'
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-2*order-1);
```

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

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + maxPwr;
  end
end
```

The above code generates the following Excel table of results.

QSPfactor1	QSPfactor2	QSPfactor3	QSPfactor4	QSPfactor5	QSPfactor6	
0.66145906	1.217417181	1.978081887	2.311965034	2.660102771	3.080814908	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
0.155195294	-0.14817916	-0.174205844	0.039010178	0.018677987	-0.000635501	0.002343905

Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.13694724	0.123188494	0.003034555	0.007604924	-0.010429409	-0.010799934	0.009984818
r_sqr1	r_sqr2					
0.99984577	0.121879049					

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

Again, the adjusted coefficient of determination for the fitted Quantum Shammas Fourier Series is well higher than the one for the regular Fourier series. This condition indicates that the Quantum Shammas Fourier Series performs a better fit for the above example.

Here is the graph for the test function and the two fitted polynomials:

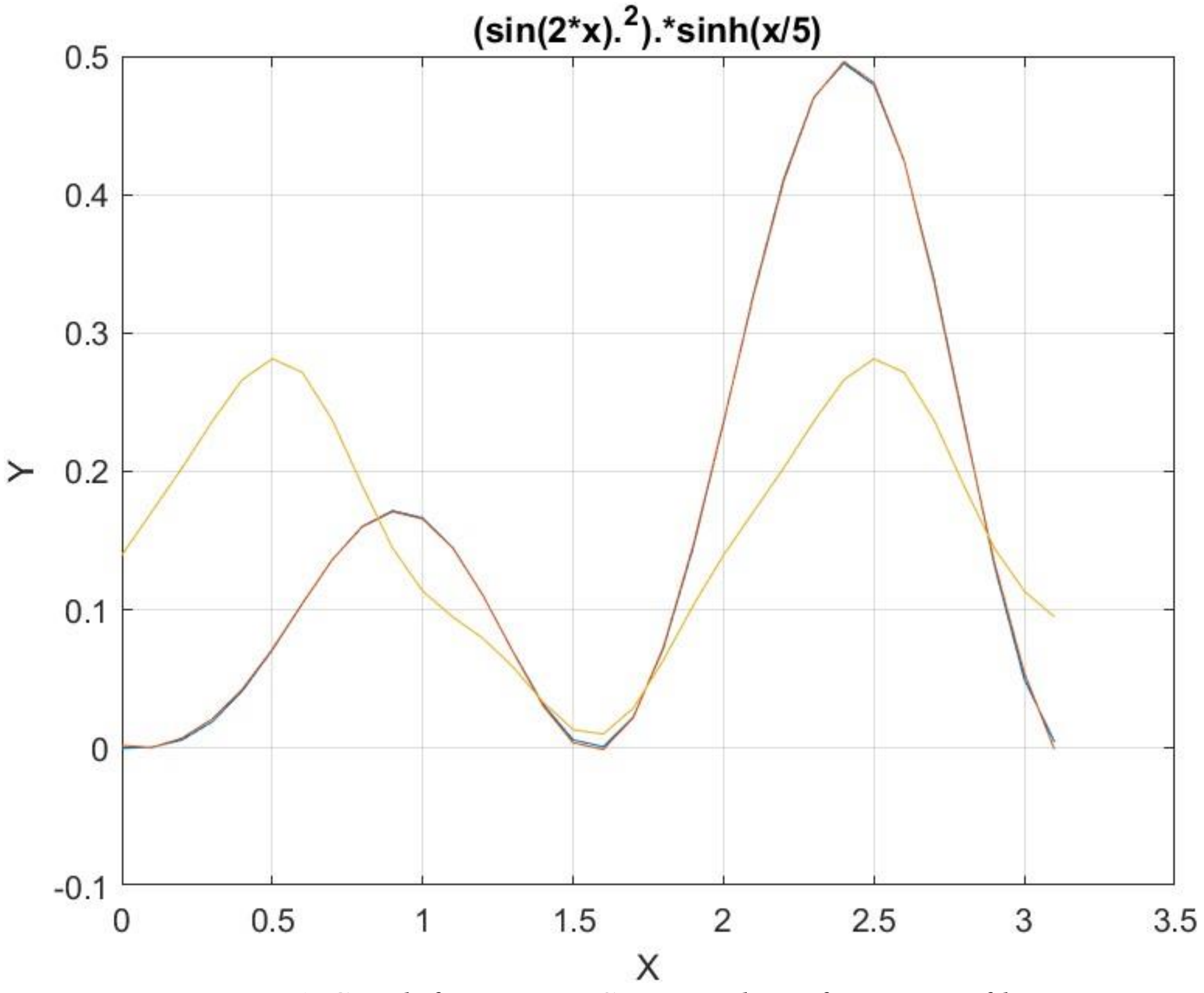


Figure 6. Graph for Fourier Series and test function in file sqr_sin_by_sinh_rand_fourier.jpg.

The above figure shows that the Quantum Shammas Fourier Series fit *function 2* well and much better than the regular Fourier series.

Testing Function 2 Fit with Halton Random Search

The next MATLAB script (found in file found in file testFourierFx2Halton.m) tests fitting *function 2* for x in the range (0, π) and samples at 0.1 steps. The curve fit uses the third order Quantum Shammas Fourier Series and a third order classical Fourier series.

```
clc
```

```
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "sqr_sin_by_sinh_halton_rand_fourier";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "(sin(2*x).^2).*sinh(x/5)";
fprintf("%s\n", sEqn);
xData = 0:.1:pi;
xData = xData';
n = length(xData);
yData = (sin(2*xData).^2).*sinh(xData/5);
fprintf("x=0:.1:pi\n")
order = 3;
[LbS,UbS] = makeLimits(order, 0.5, 1.4);
Lb = [];
Ub = [];
for i=1:order
  Lb = [Lb LbS(i) LbS(i)];
  Ub = [Ub UbS(i) UbS(i)];
end
[bestX,bestFx] =
haltonRandomSearch(@quantShammasFourierPoly,Lb,Ub,1000000);

SSE = quantShammasFourierPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Fourier Series factors\n");
bestX
fprintf("Quantum Shammas Fourier Series Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular Fourier fit\n");
sincos_factors = [];
for i=1:order
  sincos_factors =[sincos_factors i i];
end
sincos_factors
[c,r,yFourier] = fourierfit(xData,yData,order);
c = c'
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-2*order-1);
```

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

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + maxPwr;
  end
end
```

The above code generates the following Excel table of results.

QSPfactor1	QSPfactor2	QSPfactor3	QSPfactor4	QSPfactor5	QSPfactor6	
0.66145906	1.217417181	1.978081887	2.311965034	2.660102771	3.080814908	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
0.155195294	-0.14817916	-0.174205844	0.039010178	0.018677987	-0.000635501	0.002343905

Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.13694724	0.123188494	0.003034555	0.007604924	-0.010429409	-0.010799934	0.009984818
r_sqr1	r_sqr2					
0.99984577	0.121879049					

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

The adjusted coefficient of determination for the fitted Quantum Shammas Fourier Series is well higher than the one for the regular Fourier series. This condition indicates that the Quantum Shammas Fourier Series performs a better fit for the above example.

Here is the graph for the test function and the two fitted polynomials:

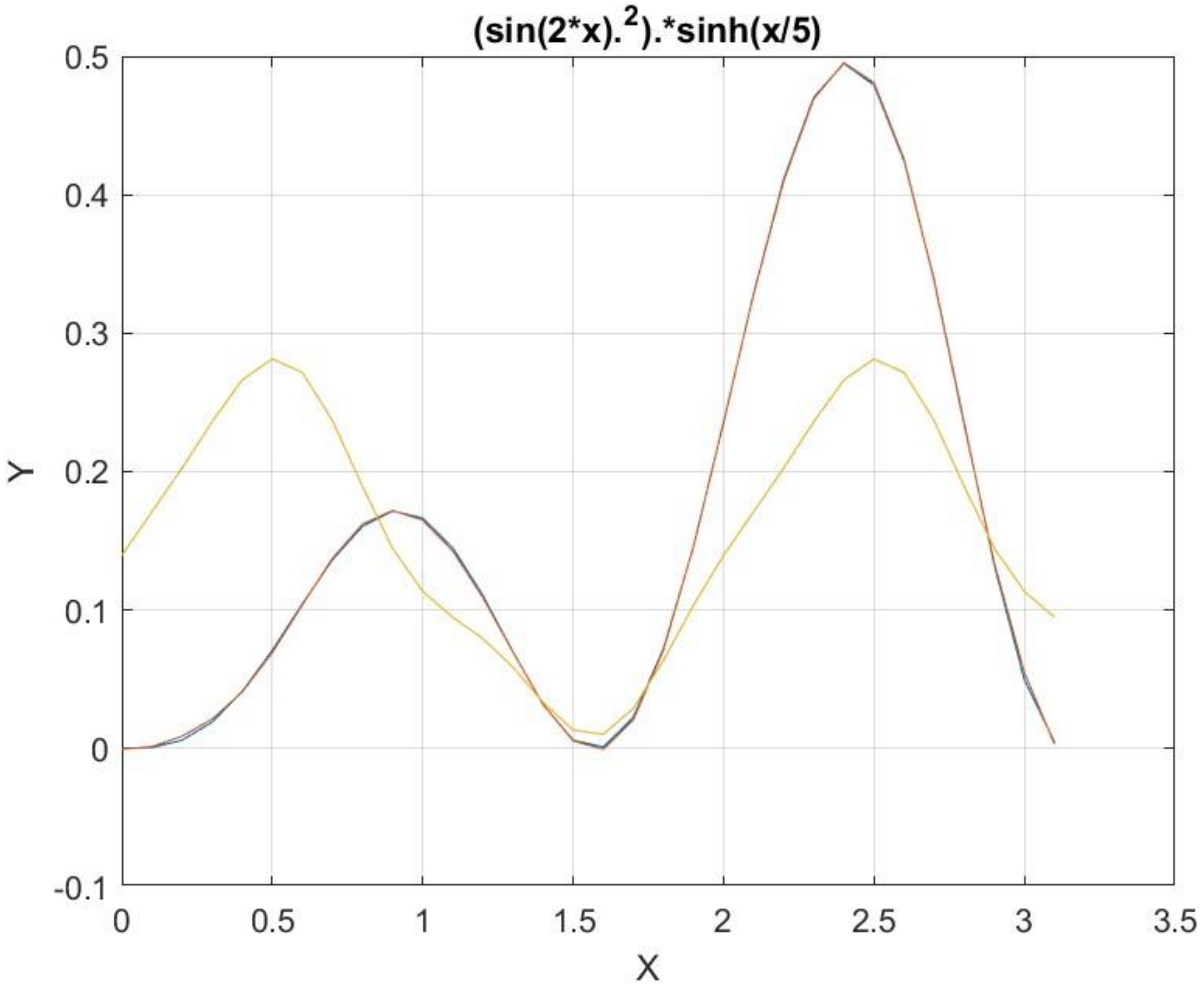


Figure 7. Graph for Fourier Series and test function in file sqr_sin_by_sinh_halton_rand_fourier.jpg.

The above figure shows that the Quantum Shammas Fourier Series fit *function 2* well and much better than the regular Fourier series.

Testing Function 2 Fit with Sobol Random Search

The next MATLAB script (found in file found in file testFourierFx2Sobol.m) tests fitting *function 2* for x in the range (0, π) and samples at 0.1 steps. The curve fit uses the third order Quantum Shammas Fourier Series and a third order classical Fourier series.

```
clc
```

```
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "sqr_sin_by_sinh_sobol_rand_fourier";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "(sin(2*x).^2).*sinh(x/5)";
fprintf("%s\n", sEqn);
xData = 0:.1:pi;
xData = xData';
n = length(xData);
yData = (sin(2*xData).^2).*sinh(xData/5);
fprintf("x=0:.1:pi\n")
order = 3;
[LbS,UbS] = makeLimits(order, 0.5, 1.4);
Lb = [];
Ub = [];
for i=1:order
  Lb = [Lb LbS(i) LbS(i)];
  Ub = [Ub UbS(i) UbS(i)];
end
[bestX,bestFx] =
sobolRandomSearch(@quantShammasFourierPoly,Lb,Ub,1000000);

SSE = quantShammasFourierPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Fourier Series factors\n");
bestX
fprintf("Quantum Shammas Fourier Series Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular Fourier fit\n");
sincos_factors = [];
for i=1:order
  sincos_factors =[sincos_factors i i];
end
sincos_factors
[c,r,yFourier] = fourierfit(xData,yData,order);
c = c'
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-2*order-1);
```

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

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + maxPwr;
  end
end
```

The above code generates the following Excel table of results.

QSPfactor1	QSPfactor2	QSPfactor3	QSPfactor4	QSPfactor5	QSPfactor6	
0.661987678	1.219601694	1.865388363	1.864163393	2.643311036	3.020374286	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
0.155541492	-0.147344852	-0.173175289	0.024556806	0.014250181	0.008883057	0.004974452

Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.13694724	0.123188494	0.003034555	0.007604924	-0.010429409	-0.010799934	0.009984818
r_sqr1	r_sqr2					
0.999792874	0.121879049					

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

The adjusted coefficient of determination for the fitted Quantum Shammas Fourier Series is well higher than the one for the regular Fourier series. This condition indicates that the Quantum Shammas Fourier Series performs a better fit for the above example.

Here is the graph for the test function and the two fitted polynomials:

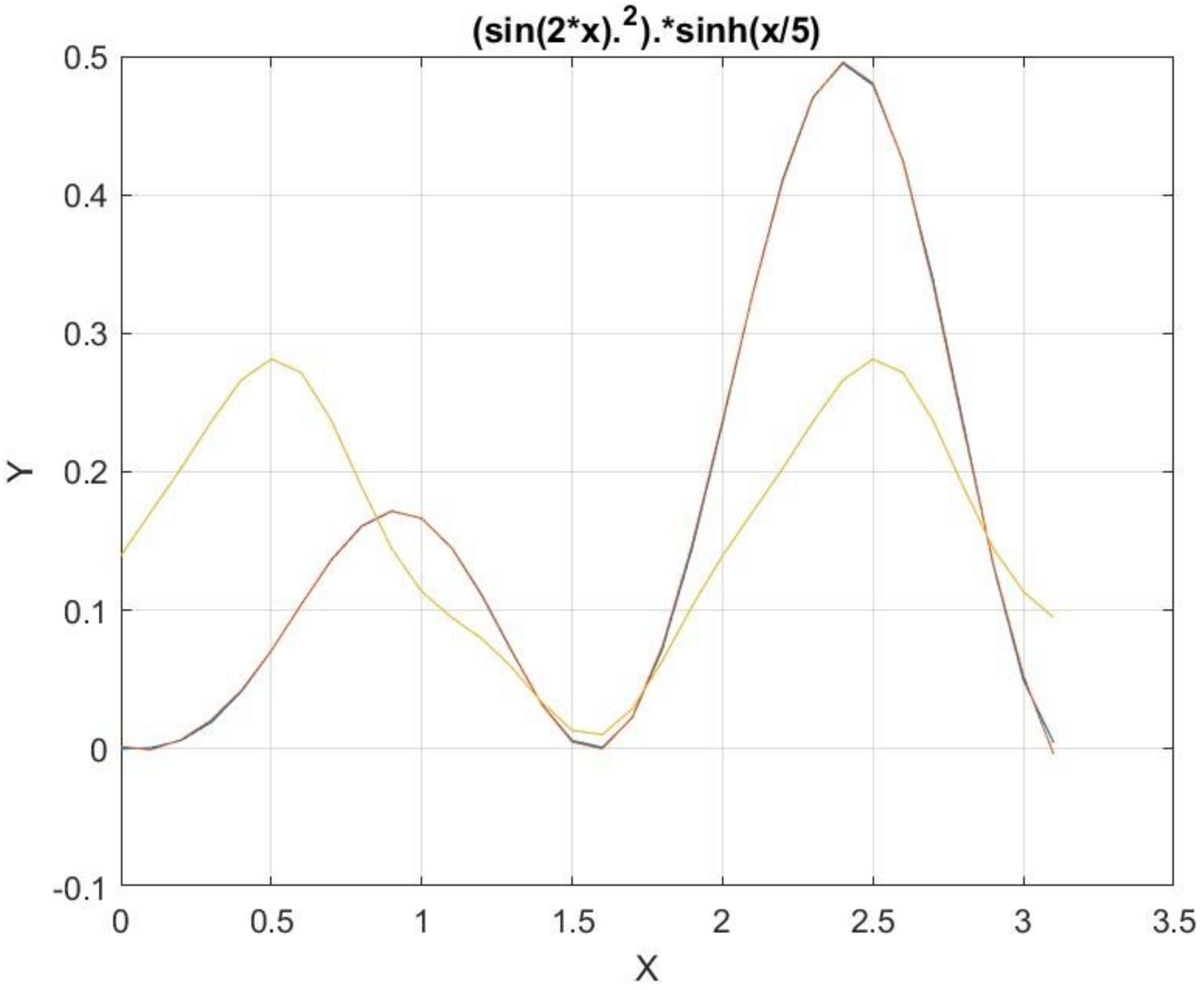


Figure 8. Graph for Fourier Series and test function in file sin_by_sqr_sinh_sobol_rand_fourier.jpg.

The above figure shows that the Quantum Shammas Fourier Series fit *function 2* well and much better than the regular Fourier series.

Conclusion for fitting Function 2

The above four sections showed that the Quantum Shammas Fourier Series fitted the test function much better than using regular Fourier series.

Testing Function 3 Fit with PSO

The next MATLAB script (found in file found in file testFourierFx3pso.m) tests fitting *function 3* for x in the range (0, π) and samples at 0.1 steps. The curve fit uses the third order Quantum Shammas Fourier Series and a third order classical Fourier series.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj0_by_sin_fourier_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 = "1+bessel(0,x)*sin(x)";
fprintf("%s\n", sEqn);
xData = 0:.1:pi;
xData = xData';
n = length(xData);
yData = 1+besselj(0,xData).*sin(xData);
fprintf("x=0:.1:pi\n")
order = 3;
[LbS,UbS] = makeLimits(order, 0.5, 1.4);
Lb = [];
Ub = [];
for i=1:order
  Lb = [Lb LbS(i) LbS(i)];
  Ub = [Ub UbS(i) UbS(i)];
end
[bestX,bestFx] =
psox(@quantShammasFourierPoly,Lb,Ub,1000,5000,true);

SSE = quantShammasFourierPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Fourier Series factors\n");
bestX
fprintf("Quantum Shammas Fourier Series Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular Fourier fit\n");
sincos_factors = [];
for i=1:order
  sincos_factors =[sincos_factors i i];
end
sincos_factors
```

```
[c,r,yFourier] = fourierfit(xData,yData,order);
c = c'
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,yCalc,xData,yFourier);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + maxPwr;
  end
end
```

The above code generates the following Excel table of results.

QSPfactor1	QSPfactor2	QSPfactor3	QSPfactor4	QSPfactor5	QSPfactor6	
0.596194012	0.500031555	1.628945357	1.93623116	2.852129718	2.551351763	

QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.242463259	0.417654982	-0.195964609	-0.022691257	-0.023532117	0.010547698	-0.015687139
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
1.311833902	-0.165956535	-0.094721547	0.018923166	-0.076587451	-0.003160329	0.015716183
r_sqr1	r_sqr2					
0.999888688	0.088234629					

Table 9. Summary of the results appearing in file besselj0_by_sin_fourier_pso.xlsx

The adjusted coefficient of determination for the fitted Quantum Shammas Fourier Series is well higher than the one for the regular Fourier series. This condition indicates that the Quantum Shammas Fourier Series performs a better fit for the above example.

Here is the graph for the test function and the two fitted polynomials:

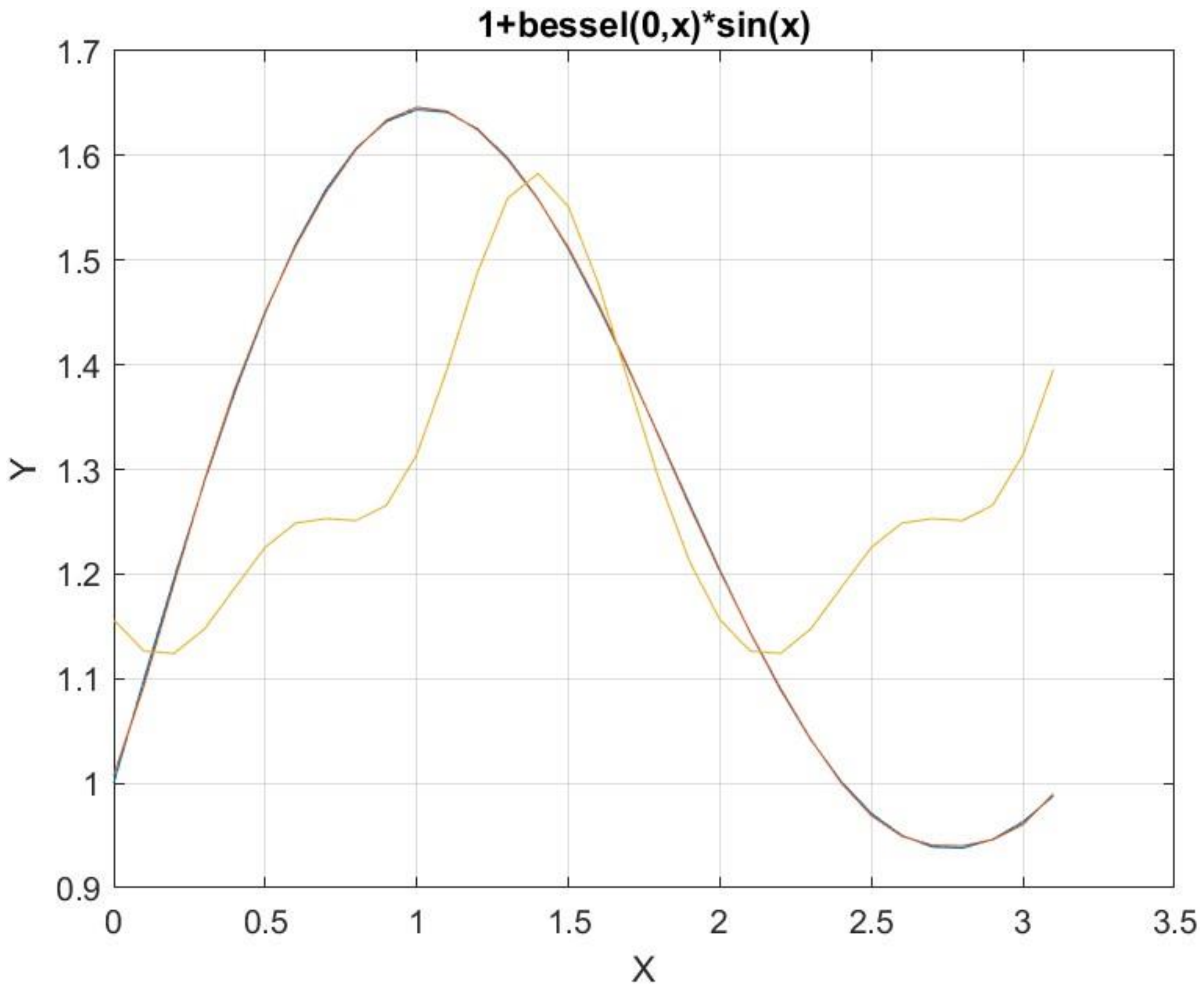


Figure 9. Graph for Fourier Series and test function in file besselj0_by_sin_fourier_pso.jpg.

The above figure shows that the Quantum Shammas Fourier Series fit *function 3* well and much better than the regular Fourier series.

Testing Function 3 Fit with Random Search

The next MATLAB script (found in file found in file testFourierFx3Random.m) tests fitting *function 3* for x in the range (0, π) and samples at 0.1 steps. The curve fit uses the third order Quantum Shammas Fourier Series and a third order classical Fourier series.

```
clc
```

```
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj0_by_sin_rand_fourier";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "1+bessel(0,x)*sin(x)";
fprintf("%s\n", sEqn);
xData = 0:.1:pi;
xData = xData';
n = length(xData);
yData = 1+besselj(0,xData).*sin(xData);
fprintf("x=0:.1:pi\n")
order = 3;
[LbS,UbS] = makeLimits(order, 0.5, 1.4);
Lb = [];
Ub = [];
for i=1:order
  Lb = [Lb LbS(i) LbS(i)];
  Ub = [Ub UbS(i) UbS(i)];
end
[bestX,bestFx] =
randomSearch(@quantShammasFourierPoly,Lb,Ub,1000000);

SSE = quantShammasFourierPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Fourier Series factors\n");
bestX
fprintf("Quantum Shammas Fourier Series Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular Fourier fit\n");
sincos_factors = [];
for i=1:order
  sincos_factors =[sincos_factors i i];
end
sincos_factors
[c,r,yFourier] = fourierfit(xData,yData,order);
c = c'
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-2*order-1);
```

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

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + maxPwr;
  end
end
```

The above code generates the following Excel table of results.

QSPfactor1	QSPfactor2	QSPfactor3	QSPfactor4	QSPfactor5	QSPfactor6	
0.577972206	0.453690309	1.573787215	1.881730163	2.783579676	2.50117391	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.231662596	0.449960455	-0.1907735	-0.020271723	-0.020185345	0.009087071	-0.013394779

Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
1.311833902	-0.165956535	-0.094721547	0.018923166	-0.076587451	-0.003160329	0.015716183
r_sqr1	r_sqr2					
0.99990544	0.088234629					

Table 10. Summary of the results appearing in file besselj0_by_sin_rand_fourier.xlsx.

The adjusted coefficient of determination for the fitted Quantum Shammas Fourier Series is well higher than the one for the regular Fourier series. This condition indicates that the Quantum Shammas Fourier Series performs a better fit for the above example.

Here is the graph for the test function and the two fitted polynomials:

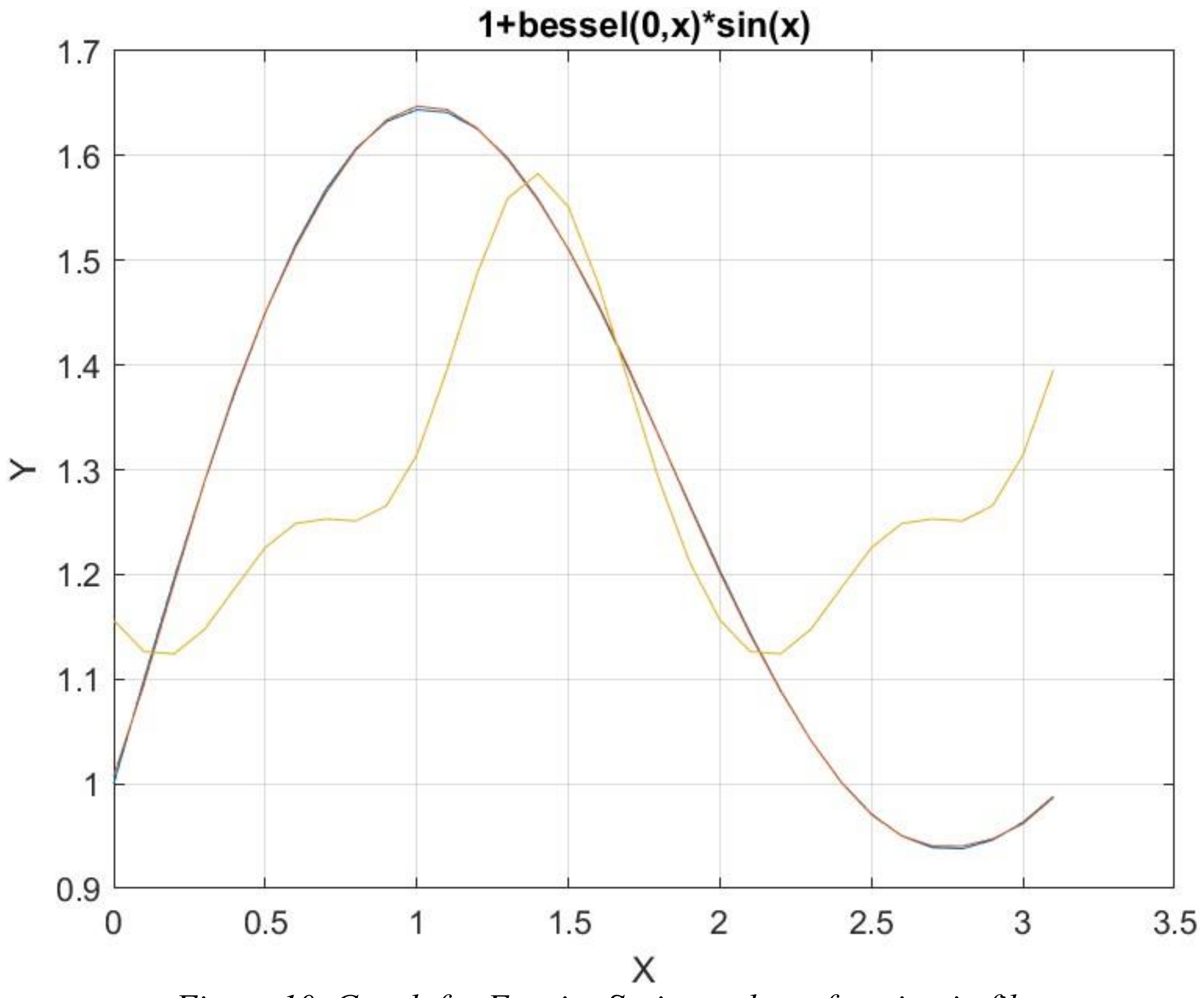


Figure 10. Graph for Fourier Series and test function in file besselj0_by_sin_rand_fourier.jpg.

The above figure shows that the Quantum Shammas Fourier Series fit *function 3* well and better than the regular Fourier series.

Testing Function 3 Fit with Halton Random Search

The next MATLAB script (found in file found in file testFourierFx3Halton.m) tests fitting *function 3* for x in the range (0, π) and samples at 0.1 steps. The curve fit uses the third order Quantum Shammas Fourier Series and a third order classical Fourier series.

```
clc
```

```
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj0_by_sin_halton_rand_fourier";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "1+bessel(0,x)*sin(x)";
fprintf("%s\n", sEqn);
xData = 0:.1:pi;
xData = xData';
n = length(xData);
yData = 1+besselj(0,xData).*sin(xData);
fprintf("x=0:.1:pi\n")
order = 3;
[LbS,UbS] = makeLimits(order, 0.5, 1.4);
Lb = [];
Ub = [];
for i=1:order
  Lb = [Lb LbS(i) LbS(i)];
  Ub = [Ub UbS(i) UbS(i)];
end
[bestX,bestFx] =
haltonRandomSearch(@quantShammasFourierPoly,Lb,Ub,1000000);

SSE = quantShammasFourierPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Fourier Series factors\n");
bestX
fprintf("Quantum Shammas Fourier Series Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular Fourier fit\n");
sincos_factors = [];
for i=1:order
  sincos_factors =[sincos_factors i i];
end
sincos_factors
[c,r,yFourier] = fourierfit(xData,yData,order);
c = c'
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-2*order-1);
```

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

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + maxPwr;
  end
end
```

The above code generates the following Excel table of results.

QSPfactor1	QSPfactor2	QSPfactor3	QSPfactor4	QSPfactor5	QSPfactor6	
0.601285317	0.522675741	1.60168413	1.887947757	2.76646179	2.473773618	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.248757207	0.396451004	-0.19185405	-0.027283552	-0.028627657	0.011864534	-0.018197068

Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
1.311833902	-0.165956535	-0.094721547	0.018923166	-0.076587451	-0.003160329	0.015716183
r_sqr1	r_sqr2					
0.999811958	0.088234629					

Table 11. Summary of the results appearing in file besselj0_by_sin_halton_rand_fourier.xlsx.

The adjusted coefficient of determination for the fitted Quantum Shammas Fourier Series is well higher than the one for the regular Fourier series. This condition indicates that the Quantum Shammas Fourier Series performs a better fit for the above example.

Here is the graph for the test function and the two fitted polynomials:

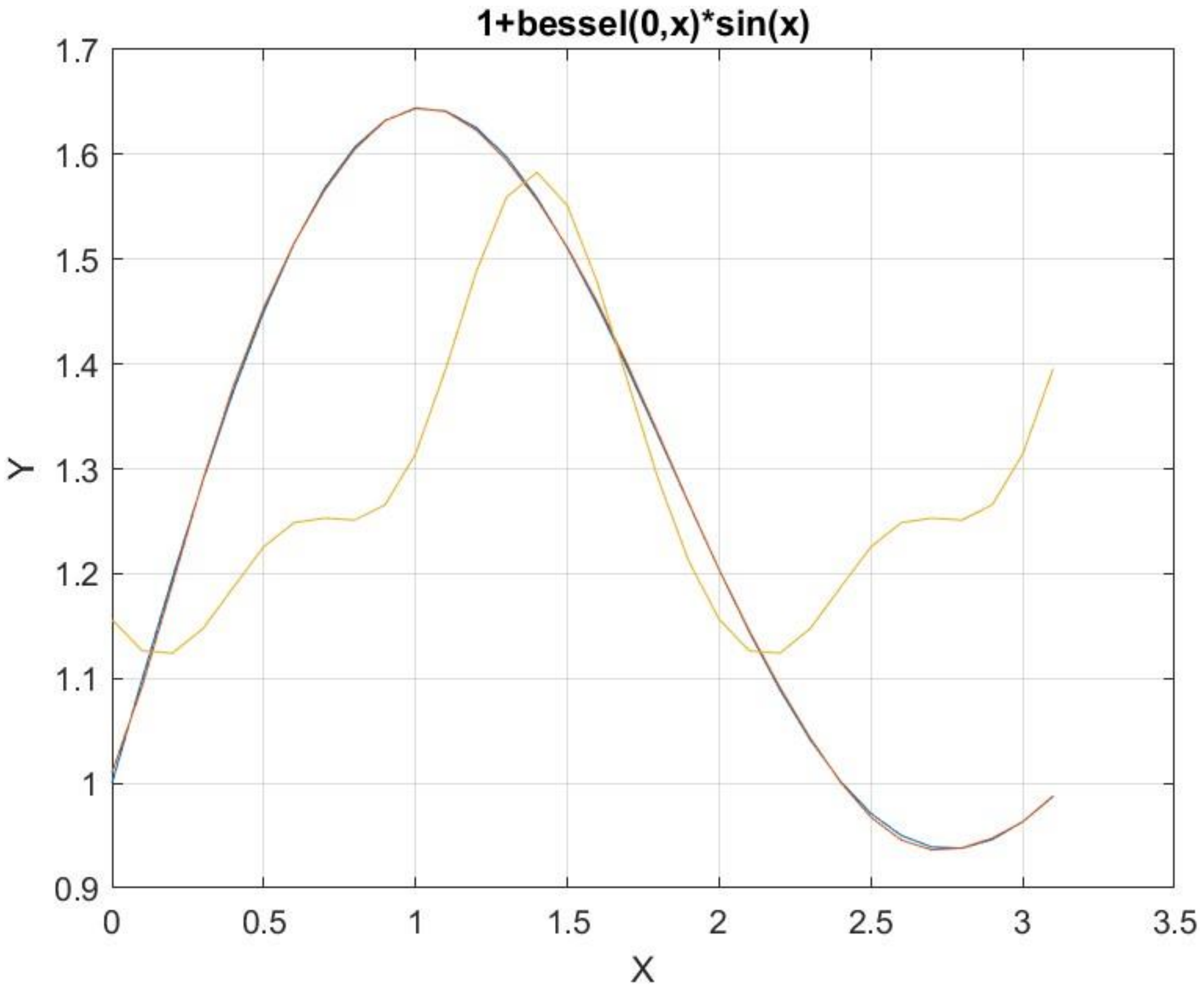


Figure 11. Graph for Fourier Series and test function in file besselj0_by_sin_halton_rand_fourier.jpg.

The above figure shows that the Quantum Shammas Fourier Series fit *function 3* well and much better than the regular Fourier series.

Testing Function 3 Fit with Sobol Random Search

The next MATLAB script (found in file found in file testFourierFx3Sobol.m) tests fitting *function 3* for x in the range (0, π) and samples at 0.1 steps. The curve fit uses the third order Quantum Shammas Fourier Series and a third order classical Fourier series.

```
clc
clear
```

```
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj0_by_sin_sobol_rand_fourier";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "1+bessel(0,x)*sin(x)";
fprintf("%s\n", sEqn);
xData = 0:.1:pi;
xData = xData';
n = length(xData);
yData = 1+besselj(0,xData).*sin(xData);
fprintf("x=0:.1:pi\n")
order = 3;
[LbS,UbS] = makeLimits(order, 0.5, 1.4);
Lb = [];
Ub = [];
for i=1:order
  Lb = [Lb LbS(i) LbS(i)];
  Ub = [Ub UbS(i) UbS(i)];
end
[bestX,bestFx] =
sobolRandomSearch(@quantShammasFourierPoly,Lb,Ub,1000000);

SSE = quantShammasFourierPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Fourier Series factors\n");
bestX
fprintf("Quantum Shammas Fourier Series Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular Fourier fit\n");
sincos_factors = [];
for i=1:order
  sincos_factors =[sincos_factors i i];
end
sincos_factors
[c,r,yFourier] = fourierfit(xData,yData,order);
c = c'
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", r);
```

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

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + maxPwr;
  end
end
```

The above code generates the following Excel table of results.

QSPfactor1	QSPfactor2	QSPfactor3	QSPfactor4	QSPfactor5	QSPfactor6	
0.621190928	0.559572654	1.654834776	1.975340166	2.293678798	2.643235348	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.260715095	0.369232093	-0.206541165	-0.041636652	-0.063460511	0.043649265	0.013564997
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7

1.311833902	-0.165956535	-0.094721547	0.018923166	-0.076587451	-0.003160329	0.015716183
r_sqr1	r_sqr2					
0.99989106	0.088234629					

Table 12. Summary of the results appearing in file besselj0_by_sin_sobol_rand_fourier.xlsx.

The adjusted coefficient of determination for the fitted Quantum Shammas Fourier Series is well higher than the one for the regular Fourier series. This condition indicates that the Quantum Shammas Fourier Series performs a better fit for the above example.

Here is the graph for the test function and the two fitted polynomials:

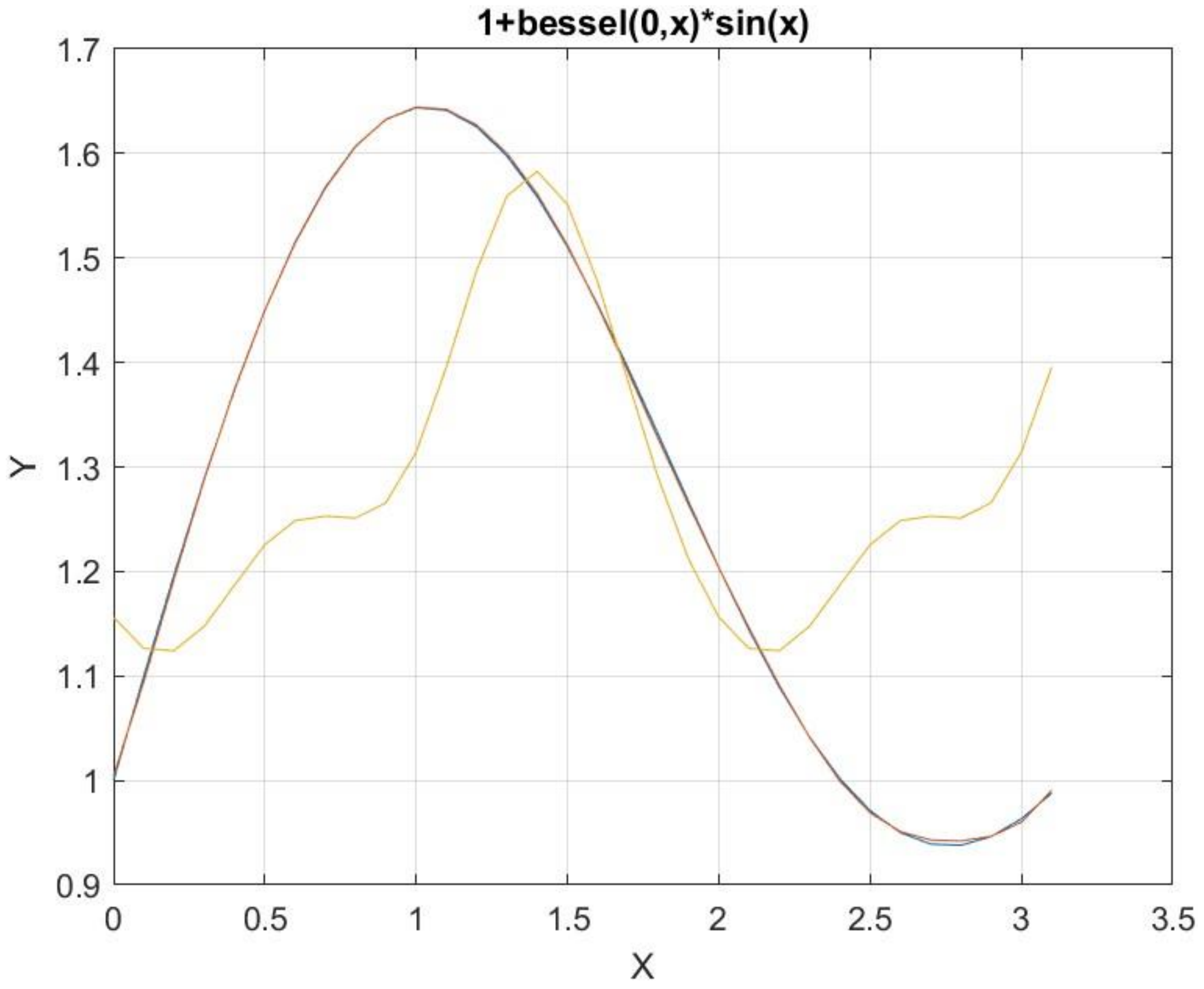


Figure 12. Graph for Fourier Series and test function in file besselj0_by_sin_sobol_rand_fourier.jpg.

The above figure shows that the Quantum Shammas Fourier Series fit *function 3* well and much better than the regular Fourier series.

Conclusion for Fitting Function 3

The above four sections showed that the Quantum Shammas Fourier Series fitted the test function much better than using regular Fourier series.

Testing Function 4 Fit with PSO

The next MATLAB script (found in file found in file testFourierFx4pso.m) tests fitting *function 4* for x in the range (0, π) and samples at 0.05 steps. The curve fit uses the fourth order Quantum Shammas Fourier Series and a fourth order classical Fourier series.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "sin_sum_by_cos_sum_fourier_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 = "1 + (sin(x) - 2*sin(2*x-pi) + 3*sin(3*x-pi))*(cos(x) -
2*cos(2*x-pi) + 3*cos(3*x-pi))";
fprintf("%s\n", sEqn);
xData = 0:.05:pi;
xData = xData';
n = length(xData);
yData = 1 + (sin(xData) - 2*sin(2*xData-pi) + 3*sin(3*xData-
pi)).* ...
            (cos(xData) - 2*cos(2*xData-pi) + 3*cos(3*xData-
pi));
fprintf("x=0:.05:pi\n")
order = 4;
[LbS,UbS] = makeLimits(order, 0.5, 1.4);
Lb = [];
Ub = [];
for i=1:order
  Lb = [Lb LbS(i) LbS(i)];
  Ub = [Ub UbS(i) UbS(i)];
end
[bestX,bestFx] =
psox(@quantShammasFourierPoly,Lb,Ub,1000,5000,true);

SSE = quantShammasFourierPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Fourier Series factors\n");
```

```
bestX
fprintf("Quantum Shammas Fourier Series Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular Fourier fit\n");
sincos_factors = [];
for i=1:order
  sincos_factors =[sincos_factors i i];
end
sincos_factors
[c,r,yFourier] = fourierfit(xData,yData,order);
c = c'
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,yCalc,xData,yFourier);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + maxPwr;
  end
end
```

The above code generates the following Excel table of results.

QSPfactor1	QSPfactor2	QSPfactor3	QSPfactor4	QSPfactor5
0.934677423	0.651383259	2.343773846	1.786378606	2.966685201
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5
-0.002827182	6.362120086	-3.981620632	-3.190880551	7.201168116
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
-0.068665115	3.185240212	-1.670822563	5.458038604	0.608478661
r_sqr1	r_sqr2			
0.999994207	0.503588019			

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

QSPfactor6	QSPfactor7	QSPfactor8	
2.648301342	3.640914739	3.954891161	
QSPcoeff6	QSPcoeff7	QSPcoeff8	QSPcoeff9
0.707395646	-2.274239452	0.067381977	0.009605781
Coeff6	Coeff7	Coeff8	Coeff9
-0.475703429	0.871416365	0.092743087	-0.118935733

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

The adjusted coefficient of determination for the fitted Quantum Shammas Fourier Series is much higher than the one for the regular Fourier series. This condition indicates that the Quantum Shammas Fourier Series performs a better fit for the above example.

Here is the graph for the test function and the two fitted polynomials:

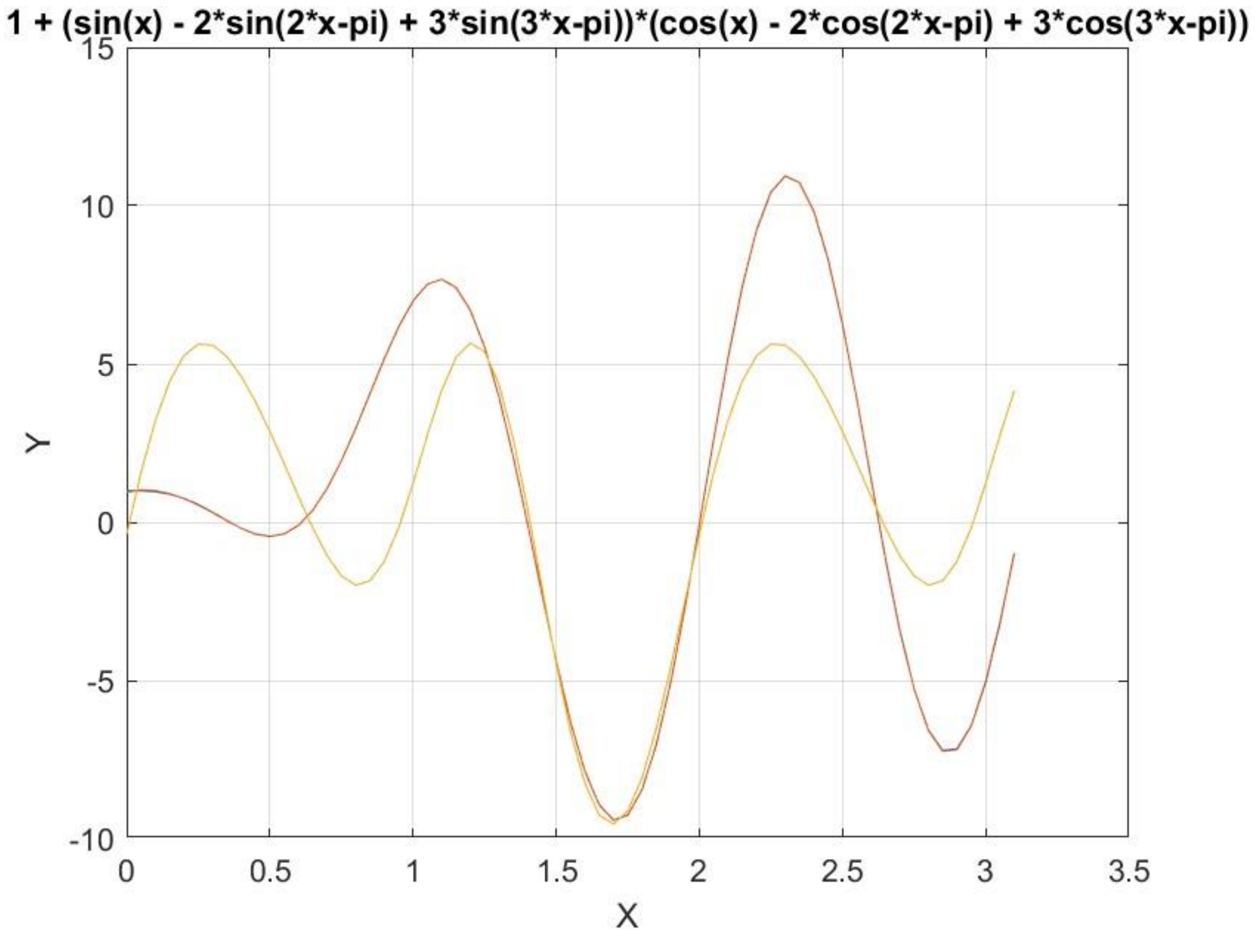


Figure 13. Graph for Fourier Series and test function in file sin_sum_by_cos_sum_fourier_pso.jpg.

The above figure shows that the Quantum Shammas Fourier Series fit *function 4* well and much better than the regular Fourier series.

Testing Function 4 Fit with Random Search

The next MATLAB script (found in file found in file testFourierFx4Random.m) tests fitting *function 4* for x in the range (0, 2π) and samples at 0.05 steps. The curve fit uses the fourth order Quantum Shammas Fourier Series and a fourth order classical Fourier series.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
```

```
zFilename = "sin_sum_by_cos_sum_rand_fourier";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "1 + (sin(x) - 2*sin(2*x-pi) + 3*sin(3*x-pi))*(cos(x) -
2*cos(2*x-pi) + 3*cos(3*x-pi))";
fprintf("%s\n", sEqn);
xData = 0:.05:pi;
xData = xData';
n = length(xData);
yData = 1 + (sin(xData) - 2*sin(2*xData-pi) + 3*sin(3*xData-
pi)).* ...
            (cos(xData) - 2*cos(2*xData-pi) + 3*cos(3*xData-
pi));
fprintf("x=0:.05:pi\n")
order = 4;
[LbS,UbS] = makeLimits(order, 0.5, 1.4);
Lb = [];
Ub = [];
for i=1:order
  Lb = [Lb LbS(i) LbS(i)];
  Ub = [Ub UbS(i) UbS(i)];
end
[bestX,bestFx] =
randomSearch(@quantShammasFourierPoly,Lb,Ub,1000000);

SSE = quantShammasFourierPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Fourier Series factors\n");
bestX
fprintf("Quantum Shammas Fourier Series Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular Fourier fit\n");
sincos_factors = [];
for i=1:order
  sincos_factors =[sincos_factors i i];
end
sincos_factors
[c,r,yFourier] = fourierfit(xData,yData,order);
c = c'
% calculate adjusted value of the coefficient of determination
r = 1 - (1 - r)*(n-1)/(n-2*order-1);
```

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

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + maxPwr;
  end
end
```

The above code generates the following Excel table of results.

QSPfactor1	QSPfactor2	QSPfactor3	QSPfactor4	QSPfactor5
0.612715023	1.241326553	1.968176035	1.77644997	2.933778524
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5

0.758170027	0.553044091	-4.59905903	2.424226434	3.839931992
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
-0.068665115	3.185240212	-1.670822563	5.458038604	0.608478661
r_sqr1	r_sqr2			
0.999541487	0.503588019			

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

QSPfactor6	QSPfactor7	QSPfactor8	
2.600297963	3.614529041	4.300964981	
QSPcoeff6	QSPcoeff7	QSPcoeff8	QSPcoeff9
-0.657414094	0.963447095	-0.18623225	0.018825396
Coeff6	Coeff7	Coeff8	Coeff9
-0.475703429	0.871416365	0.092743087	-0.118935733

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

The adjusted coefficient of determination for the fitted Quantum Shammas Fourier Series is much higher than the one for the regular Fourier series. This condition indicates that the Quantum Shammas Fourier Series performs a better fit for the above example.

Here is the graph for the test function and the two fitted polynomials:

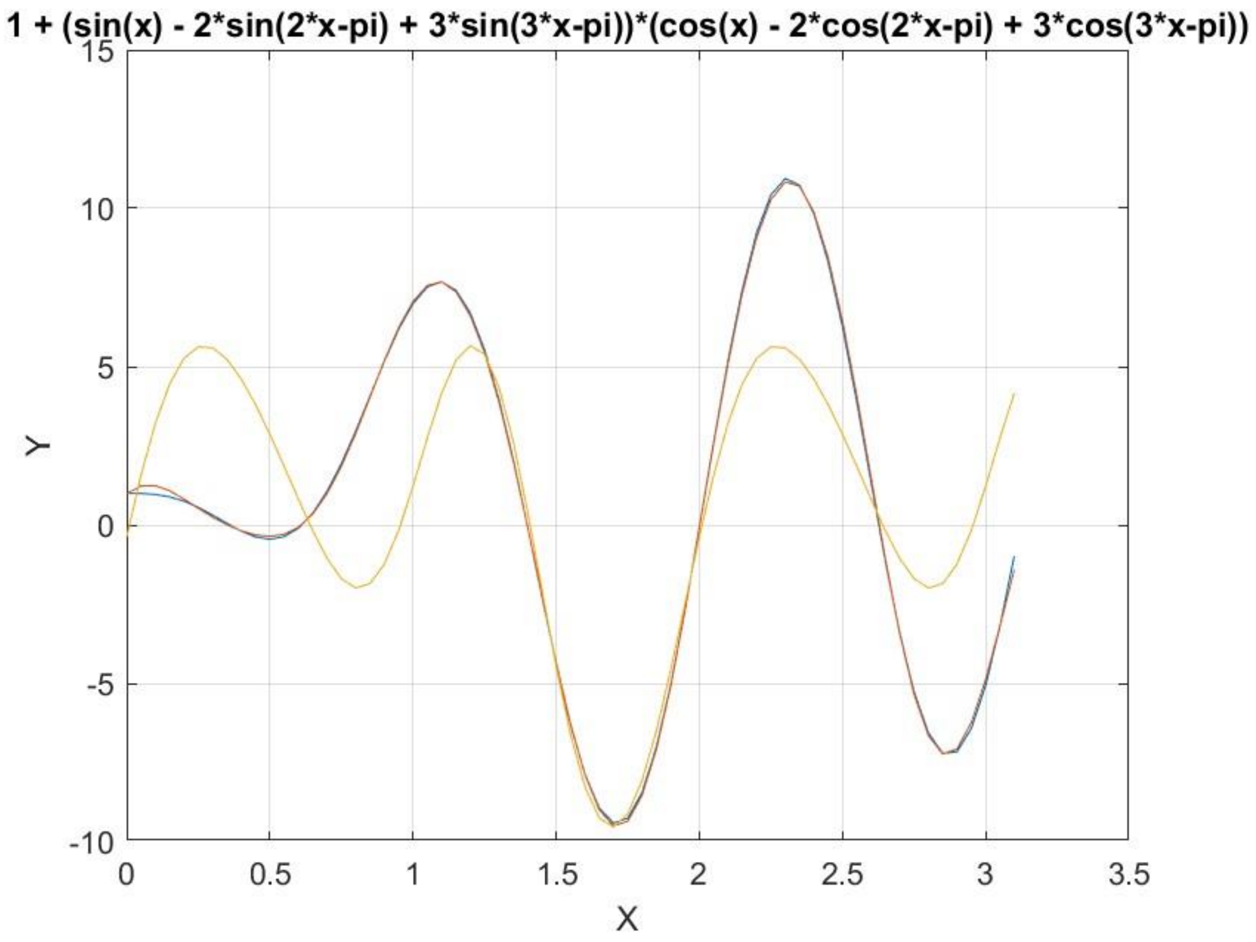


Figure 14. Graph for Fourier Series and test function in file sin_sum_by_cos_sum_rand_fourier.jpg

The above figure shows that the Quantum Shammas Fourier Series fit *function 4* well and much better than the regular Fourier series.

Testing Function 4 Fit with Halton Random Search

The next MATLAB script (found in file found in file testFourierFx4Halton.m) tests fitting *function 4* for x in the range (0, π) and samples at 0.05 steps. The curve fit uses the fourth order Quantum Shammas Fourier Series and a fourth order classical Fourier series.

```
clc
clear
close all
```

```
global xData yData yCalc glbRsqr QSPcoeff
zFilename = "sin_sum_by_cos_sum_halton_rand_fourier";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "1 + (sin(x) - 2*sin(2*x-pi) + 3*sin(3*x-pi))*(cos(x) -
2*cos(2*x-pi) + 3*cos(3*x-pi))";
fprintf("%s\n", sEqn);
xData = 0:.05:pi;
xData = xData';
n = length(xData);
yData = 1 + (sin(xData) - 2*sin(2*xData-pi) + 3*sin(3*xData-
pi)).* ...
            (cos(xData) - 2*cos(2*xData-pi) + 3*cos(3*xData-
pi));
fprintf("x=0:.05:pi\n")
order = 4;
[LbS,UbS] = makeLimits(order, 0.5, 1.4);
Lb = [];
Ub = [];
for i=1:order
  Lb = [Lb LbS(i) LbS(i)];
  Ub = [Ub UbS(i) UbS(i)];
end
[bestX,bestFx] =
haltonRandomSearch(@quantShammasFourierPoly,Lb,Ub,1000000);

SSE = quantShammasFourierPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Fourier Series factors\n");
bestX
fprintf("Quantum Shammas Fourier Series Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular Fourier fit\n");
sincos_factors = [];
for i=1:order
  sincos_factors =[sincos_factors i i];
end
sincos_factors
[c,r,yFourier] = fourierfit(xData,yData,order);
c = c'
% calculate adjusted value of the coefficient of determination
```

```
r = 1 - (1 - r)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,yCalc,xData,yFourier);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + maxPwr;
  end
end
```

The above code generates the following Excel table of results.

QSPfactor1	QSPfactor2	QSPfactor3	QSPfactor4	QSPfactor5
0.714579899	1.242874698	1.902641635	1.811223238	2.344030441
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5

0.879858131	0.611715101	-4.203501191	1.986227448	4.921462952
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
-0.068665115	3.185240212	-1.670822563	5.458038604	0.608478661
r_sqr1	r_sqr2			
0.999867844	0.503588019			

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

QSPfactor6	QSPfactor7	QSPfactor8	
2.665236031	3.811687486	3.349431284	
QSPcoeff6	QSPcoeff7	QSPcoeff8	QSPcoeff9
-1.453049195	-0.664005924	0.06341498	-0.125283937
Coeff6	Coeff7	Coeff8	Coeff9
-0.475703429	0.871416365	0.092743087	-0.118935733

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

The adjusted coefficient of determination for the fitted Quantum Shammas Fourier Series is much higher than the one for the regular Fourier series. This condition indicates that the Quantum Shammas Fourier Series performs a better fit for the above example.

Here is the graph for the test function and the two fitted polynomials:

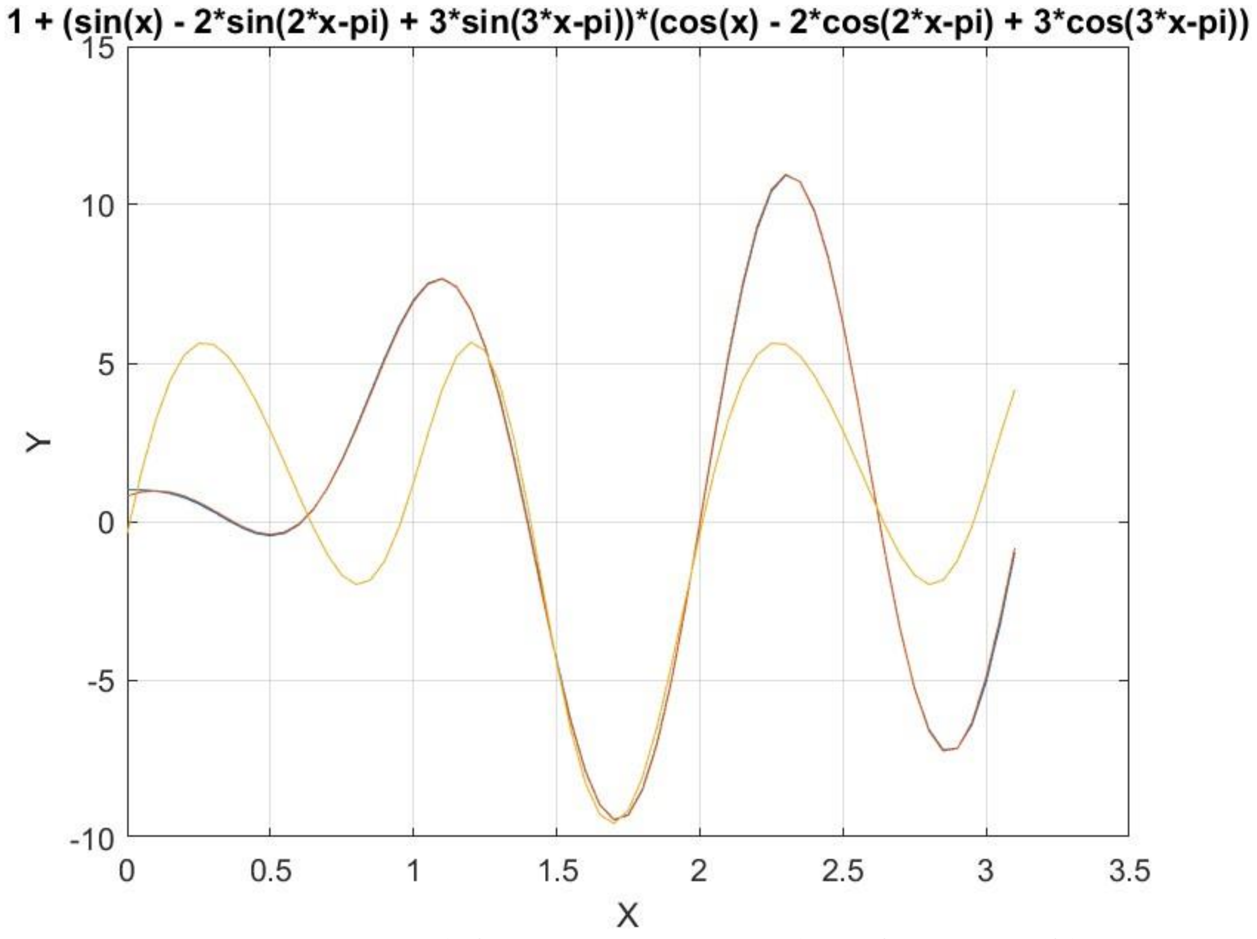


Figure 15. Graph for Fourier Series and test function in file sin_sum_by_cos_sum_halton_rand_fourier.jpg

The above figure shows that the Quantum Shammas Fourier Series fit *function 4* well and much better than the regular Fourier series.

Testing Function 4 Fit with Sobol Random Search

The next MATLAB script (found in file found in file testFourierFx4Sobol.m) tests fitting *function 4* for x in the range (0, π) and samples at 0.05 steps. The curve fit uses the fourth order Quantum Shammas Fourier Series and a fourth order classical Fourier series.

```
clc
clear
close all
```

```
global xData yData yCalc glbRsqr QSPcoeff
zFilename = "sin_sum_by_cos_sum_sobol_rand_fourier";
txtFile = strcat(zFilename, ".txt");
xlFile = strcat(zFilename, ".xlsx");
diary(txtFile)
gFile =  strcat(zFilename, ".jpg");
fprintf("%s\n", datetime(now,'ConvertFrom','datenum'));
format longE
sEqn = "1 + (sin(x) - 2*sin(2*x-pi) + 3*sin(3*x-pi))*(cos(x) -
2*cos(2*x-pi) + 3*cos(3*x-pi))";
fprintf("%s\n", sEqn);
xData = 0:.05:pi;
xData = xData';
n = length(xData);
yData = 1 + (sin(xData) - 2*sin(2*xData-pi) + 3*sin(3*xData-
pi)).* ...
            (cos(xData) - 2*cos(2*xData-pi) + 3*cos(3*xData-
pi));
fprintf("x=0:.05:pi\n")
order = 4;
[LbS,UbS] = makeLimits(order, 0.5, 1.4);
Lb = [];
Ub = [];
for i=1:order
  Lb = [Lb LbS(i) LbS(i)];
  Ub = [Ub UbS(i) UbS(i)];
end
[bestX,bestFx] =
sobolRandomSearch(@quantShammasFourierPoly,Lb,Ub,1000000);

SSE = quantShammasFourierPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Fourier Series factors\n");
bestX
fprintf("Quantum Shammas Fourier Series Coefficients\n");
QSPcoeff = QSPcoeff'
fprintf("\nRegular Fourier fit\n");
sincos_factors = [];
for i=1:order
  sincos_factors =[sincos_factors i i];
end
sincos_factors
[c,r,yFourier] = fourierfit(xData,yData,order);
c = c'
% calculate adjusted value of the coefficient of determination
```

```
r = 1 - (1 - r)*(n-1)/(n-2*order-1);
fprintf("Adjusted Rsqr = %f\n", r);
figure(1)
plot(xData,yData,xData,yCalc,xData,yFourier);
title(sEqn)
xlabel("X")
ylabel("Y");
grid;
ax = gca;
exportgraphics(ax,gFile);

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

format short
diary off

function [Lb,Ub] = makeLimits(order, minPwr, maxPwr)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  Lb(1) = minPwr;
  Ub(1) = maxPwr;
  for i=2:order
    j = i - 1;
    Lb(i) = j + minPwr;
    Ub(i) = j + maxPwr;
  end
end
```

The above code generates the following Excel table of results.

QSPfactor1	QSPfactor2	QSPfactor3	QSPfactor4	QSPfactor5
0.611821196	1.238213226	1.934052134	1.813105337	2.887312705
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5

0.725885554	0.522268089	-4.745956676	2.85073591	3.802824024
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5
-0.068665115	3.185240212	-1.670822563	5.458038604	0.608478661
r_sqr1	r_sqr2			
0.998757463	0.503588019			

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

QSPfactor6	QSPfactor7	QSPfactor8	
2.596566966	4.211567807	4.478532636	
QSPcoeff6	QSPcoeff7	QSPcoeff8	QSPcoeff9
-0.725550696	1.029863072	-0.043296494	0.049028843
Coeff6	Coeff7	Coeff8	Coeff9
-0.475703429	0.871416365	0.092743087	-0.118935733

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

The adjusted coefficient of determination for the fitted Quantum Shammas Fourier Series is much higher than the one for the regular Fourier series. This condition indicates that the Quantum Shammas Fourier Series performs a better fit for the above example.

Here is the graph for the test function and the two fitted polynomials:

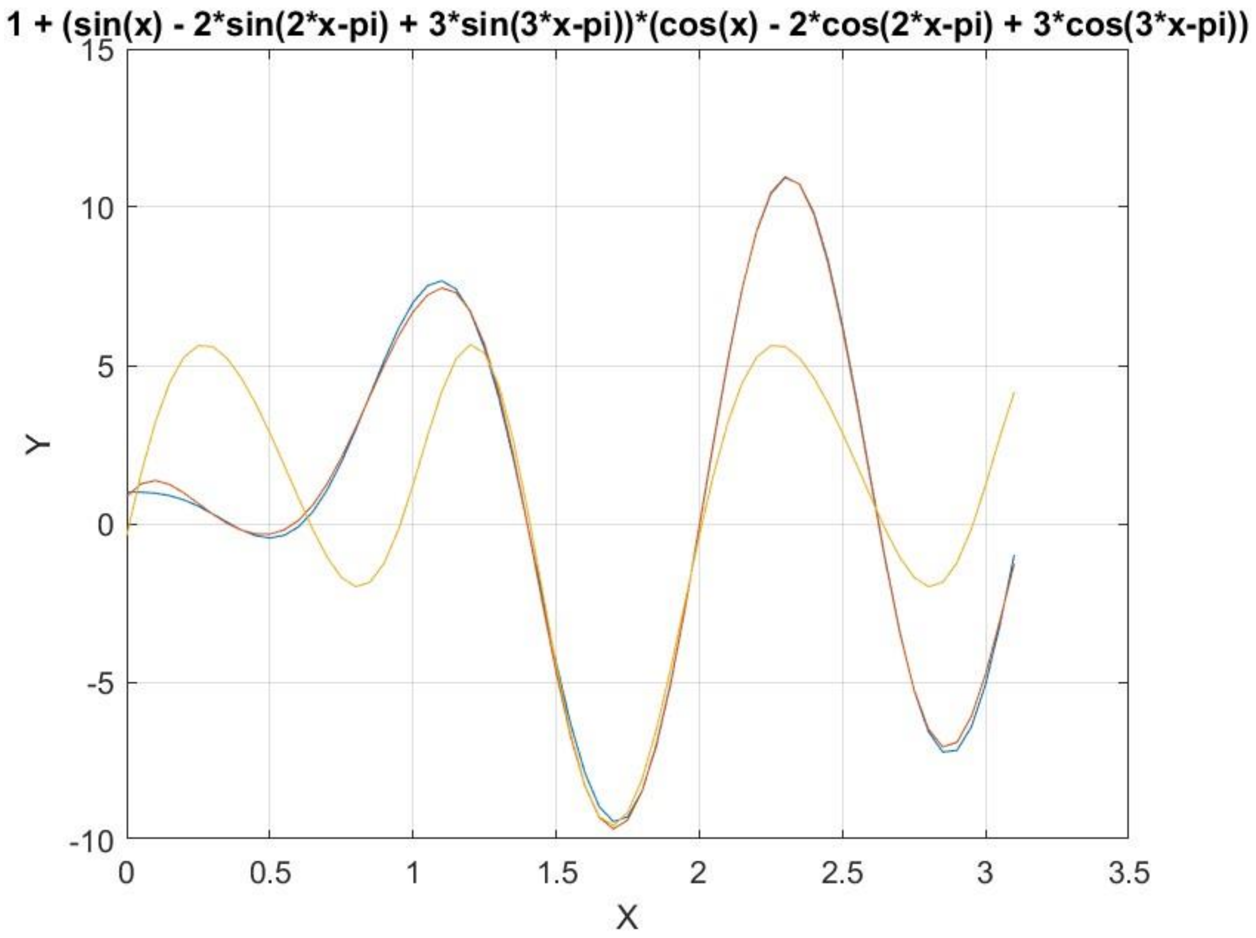


Figure 16. Graph for Fourier Series and test function in file sin_sum_by_cos_sum_sobol_rand_fourier.jpg.

The above figure shows that the Quantum Shammas Fourier Series fit *function 4* well and much better than the regular Fourier series.

Conclusion for Fitting Test Function 4

The above four sections showed that the Quantum Shammas Fourier Series fitted the test function much better than using regular Fourier series.

Conclusion for Fitting All Test Functions

The above 16 sections showed that the Quantum Shammas Fourier Series fitted the the tested functions much better than using regular Fourier series. You can say that the Quantum Shammas Fourier Series was a big success.

☛ Interestingly, in a precursor version of this study, I had removed pi from the trigonometric functions for all the calculations. The results were that the neo-Fourier series did much better, than the regular Fourier series in this study, but still a bit behind the Quantum Shammas Fourier Series. I included the value of pi in order to use the classical definition of Fourier series. I also included the values of pi with the Quantum Shammas Fourier Series so I would be able to compare proverbial apples with apples.

Document History

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