Quantum Shammas Polynomials Part 1E

By
Namir Shammas

Contents

Introduction

This study looks at borrowing aspects of the Shammas Polynomials (which I introduced in 2008 at the HHC 2008 calculator conference in Corvallis, Oregon and also published on my website). and use them with Quantum Shammas Polynomials. The Shammas Polynomials use the following equations:

$$y(x) = a_0 + a_1*x^{p(1)} + a_2*x^{p(2)} + \ldots + a_n*x^{p(n)} \quad \text{for } x>=0 \qquad (1)$$

Where p(i) is a function that takes the index i and yields a real-value power (that is a non-integer in most cases). The condition for the power function is:

$$p(i) > p(j) \text{ for all } j > 0 \text{ and } i > j$$

The Shammas Polynomials do not require optimization as the power function p(i) is supplied outright by the user. The powers of the various terms are fixed by the results of the power function p(i). Contrast this to the Quantum Shammas Polynomials that use random values for the powers. These random powers fall between lower and upper bounds (arrays Lb and Ub). This study looks starting with a power function p(i) and calculating the arrays Lb and Ub before optimizing the powers of the Quantum Shammas Polynomials. We obtain the arrays Lb and Ub given the following input:

- A user/programmer supplied expression for power function p(i).
- The order for the Quantum Shammas Polynomial.
- A gap value, g, between the ranges of powers for each two successive terms.

Armed with the above inputs, we use the following pseudo-code to calculate the arrays Lb and Ub that define the ranges for each power as shown in the following MATLAB function:

```
function [Lb,Ub] = makeLimits(fx, order, gap)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  pwr = [];
  for i=1:order+1
    pwr =  [pwr fx(i)];
  end
  diff = pwr(2) - pwr(1);
  delta = (diff - gap)/2;
  Lb(1) = pwr(1) - delta;
  for i=2:order
    Ub(i-1) = pwr(i-1) + delta;
    diff = pwr(i+1) - pwr(i);
    delta = (diff - gap)/2;
    Lb(i) = pwr(i) - delta;
  end
  Ub(order) = pwr(order) + delta;
end
```

The above function shows how to calculate the arrays of lower and upper bound, Lb and Ub, respectively. Once these values are at hand, the calculations proceed with the *usual optimization steps* (as appearing in Part 1 of the study) for the Quantum Shammas Polynomials. These polynomials have the general equation:

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

Where $Lb(1) <= r_1 <= Ub(1)$, $Lb(2) <= r_2 <= Ub(2)$, …, and $Lb(n) <= r_n < Ub(n)$. The values of arrays Lb and Ub are calculated such as $(Lb(i+1) – Ub(i)) = gap\ g$ for i=1, …, n-1. This condition ensures that no two successive random powers overlap.

The Quantum Shammas Polynomial Function

The Quantum Shammas Polynomial function in MATLAB is:

```
function SSE = quantShammasPoly(pwr)
  global xData yData yCalc glbRsqr QSPcoeff

  n = length(xData);
  order = length(pwr);
```



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

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

The PSO Function

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

```
function [bestX,bestFx] = psox(fx,Lb,Ub,MaxPop,MaxIters,bShow)
% PSOX implements particle swarm optimization.
%
%
% INPUT
% ======
% fx - handle of optimized function.
% Lb - array of low bound values.
% Ub - array of upper bound values.
% MaxPop - maximum population of swarm.
% MaxIters - maximum number of iterations
```



```
% bShow - Boolean flag to request viewing intermediate results.
%
% OUTPUT
% ======
% bestX - array of best solutions.
% bestFx - best optimized function value.
%
% Example
% =======
%
% >>
%
  if nargin < 6, bShow = false; end
  n = length(Lb);
  m = n + 1;
  pop = 1e+99+zeros(MaxPop,m);
  pop2 = pop;
  aPop = zeros(1,n);
  vel = zeros(MaxPop,n);

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

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

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

  % pso loop
  for iter = 1:MaxIters

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



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

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

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

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

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

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

The function has the following input parameters:

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

- The parameter bShow is the Boolean flag to request viewing intermediate results.

The output parameters are:

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

The Random Search Function

The next function performs a random search optimization:

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

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

    delta = 0.15;
```



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

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

The function has the following input parameters:

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

The output parameters are:

- The parameter bestX is the array of best solutions.

- The parameter bestFx is the best optimized function value.

The above function is easy to code and works well with Quantum Shammas Polynomials since the range of each power is relatively small (<1). The above improvement performs two passes for the random search. The first pass uses the lower and upper ranges (in parameters Lb and Ub) that are supplied to the function. The second pass narrows the values of arrays Lb and Ub to be around the best values of X obtained at the end of the first pass.

The Halton Quasi Random Search Function

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

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

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

  % set up halton sequences
  p = haltonset(n,'Skip',1e3,'Leap',1e2);
  p = scramble(p,'RR2');
  rando = net(p,MaxIters);
  for irun=1:2
    for iter = 1:MaxIters
      for i=1:n
        X(i) = Lb(i) + (Ub(i) - Lb(i))*rando(iter,i);
      end
      f = fx(X);
```



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

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

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


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

The Sobol Quasi Random Search Function

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

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

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

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



```
        fprintf("%7i: Fx = %e, X=[", k, bestFx);
        fprintf("%f, ", X)
        fprintf("]\n");
      end
    end

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

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

The above function has the same input and output parameters as the randomSearch() function. The above code shows lines in red that highlight the statements that generate multiple columns of the Sobol sequence and store them in

the matrix rando. The function accesses the elements of matrix rando as pseudo-random numbers are needed.

Testing Quantum Shammas Polynomials

The next sections show examples of calculating the power ranges, given the power functions, and using these ranges to fit Quantum Shammas Polynomials. Will be using the Bessel function $J_0(x)$ for x in the range of (0, 10) to fit various Quantum Shammas Polynomials. The tested power functions are:

- The first power function: $p_4(i) = i/2$.
- The second power function: $p_2(i) = i\text{^}1.5/\ln(1+i)$.
- The third power function: $p_3(i) = 3*i/4$.
- The fourth power function: $p_4(i) = 0.85*\log(1+i)*\sqrt{i}$

Testing Function P1(i) with PSO

The next MATLAB script (found in file testFx1pso.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial, using power function $p_1(i)$, and a sixth order classical polynomial.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_fx1";
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)";
sEqnFx = "sqrt(x)";
fprintf("%s\n", sEqn);
fprintf("%pwrFx = s\n", sEqnFx);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
```



```
[Lb,Ub] = makeLimits(@(x)x/2, order, 0.1);
fprintf("       Lb              Ub\n");
fprintf("-------------------------------\n")
for i=1:order
  fprintf("    %f         %f\n", Lb(i), Ub(i))
end
[bestX,bestFx] = psox(@quantShammasPoly,Lb,Ub,1000,5000,true);

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

QSPpwr = bestX;
Coeff = flip(c);
T = array2table(Lb);
writetable(T,xlFile,"Sheet","Sheet1","Range","A1");
T = array2table(Ub);
writetable(T,xlFile,"Sheet","Sheet1","Range","A4");
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A7");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A10");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A13");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A16");
```



```
format short
diary off

function [Lb,Ub] = makeLimits(fx, order, gap)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  pwr = [];
  for i=1:order+1
    pwr =  [pwr fx(i)];
  end
  diff = pwr(2) - pwr(1);
  delta = (diff - gap)/2;
  Lb(1) = pwr(1) - delta;
  for i=2:order
    Ub(i-1) = pwr(i-1) + delta;
    diff = pwr(i+1) - pwr(i);
    delta = (diff - gap)/2;
    Lb(i) = pwr(i) - delta;
  end
  Ub(order) = pwr(order) + delta;
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

The above code has function makeLimits() that uses a power function to calculate the arrays Lb and Ub. These arrays define the range of power for each term in the Quantum Shammas Polynomial. The above code also copies the console output to a diary text file. It writes the summary results to an Excel table, shown below:

Lb1	Lb2	Lb3	Lb4	Lb5	Lb6	
0.3	0.8	1.3	1.8	2.3	2.8	
Ub1	Ub2	Ub3	Ub4	Ub5	Ub6	
0.7	1.2	1.7	2.2	2.7	3.2	
QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
0.698016618	1.199666312	1.699123668	2.195201566	2.688079611	3.198830488	

QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.109241898	-4.862394194	16.61312244	-21.18027794	11.91228521	-3.021507115	0.269020365
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.979445821	0.996718149					

Table 1. Summary of the results appearing in file besselj_0_x_fx1.xlsx.

The second and fifth rows show the values for calculated arrays Lb and Ub that define the ranges for the powers. The eighth row shows the intercept (below QSPcoeff1) and to its right the coefficients for the various Quantum Shammas Polynomial. The eleventh row shows the intercept and coefficients for the classical polynomial. The cell under r_sqr1 shows the adjusted coefficient of determination for the fitted Quantum Shammas Polynomial. The cell under r_sqr2 shows the adjusted coefficient of determination for the fitted classical polynomial. The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is less than the one for the classical polynomial. This condition indicates that the Quantum Shammas Polynomial performed poorly for the above example.
Here is the graph (from file besselj_0_x_fx1.jpg) for the Bessel function and the two fitted polynomials:

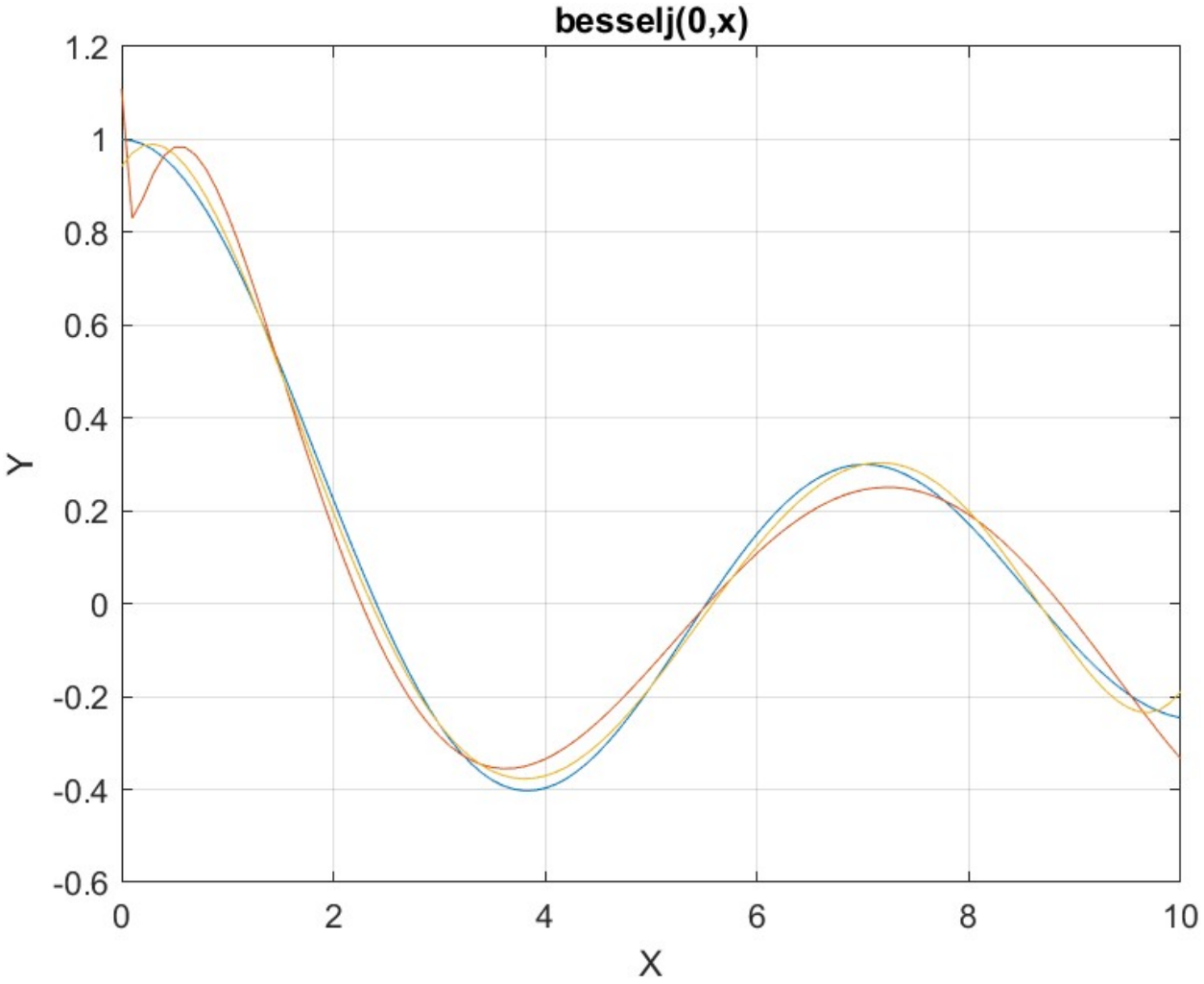


Figure 1. The graph from file besselj_0_x_fx1.jpg.

The above graph shows the deviation of both polynomials from the Bessel function curve. This is not surprising since I chose a wide range to fit, making the test a bit more difficult.

Testing Function P2(i) with PSO

The next MATLAB script (found in file testFx2pso.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial, using power function $p_2(i)$, and a sixth order classical polynomial.

```
clc
clear
close all
```



```
global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_fx2";
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)";
sEqnFx = "x^1.5/ln(1+x)";
fprintf("%s\n", sEqn);
fprintf("pwrFx = %s\n", sEqnFx);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(@(x)x^1.5./log(1+x), order, 0.1);
fprintf("       Lb              Ub\n");
fprintf("-------------------------------\n")
for i=1:order
  fprintf("    %f         %f\n", Lb(i), Ub(i))
end

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

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



```
ylabel("Y");
grid;

ax = gca;
exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
T = array2table(Lb);
writetable(T,xlFile,"Sheet","Sheet1","Range","A1");
T = array2table(Ub);
writetable(T,xlFile,"Sheet","Sheet1","Range","A4");
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A7");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A10");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A13");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A16");

format short
diary off

function [Lb,Ub] = makeLimits(fx, order, gap)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  pwr = [];
  for i=1:order+1
    pwr =  [pwr fx(i)];
  end
  diff = pwr(2) - pwr(1);
  delta = (diff - gap)/2;
  Lb(1) = pwr(1) - delta;
  for i=2:order
    Ub(i-1) = pwr(i-1) + delta;
    diff = pwr(i+1) - pwr(i);
    delta = (diff - gap)/2;
    Lb(i) = pwr(i) - delta;
  end
  Ub(order) = pwr(order) + delta;
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
```



```
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

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

Lb1	Lb2	Lb3	Lb4	Lb5	Lb6	
0.926769902	2.037702145	3.187007761	4.386085965	5.633433594	6.925917454	
Ub1	Ub2	Ub3	Ub4	Ub5	Ub6	
1.95862018	3.111388492	4.309455571	5.555272988	6.846299405	8.179547167	
QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
1.619929546	3.108716548	4.291321777	4.986910078	5.635018494	6.928204459	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.002975473	-0.17413063	-0.111779707	0.075013519	-0.030273945	0.004324314	-2.90366E-05
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.999979347	0.996718149					

Table 2. Summary of the results appearing in file besselj_0_x_fx2.xlsx.

The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is higher than the one for the classical polynomial. This condition indicates that the Quantum Shammas Polynomial performed well when using the power function $p_2(i)$.

Here is the graph (from file besselj_0_x_fx2.jpg) for the Bessel function and the two fitted polynomials:

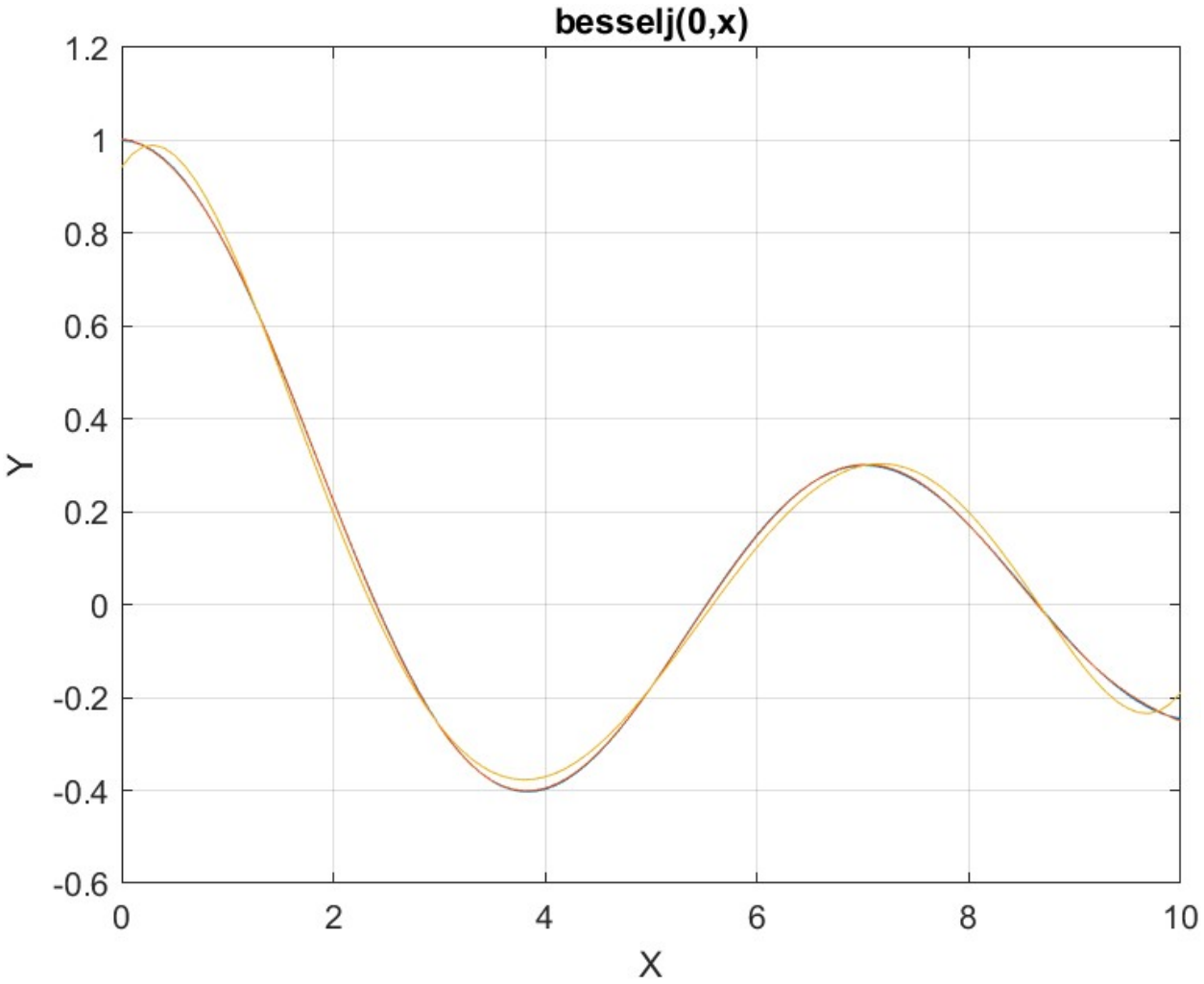


Figure 2. The graph from file besselj_0_x_fx2.jpg.

The above graph shows the deviation of the regular polynomial from the Bessel function curve. The Quantum Shammas Polynomial fits the Bessel function well.

Testing Function P3(i) with PSO

The next MATLAB script (found in file testFx3pso.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial, using power function $p_3(i)$, and a sixth order classical polynomial.

```
clc
clear
```



```
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_fx3";
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)";
sEqnFx = "x^0.75";
fprintf("%s\n", sEqn);
fprintf("%pwrFx = s\n", sEqnFx);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(@(x)3*x/4, order, 0.1);
fprintf("       Lb              Ub\n");
fprintf("--------------------------------\n")
for i=1:order
  fprintf("    %f         %f\n", Lb(i), Ub(i))
end
[bestX,bestFx] = psox(@quantShammasPoly,Lb,Ub,1000,5000,true);

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



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

QSPpwr = bestX;
Coeff = flip(c);
T = array2table(Lb);
writetable(T,xlFile,"Sheet","Sheet1","Range","A1");
T = array2table(Ub);
writetable(T,xlFile,"Sheet","Sheet1","Range","A4");
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A7");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A10");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A13");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A16");
format short
diary off

function [Lb,Ub] = makeLimits(fx, order, gap)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  pwr = [];
  for i=1:order+1
    pwr =  [pwr fx(i)];
  end
  diff = pwr(2) - pwr(1);
  delta = (diff - gap)/2;
  Lb(1) = pwr(1) - delta;
  for i=2:order
    Ub(i-1) = pwr(i-1) + delta;
    diff = pwr(i+1) - pwr(i);
    delta = (diff - gap)/2;
    Lb(i) = pwr(i) - delta;
  end
  Ub(order) = pwr(order) + delta;
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
```



```
  r = 1 - SSE / SStot;
end
```

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

Lb1	Lb2	Lb3	Lb4	Lb5	Lb6	
0.425	1.175	1.925	2.675	3.425	4.175	
Ub1	Ub2	Ub3	Ub4	Ub5	Ub6	
1.075	1.825	2.575	3.325	4.075	4.825	
QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
1.074040755	1.824273931	2.54137332	2.797189803	3.428140406	4.177697583	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.022923195	-0.71348962	2.567752492	-7.290567501	5.784581364	-0.617783248	0.026649615
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.999979347	0.996718149					

Table 3. Summary of the results appearing in file besselj_0_x_fx3.xlsx.

The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is higher than the one for the classical polynomial. This condition indicates that the Quantum Shammas Polynomial performed well when using the power function $p_3(i)$.

Here is the graph (from file besselj_0_x_fx3.jpg) for the Bessel function and the two fitted polynomials:

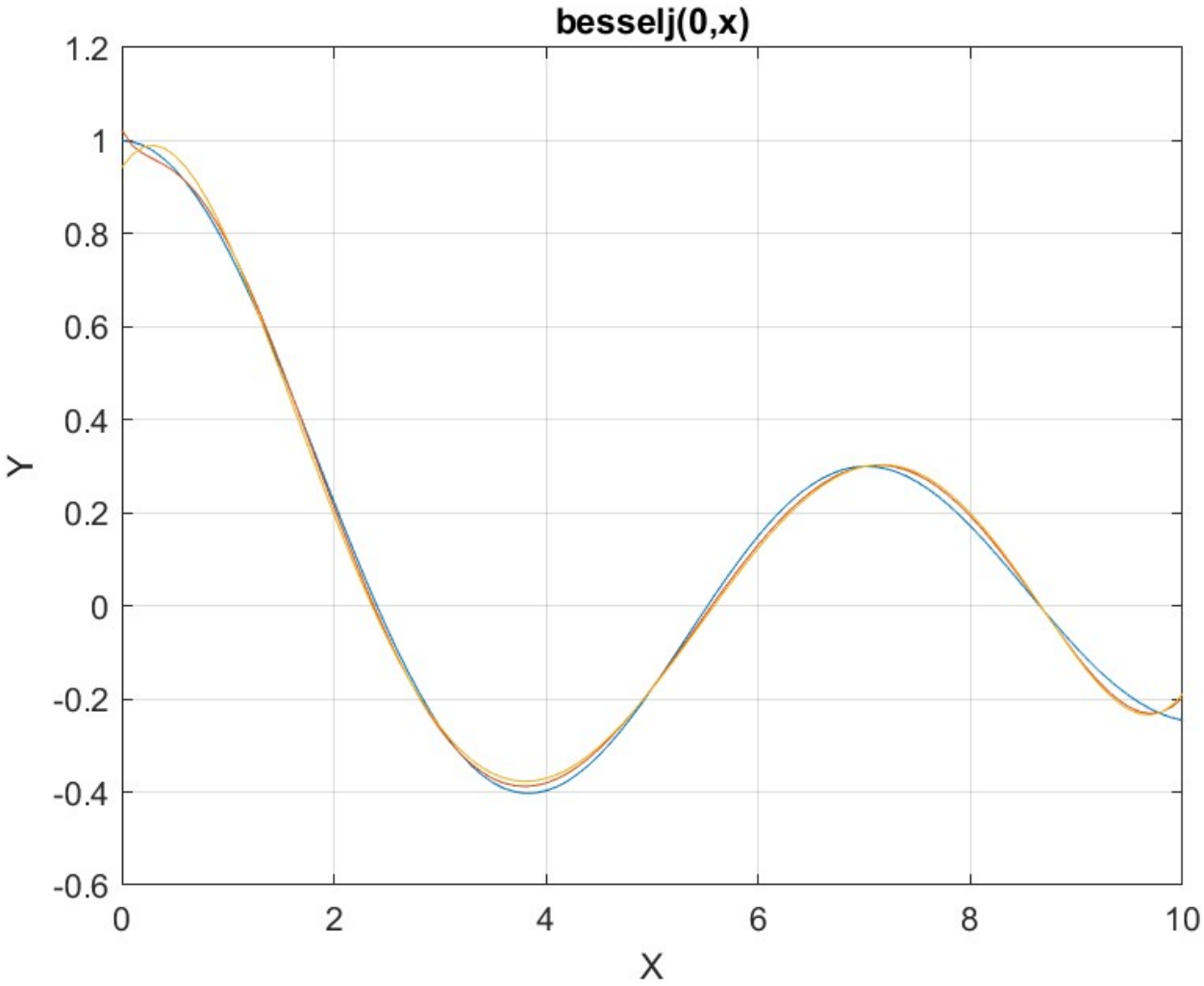


Figure 3. The graph from file besselj_0_x_fx3.jpg.

The above graph shows the deviation of the regular polynomial from the Bessel function curve. The Quantum Shammas Polynomial fits the Bessel function well.

Testing Function P4(i) with PSO

The next MATLAB script (found in file testFx4pso.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial, using power function $p_4(i)$, and a sixth order classical polynomial.

```
clc
clear
close all
```



```
global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_fx4";
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)";
sEqnFx = "0.85*log(1+x)*sqrt(x)";
fprintf("%s\n", sEqn);
fprintf("%pwrFx = s\n", sEqnFx);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(@(x)0.85*log(1+x).*sqrt(x), order, 0.1);
fprintf("       Lb              Ub\n");
fprintf("-------------------------------\n")
for i=1:order
  fprintf("    %f         %f\n", Lb(i), Ub(i))
end
[bestX,bestFx] = psox(@quantShammasPoly,Lb,Ub,1000,5000,true);

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



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

QSPpwr = bestX;
Coeff = flip(c);
T = array2table(Lb);
writetable(T,xlFile,"Sheet","Sheet1","Range","A1");
T = array2table(Ub);
writetable(T,xlFile,"Sheet","Sheet1","Range","A4");
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A7");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A10");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A13");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A16");
format short
diary off

function [Lb,Ub] = makeLimits(fx, order, gap)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  pwr = [];
  for i=1:order+1
    pwr =  [pwr fx(i)];
  end
  diff = pwr(2) - pwr(1);
  delta = (diff - gap)/2;
  Lb(1) = pwr(1) - delta;
  for i=2:order
    Ub(i-1) = pwr(i-1) + delta;
    diff = pwr(i+1) - pwr(i);
    delta = (diff - gap)/2;
    Lb(i) = pwr(i) - delta;
  end
  Ub(order) = pwr(order) + delta;
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
```



```
end
```

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

Lb1	Lb2	Lb3	Lb4	Lb5	Lb6	
0.273451886	1.010451094	1.743421416	2.451305888	3.132525411	3.789054658	
Ub1	Ub2	Ub3	Ub4	Ub5	Ub6	
0.904898321	1.630791983	2.338503439	3.020783014	3.678517742	4.313973158	
QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
0.90375571	1.630713889	2.33793201	3.009325649	3.625459091	4.136917727	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.026862068	-0.5971088	1.761847105	-2.446209207	1.355304668	-0.363075479	0.044423852
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.998388416	0.996718149					

Table 4. Summary of the results appearing in file besselj_0_x_fx4.xlsx.

The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is slightly higher than the one for the classical polynomial. This condition indicates that the Quantum Shammas Polynomial performed slightly better when using the power function $p_4(i)$.

Here is the graph (from file besselj_0_x_fx4.jpg) for the Bessel function and the two fitted polynomials:

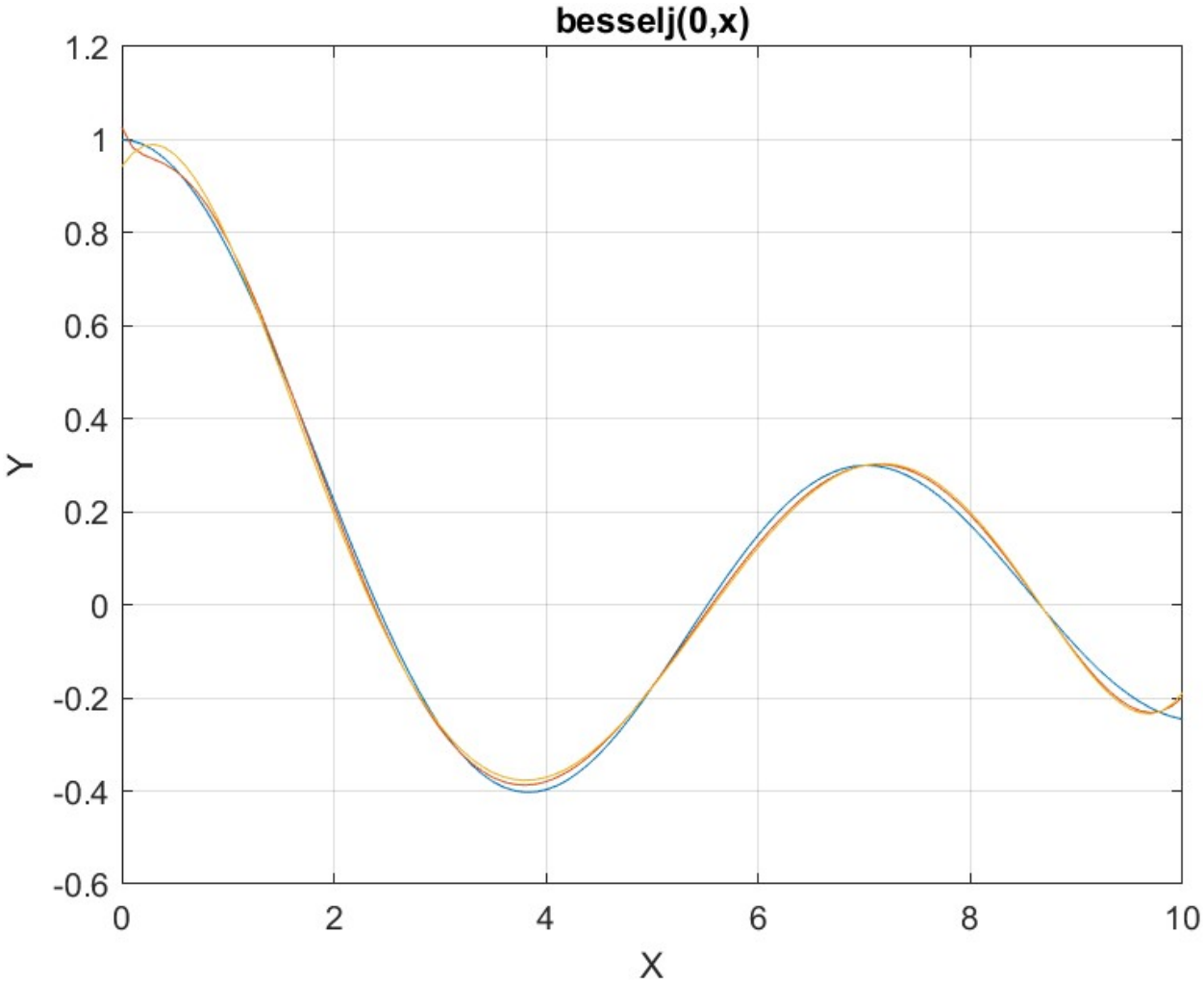


Figure 4. The graph from file besselj_0_x_fx4.jpg.

The above graph shows some deviation of both polynomials from the Bessel function curve.

Conclusion for Using the PSO Method

The PSO method did well with the power functions $p_2(i)$, $p_3(i)$, and $p_4(i)$.

Testing Function P1(i) with Random Search Optimization

The next MATLAB script (found in file testBessel1Random.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial, using power function $p_1(i)$, and a sixth order classical polynomial.


```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_random_fx1";
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);
sEqnFx = "sqrt(x)";
fprintf("%s\n", sEqn);
fprintf("pwrFx = %s\n", sEqnFx);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(@(x)x/2, order, 0.1);
fprintf("       Lb              Ub\n");
fprintf("-------------------------------\n")
for i=1:order
  fprintf("    %f         %f\n", Lb(i), Ub(i))
end
[bestX,bestFx] = randomSearch(@quantShammasPoly,Lb,Ub,1000000);

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



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

exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
T = array2table(Lb);
writetable(T,xlFile,"Sheet","Sheet1","Range","A1");
T = array2table(Ub);
writetable(T,xlFile,"Sheet","Sheet1","Range","A4");
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A7");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A10");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A13");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A16");

format short
diary off

function [Lb,Ub] = makeLimits(fx, order, gap)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  pwr = [];
  for i=1:order+1
    pwr =  [pwr fx(i)];
  end
  diff = pwr(2) - pwr(1);
  delta = (diff - gap)/2;
  Lb(1) = pwr(1) - delta;
  for i=2:order
    Ub(i-1) = pwr(i-1) + delta;
    diff = pwr(i+1) - pwr(i);
    delta = (diff - gap)/2;
    Lb(i) = pwr(i) - delta;
  end
  Ub(order) = pwr(order) + delta;
end
```



```
function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

The above code has uses the randomSearch() function to perform random search optimization. The above code also copies the console output to a diary text file. It writes the summary results to an Excel table, shown below:

Lb1	Lb2	Lb3	Lb4	Lb5	Lb6	
0.3	0.8	1.3	1.8	2.3	2.8	
Ub1	Ub2	Ub3	Ub4	Ub5	Ub6	
0.7	1.2	1.7	2.2	2.7	3.2	
QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
0.758280325	1.287595618	1.83022503	2.380273598	2.903979658	3.471867592	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.107576117	-3.762630366	11.76653629	-14.02893321	7.438607835	-1.833634466	0.143513514
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.988356148	0.996718149					

Table 5. Summary of the results appearing in file besselj_0_x_random_fx1.xlsx.

The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is less than the one for the classical polynomial. This condition indicates that the Quantum Shammas Polynomial performed with the power function $p_1(i)$.

Here is the graph (from file besselj_0_x_random_fx1.jpg) for the Bessel function and the two fitted polynomials:

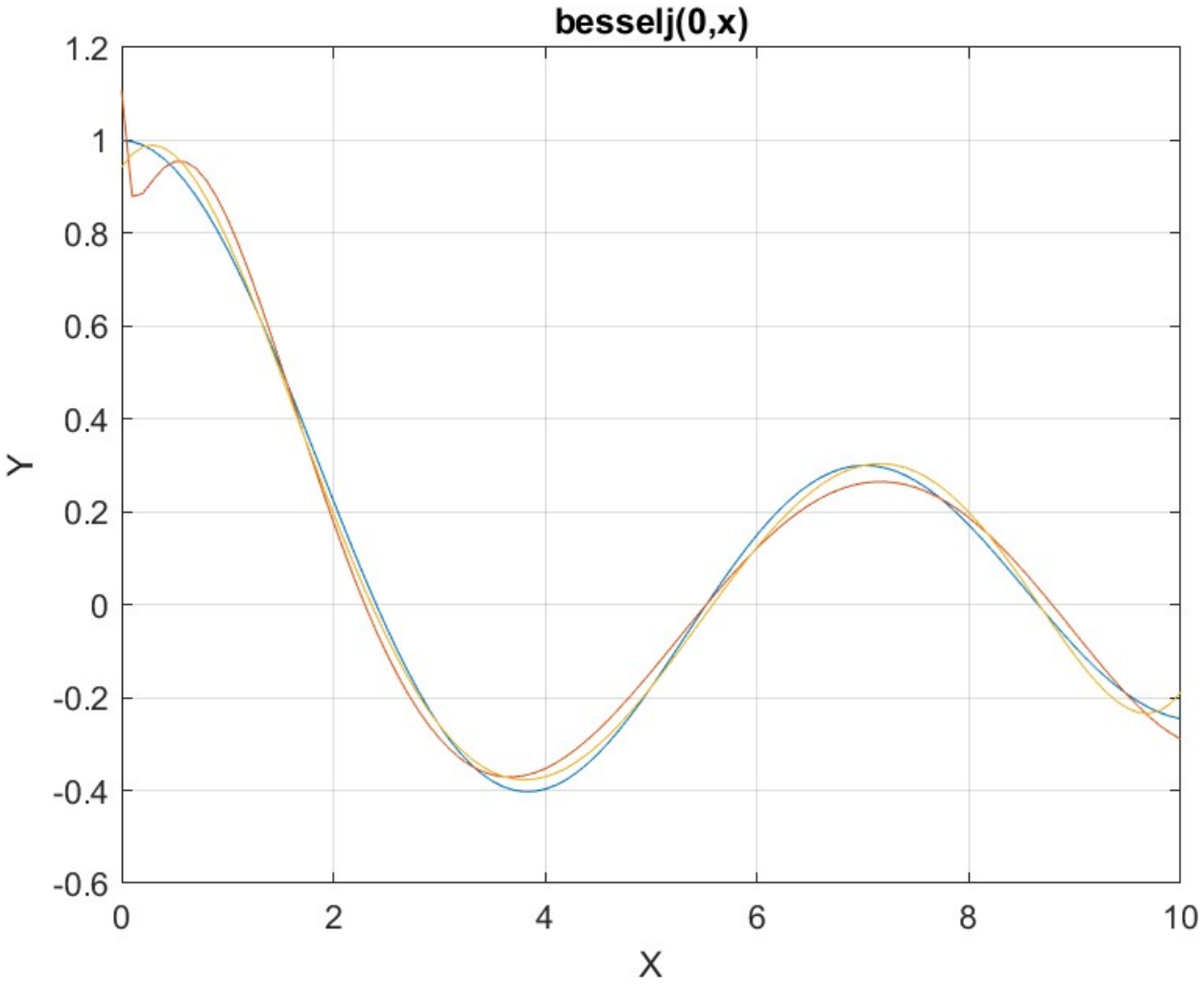


Figure 5. The graph from file besselj_0_x_random_fx1.jpg.

The above graph shows the deviation of both polynomials from the Bessel function curve. This is not surprising since I chose a wide range to fit, making the test a bit more difficult.

Testing Function P2(i) with Random Search Optimization

The next MATLAB script (found in file testBessel2Random.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial, using power function $p_2(i)$, and a sixth order classical polynomial.

```
clc
```



```
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_random_fx2";
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);
sEqnFx = "x^1.5/ln(1+x)";
fprintf("%s\n", sEqn);
fprintf("pwrFx = %s\n", sEqnFx);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(@(x)x^1.5./log(1+x), order, 0.1);
fprintf("       Lb              Ub\n");
fprintf("-------------------------------\n")
for i=1:order
  fprintf("    %f         %f\n", Lb(i), Ub(i))
end
[bestX,bestFx] = randomSearch(@quantShammasPoly,Lb,Ub,1000000);

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



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

exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
T = array2table(Lb);
writetable(T,xlFile,"Sheet","Sheet1","Range","A1");
T = array2table(Ub);
writetable(T,xlFile,"Sheet","Sheet1","Range","A4");
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A7");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A10");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A13");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A16");

format short
diary off

function [Lb,Ub] = makeLimits(fx, order, gap)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  pwr = [];
  for i=1:order+1
    pwr =  [pwr fx(i)];
  end
  diff = pwr(2) - pwr(1);
  delta = (diff - gap)/2;
  Lb(1) = pwr(1) - delta;
  for i=2:order
    Ub(i-1) = pwr(i-1) + delta;
    diff = pwr(i+1) - pwr(i);
    delta = (diff - gap)/2;
    Lb(i) = pwr(i) - delta;
  end
  Ub(order) = pwr(order) + delta;
end

function r = rsqr(y,ycalc)
```



```
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

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

Lb1	Lb2	Lb3	Lb4	Lb5	Lb6	
0.926769902	2.037702145	3.187007761	4.386085965	5.633433594	6.925917454	
Ub1	Ub2	Ub3	Ub4	Ub5	Ub6	
1.95862018	3.111388492	4.309455571	5.555272988	6.846299405	8.179547167	
QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
1.709262254	3.321352586	4.577182925	5.273854583	5.469682947	6.247861318	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.002737031	-0.197645356	-0.080404438	0.080621419	-0.093196614	0.054460455	-0.000802411
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.999982913	0.996718149					

Table 6. Summary of the results appearing in file besselj_0_x_random_fx2.xlsx.

The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is higher than the one for the classical polynomial. This condition indicates that the Quantum Shammas Polynomial performed well when using the power function $p_2(i)$.

Here is the graph (from file besselj_0_x_random_fx2.jpg) for the Bessel function and the two fitted polynomials:

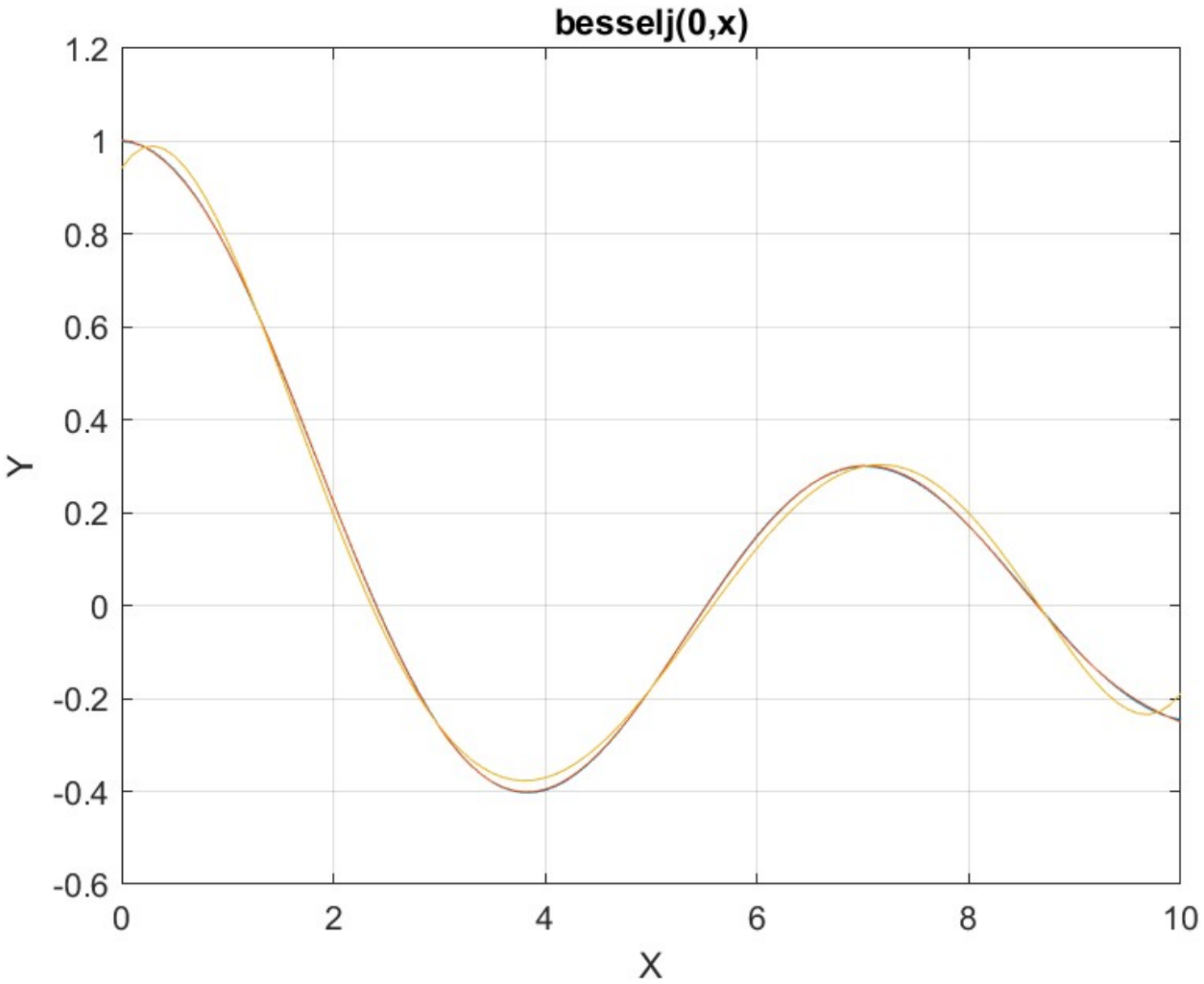


Figure 6. The graph from file besselj_0_x_random_fx2.jpg.

The above graph shows the deviation of the regular polynomial from the Bessel function curve. The Quantum Shammas Polynomial fits the Bessel function well.

Testing Function P3(i) with Random Search Optimization

The next MATLAB script (found in file testBessel3Random.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial, using power function $p_3(i)$, and a sixth order classical polynomial.

```
clc
clear
close all
```



```
global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_random_fx4";
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);
sEqnFx = "0.85*log(1+x)*sqrt(x)";
fprintf("%s\n", sEqn);
fprintf("pwrFx = %s\n", sEqnFx);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(@(x)0.85*log(1+x).*sqrt(x), order, 0.1);
fprintf("       Lb              Ub\n");
fprintf("--------------------------------\n")
for i=1:order
  fprintf("    %f         %f\n", Lb(i), Ub(i))
end
[bestX,bestFx] = randomSearch(@quantShammasPoly,Lb,Ub,1000000);

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



```
ylabel("Y");
grid;
ax = gca;

exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
T = array2table(Lb);
writetable(T,xlFile,"Sheet","Sheet1","Range","A1");
T = array2table(Ub);
writetable(T,xlFile,"Sheet","Sheet1","Range","A4");
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A7");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A10");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A13");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A16");

format short
diary off

function [Lb,Ub] = makeLimits(fx, order, gap)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  pwr = [];
  for i=1:order+1
    pwr =  [pwr fx(i)];
  end
  diff = pwr(2) - pwr(1);
  delta = (diff - gap)/2;
  Lb(1) = pwr(1) - delta;
  for i=2:order
    Ub(i-1) = pwr(i-1) + delta;
    diff = pwr(i+1) - pwr(i);
    delta = (diff - gap)/2;
    Lb(i) = pwr(i) - delta;
  end
  Ub(order) = pwr(order) + delta;
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
```



```
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

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

Lb1	Lb2	Lb3	Lb4	Lb5	Lb6	
0.425	1.175	1.925	2.675	3.425	4.175	
Ub1	Ub2	Ub3	Ub4	Ub5	Ub6	
1.075	1.825	2.575	3.325	4.075	4.825	
QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
1.154166638	1.964255454	2.792844329	2.865939761	3.203699387	3.793688485	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.024585066	-0.791763589	3.261109138	-50.57980397	53.68193354	-6.027235491	0.210899572
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.998616588	0.996718149					

Table 7. Summary of the results appearing in file besselj_0_x_random_fx2.xlsx.

The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is slightly higher than the one for the classical polynomial. This condition indicates that the Quantum Shammas Polynomial performed well when using the power function $p_3(i)$.

Here is the graph (from file besselj_0_x_random_fx3.jpg) for the Bessel function and the two fitted polynomials:

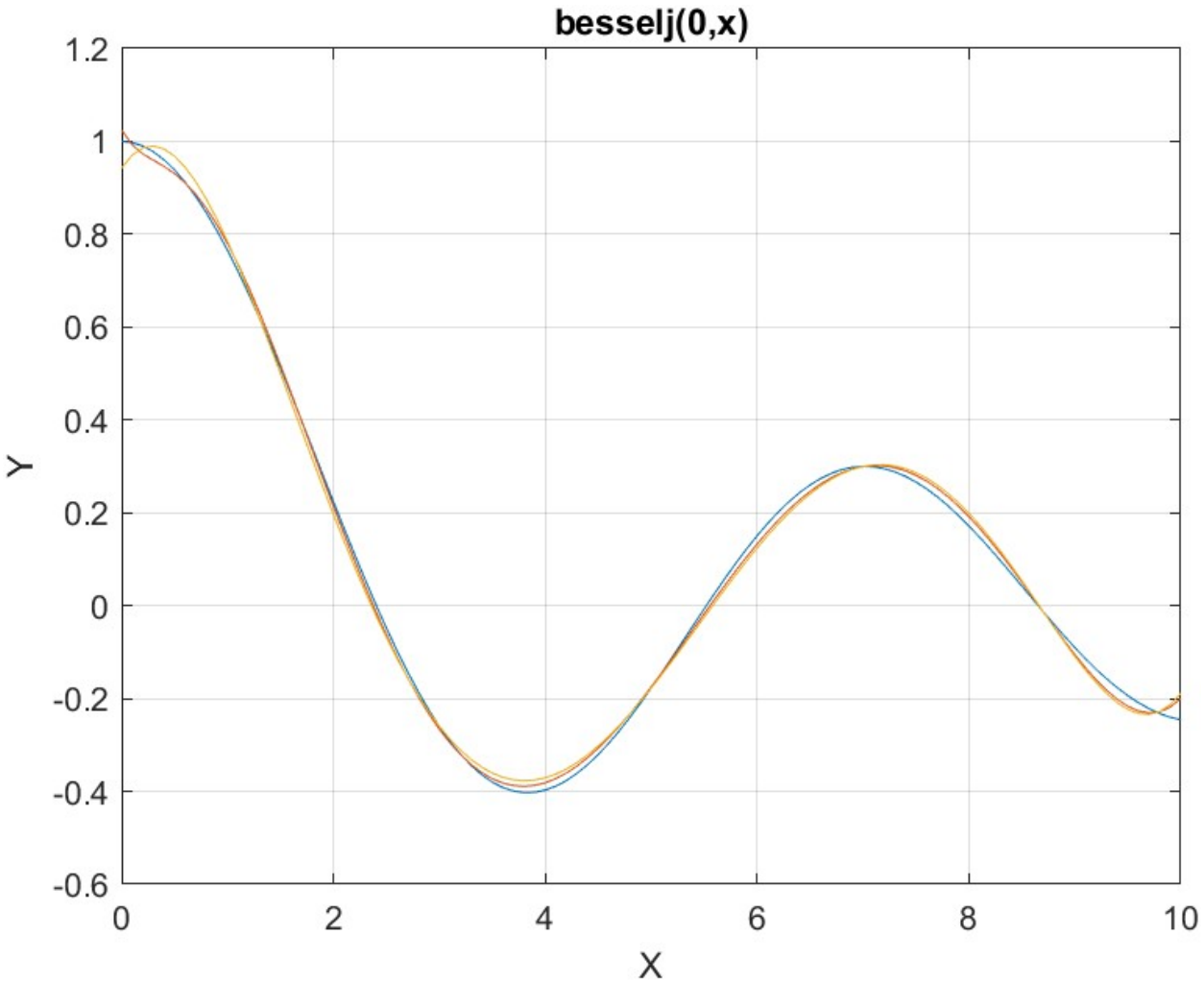


Figure 7. The graph from file besselj_0_x_random_fx3.jpg.

The above graph shows some deviation of both polynomials from the Bessel function curve.

Testing Function P4(i) with Random Search Optimization

The next MATLAB script (found in file testBessel4Random.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial, using power function $p_4(i)$, and a sixth order classical polynomial.

```
clc
clear
```



```
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_random_fx4";
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);
sEqnFx = "0.85*log(1+x)*sqrt(x)";
fprintf("%s\n", sEqn);
fprintf("pwrFx = %s\n", sEqnFx);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(@(x)0.85*log(1+x).*sqrt(x), order, 0.1);
fprintf("       Lb              Ub\n");
fprintf("--------------------------------\n")
for i=1:order
  fprintf("    %f         %f\n", Lb(i), Ub(i))
end
[bestX,bestFx] = randomSearch(@quantShammasPoly,Lb,Ub,1000000);

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



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

exportgraphics(ax,gFile);

QSPpwr = bestX;
Coeff = flip(c);
T = array2table(Lb);
writetable(T,xlFile,"Sheet","Sheet1","Range","A1");
T = array2table(Ub);
writetable(T,xlFile,"Sheet","Sheet1","Range","A4");
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A7");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A10");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A13");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A16");

format short
diary off

function [Lb,Ub] = makeLimits(fx, order, gap)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  pwr = [];
  for i=1:order+1
    pwr =  [pwr fx(i)];
  end
  diff = pwr(2) - pwr(1);
  delta = (diff - gap)/2;
  Lb(1) = pwr(1) - delta;
  for i=2:order
    Ub(i-1) = pwr(i-1) + delta;
    diff = pwr(i+1) - pwr(i);
    delta = (diff - gap)/2;
    Lb(i) = pwr(i) - delta;
  end
  Ub(order) = pwr(order) + delta;
end

function r = rsqr(y,ycalc)
  n = length(y);
```



```
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

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

Lb1	Lb2	Lb3	Lb4	Lb5	Lb6	
0.273451886	1.010451094	1.743421416	2.451305888	3.132525411	3.789054658	
Ub1	Ub2	Ub3	Ub4	Ub5	Ub6	
0.904898321	1.630791983	2.338503439	3.020783014	3.678517742	4.313973158	
QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
0.963934364	1.757150102	2.526775725	3.252775459	3.438842159	3.741536528	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.0255055	-0.557604031	1.724354982	-3.025836889	5.523530446	-4.550509085	0.641437244
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.998519558	0.996718149					

Table 8. Summary of the results appearing in file besselj_0_x_random_fx2.xlsx.

The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is slightly higher than the one for the classical polynomial. This condition indicates that the Quantum Shammas Polynomial performed well when using the power function $p_4(i)$.

Here is the graph (from file besselj_0_x_random_fx4.jpg) for the Bessel function and the two fitted polynomials:

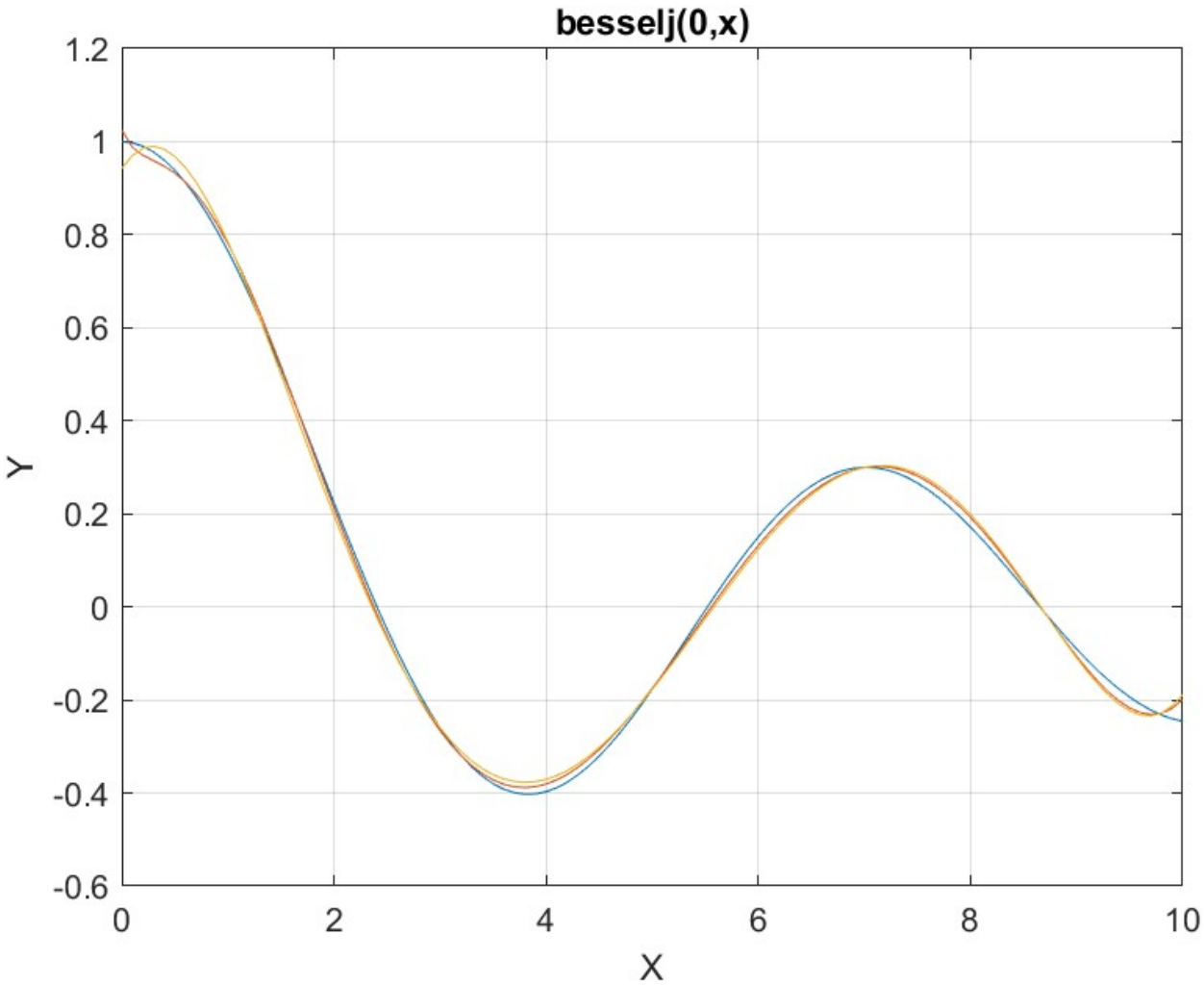


Figure 8. The graph from file besselj_0_x_random_fx4.jpg.

The above graph shows some deviation of both polynomials from the Bessel function curve.

Conclusion for Using the Random Search Optimization Method

The random search optimization method did well with the power functions $p_2(i)$, $p_3(i)$, and $p_4(i)$. This is the same pattern as with the PSO method. My conclusion, so far, is that power functions $p_2(i)$, $p_3(i)$, and $p_4(i)$ yield better least-square fits.

Testing Function P1(i) with the Halton Quasi-Random Search

The next MATLAB script (found in file testBessel1Halton.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth

order Quantum Shammas Polynomial, using power function $p_1(i)$, and a sixth order classical polynomial.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_halton_random_fx1";
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);
sEqnFx = "sqrt(x)";
fprintf("%s\n", sEqn);
fprintf("pwrFx = %s\n", sEqnFx);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(@(x)x/2, order, 0.1);
fprintf("       Lb              Ub\n");
fprintf("-------------------------------\n")
for i=1:order
  fprintf("    %f         %f\n", Lb(i), Ub(i))
end

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

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



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

QSPpwr = bestX;
Coeff = flip(c);
T = array2table(Lb);
writetable(T,xlFile,"Sheet","Sheet1","Range","A1");
T = array2table(Ub);
writetable(T,xlFile,"Sheet","Sheet1","Range","A4");
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A7");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A10");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A13");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A16");

format short
diary off

function [Lb,Ub] = makeLimits(fx, order, gap)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  pwr = [];
  for i=1:order+1
    pwr =  [pwr fx(i)];
  end
  diff = pwr(2) - pwr(1);
  delta = (diff - gap)/2;
  Lb(1) = pwr(1) - delta;
  for i=2:order
    Ub(i-1) = pwr(i-1) + delta;
    diff = pwr(i+1) - pwr(i);
    delta = (diff - gap)/2;
    Lb(i) = pwr(i) - delta;
```



```
  end
  Ub(order) = pwr(order) + delta;
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

The above code has uses the haltonRandomSearch() function to perform random search optimization. The above code also copies the console output to a diary text file. It writes the summary results to an Excel table, shown below:

Lb1	Lb2	Lb3	Lb4	Lb5	Lb6	
0.3	0.8	1.3	1.8	2.3	2.8	
Ub1	Ub2	Ub3	Ub4	Ub5	Ub6	
0.7	1.2	1.7	2.2	2.7	3.2	
QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
0.757759473	1.272296951	1.841576055	2.368689456	2.958383904	3.43188847	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.10723258	-3.807090719	11.28108029	-13.53689531	7.210579026	-1.623215358	0.199318055
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.988359066	0.996718149					

Table 9. Summary of the results appearing in file besselj_0_x_halton_random_fx1.xlsx.

The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is less than the one for the classical polynomial. This condition indicates that the Quantum Shammas Polynomial performed with the power function $p_1(i)$.

Here is the graph (from file besselj_0_x_halton_random_fx1.jpg) for the Bessel function and the two fitted polynomials:

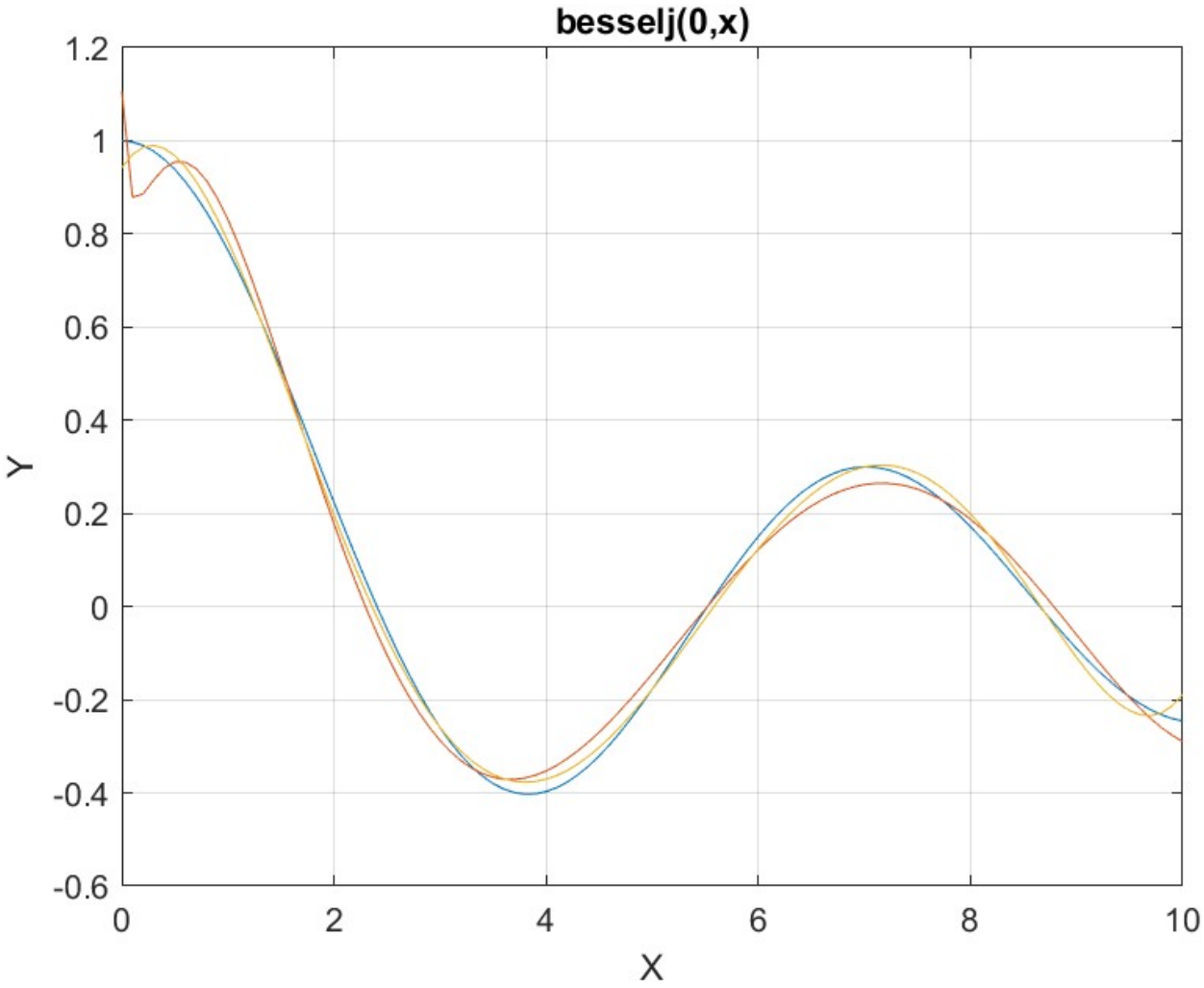


Figure 9. The graph from file besselj_0_x_halton_random_fx1.jpg.

The above graph shows the deviation of both polynomials from the Bessel function curve. This is not surprising since I chose a wide range to fit, making the test a bit more difficult.

Testing Function P2(i) with the Halton Quasi-Random Search

The next MATLAB script (found in file testBessel2Halton.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial, using power function $p_2(i)$, and a sixth order classical polynomial.


```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_halton_random_fx4";
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);
sEqnFx = "0.85*log(1+x)*sqrt(x)";
fprintf("%s\n", sEqn);
fprintf("pwrFx = %s\n", sEqnFx);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(@(x)0.85*log(1+x).*sqrt(x), order, 0.1);
fprintf("       Lb              Ub\n");
fprintf("------------------------------\n")
for i=1:order
  fprintf("    %f         %f\n", Lb(i), Ub(i))
end

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

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



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

QSPpwr = bestX;
Coeff = flip(c);
T = array2table(Lb);
writetable(T,xlFile,"Sheet","Sheet1","Range","A1");
T = array2table(Ub);
writetable(T,xlFile,"Sheet","Sheet1","Range","A4");
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A7");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A10");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A13");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A16");

format short
diary off

function [Lb,Ub] = makeLimits(fx, order, gap)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  pwr = [];
  for i=1:order+1
    pwr =  [pwr fx(i)];
  end
  diff = pwr(2) - pwr(1);
  delta = (diff - gap)/2;
  Lb(1) = pwr(1) - delta;
  for i=2:order
    Ub(i-1) = pwr(i-1) + delta;
    diff = pwr(i+1) - pwr(i);
    delta = (diff - gap)/2;
    Lb(i) = pwr(i) - delta;
  end
  Ub(order) = pwr(order) + delta;
```



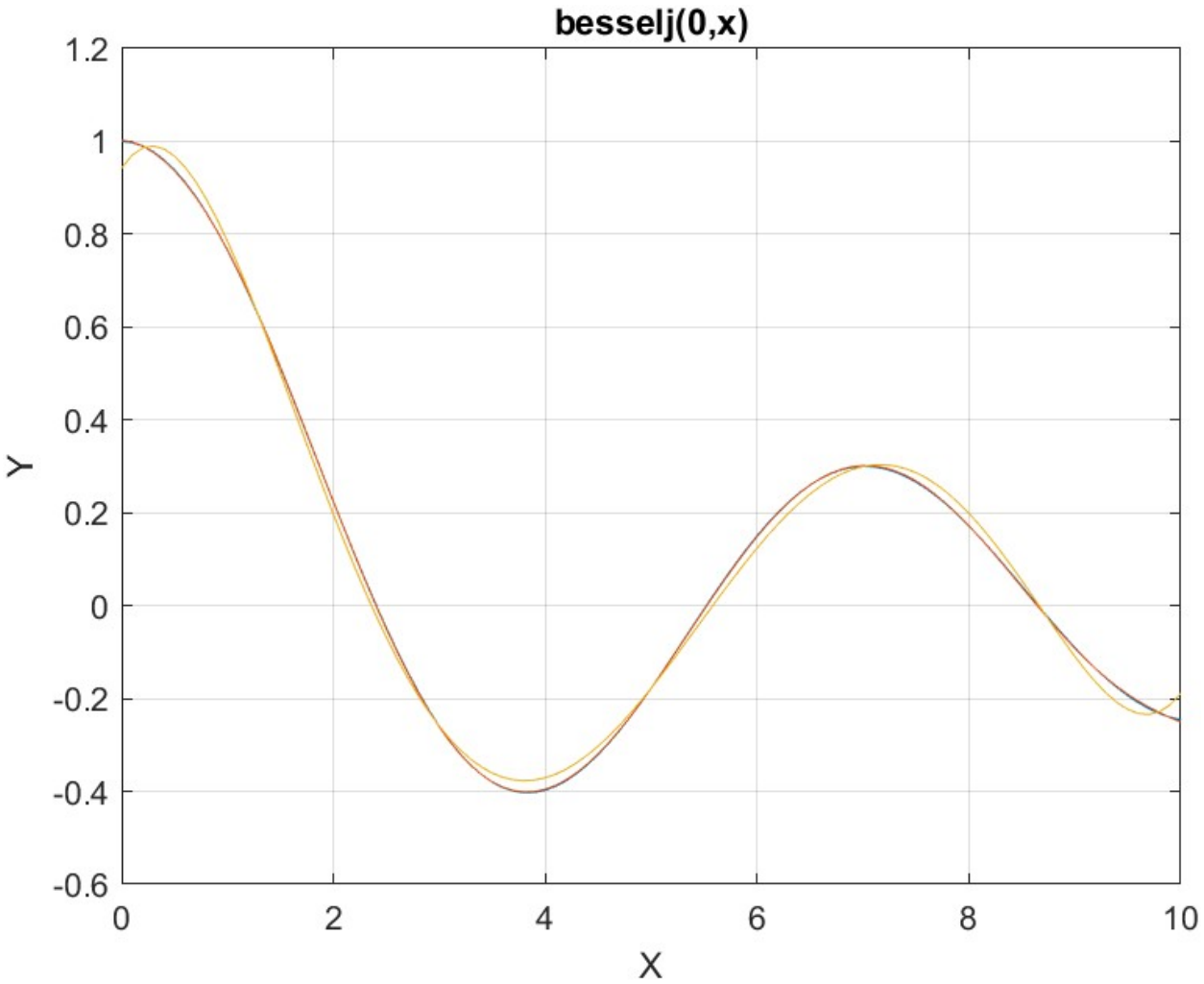
```
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

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

Lb1	Lb2	Lb3	Lb4	Lb5	Lb6	
0.926769902	2.037702145	3.187007761	4.386085965	5.633433594	6.925917454	
Ub1	Ub2	Ub3	Ub4	Ub5	Ub6	
1.95862018	3.111388492	4.309455571	5.555272988	6.846299405	8.179547167	
QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
1.731084724	3.411294119	4.583217621	5.409005654	5.27804906	6.266559028	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.002517661	-0.202228524	-0.079018157	0.089042845	0.095757195	-0.139595484	-0.00068967
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.999983147	0.996718149					

Table 10. Summary of the results appearing in file besselj_0_x_halton_random_fx2.xlsx.

The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is higher than the one for



the classical polynomial. This condition indicates that the Quantum Shammas Polynomial performed well when using the power function $p_2(i)$.

Here is the graph (from file besselj_0_x_halton_random_fx2.jpg) for the Bessel function and the two fitted polynomials:

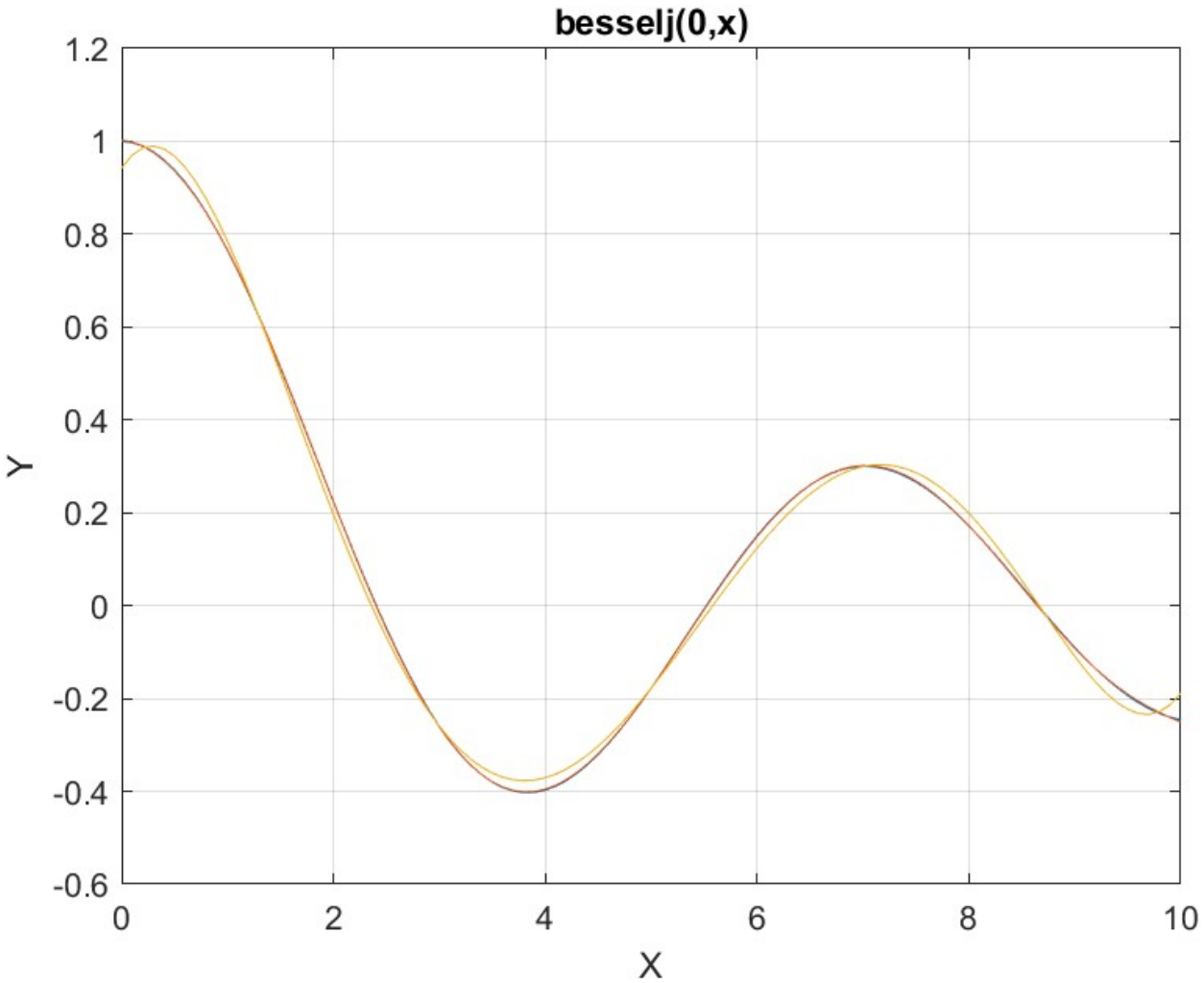


Figure 10. The graph from file besselj_0_x_halton_random_fx2.jpg.

The above graph shows the deviation of the regular polynomial from the Bessel function curve. The Quantum Shammas Polynomial fits the Bessel function well.

Testing Function P3(i) with the Halton Quasi-Random Search

The next MATLAB script (found in file testBessel3Halton.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial, using power function $p_3(i)$, and a sixth order classical polynomial.

```
clc
```



```
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_halton_random_fx3";
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);
sEqnFx = "x^0.75";
fprintf("%s\n", sEqn);
fprintf("%pwrFx = s\n", sEqnFx);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(@(x)3*x/4, order, 0.1);fprintf("       Lb
Ub\n");
fprintf("------------------------------\n")
for i=1:order
  fprintf("    %f         %f\n", Lb(i), Ub(i))
end

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

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



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

QSPpwr = bestX;
Coeff = flip(c);
T = array2table(Lb);
writetable(T,xlFile,"Sheet","Sheet1","Range","A1");
T = array2table(Ub);
writetable(T,xlFile,"Sheet","Sheet1","Range","A4");
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A7");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A10");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A13");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A16");

format short
diary off

function [Lb,Ub] = makeLimits(fx, order, gap)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  pwr = [];
  for i=1:order+1
    pwr =  [pwr fx(i)];
  end
  diff = pwr(2) - pwr(1);
  delta = (diff - gap)/2;
  Lb(1) = pwr(1) - delta;
  for i=2:order
    Ub(i-1) = pwr(i-1) + delta;
    diff = pwr(i+1) - pwr(i);
    delta = (diff - gap)/2;
    Lb(i) = pwr(i) - delta;
  end
  Ub(order) = pwr(order) + delta;
end
```



```
function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

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

Lb1	Lb2	Lb3	Lb4	Lb5	Lb6	
0.425	1.175	1.925	2.675	3.425	4.175	
Ub1	Ub2	Ub3	Ub4	Ub5	Ub6	
1.075	1.825	2.575	3.325	4.075	4.825	
QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
1.158053458	1.908778871	2.756663646	3.039837585	3.12693293	3.780818594	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.024454366	-0.823867308	2.847347324	-14.83127187	39.76219675	-27.43317026	0.234034224
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.998613693	0.996718149					

Table 11. Summary of the results appearing in file besselj_0_x_halton_random_fx3.xlsx.

The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is slightly higher than the one for the classical polynomial. This condition indicates that the Quantum Shammas Polynomial performed well when using the power function $p_3(i)$.

Here is the graph (from file besselj_0_x_halton_random_fx3.jpg) for the Bessel function and the two fitted polynomials:

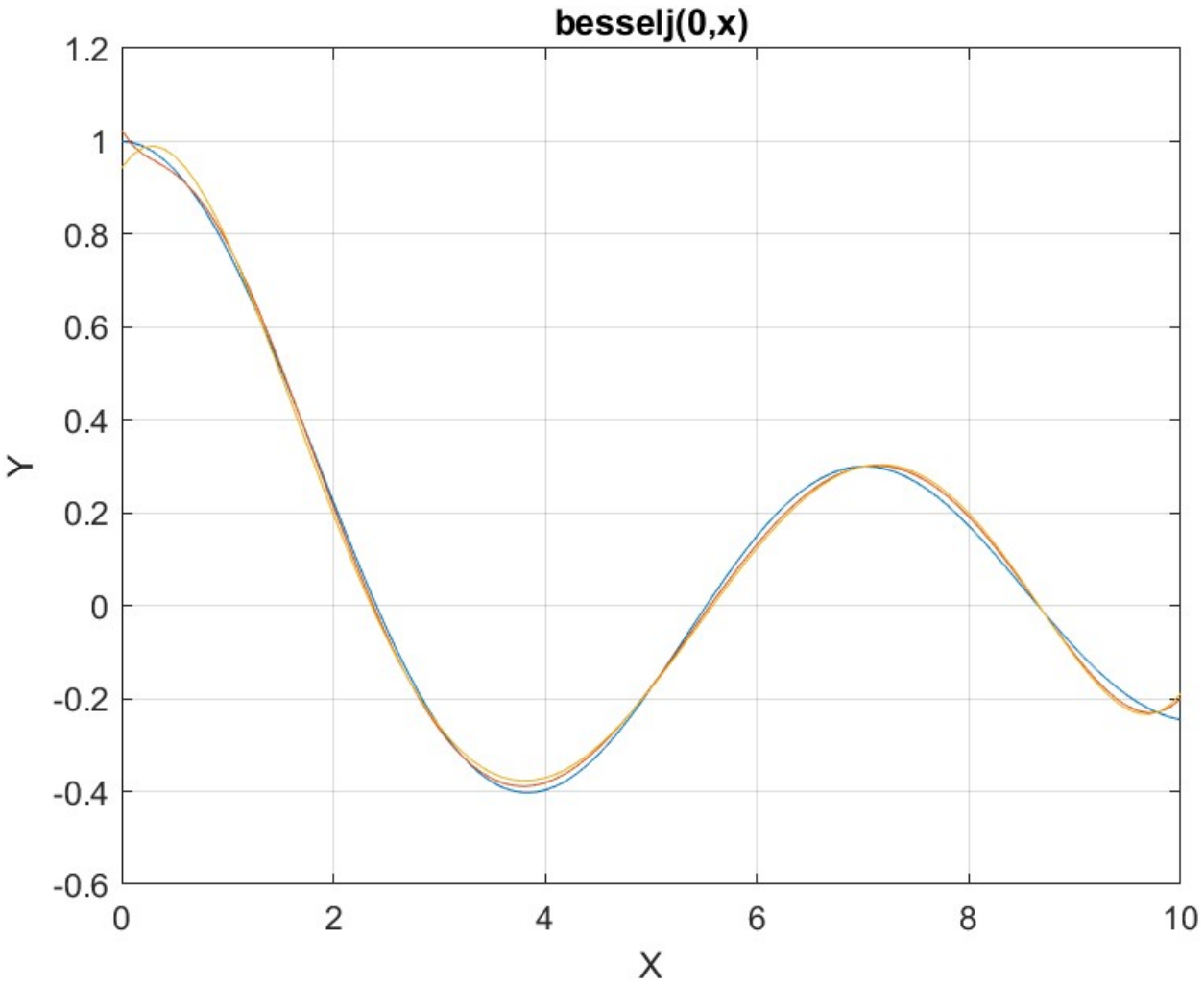


Figure 11. The graph from file besselj_0_x_halton_random_fx3.jpg.

The above graph shows some deviation of both polynomials from the Bessel function curve.

Testing Function P4(i) with the Halton Quasi-Random Search

The next MATLAB script (found in file testBessel4Halton.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial, using power function $p_4(i)$, and a sixth order classical polynomial.

```
clc
clear
```



```
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_halton_random_fx4";
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);
sEqnFx = "0.85*log(1+x)*sqrt(x)";
fprintf("%s\n", sEqn);
fprintf("pwrFx = %s\n", sEqnFx);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(@(x)0.85*log(1+x).*sqrt(x), order, 0.1);
fprintf("       Lb              Ub\n");
fprintf("--------------------------------\n")
for i=1:order
  fprintf("    %f         %f\n", Lb(i), Ub(i))
end

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

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



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

QSPpwr = bestX;
Coeff = flip(c);
T = array2table(Lb);
writetable(T,xlFile,"Sheet","Sheet1","Range","A1");
T = array2table(Ub);
writetable(T,xlFile,"Sheet","Sheet1","Range","A4");
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A7");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A10");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A13");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A16");

format short
diary off

function [Lb,Ub] = makeLimits(fx, order, gap)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  pwr = [];
  for i=1:order+1
    pwr =  [pwr fx(i)];
  end
  diff = pwr(2) - pwr(1);
  delta = (diff - gap)/2;
  Lb(1) = pwr(1) - delta;
  for i=2:order
    Ub(i-1) = pwr(i-1) + delta;
    diff = pwr(i+1) - pwr(i);
    delta = (diff - gap)/2;
    Lb(i) = pwr(i) - delta;
  end
  Ub(order) = pwr(order) + delta;
end

function r = rsqr(y,ycalc)
```



```
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

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

Lb1	Lb2	Lb3	Lb4	Lb5	Lb6	
0.273451886	1.010451094	1.743421416	2.451305888	3.132525411	3.789054658	
Ub1	Ub2	Ub3	Ub4	Ub5	Ub6	
0.904898321	1.630791983	2.338503439	3.020783014	3.678517742	4.313973158	
QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
0.988284454	1.751035814	2.548715204	3.302303303	3.400915458	3.709801356	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.024937747	-0.584695928	1.714908138	-3.058599572	10.68486569	-9.849384366	0.848588193
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.998531009	0.996718149					

Table 12. Summary of the results appearing in file besselj_0_x_halton_random_fx4.xlsx.

The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is slightly higher than the one for the classical polynomial. This condition indicates that the Quantum Shammas Polynomial performed well when using the power function $p_4(i)$.

Here is the graph (from file besselj_0_x_halton_random_fx4.jpg) for the Bessel function and the two fitted polynomials:

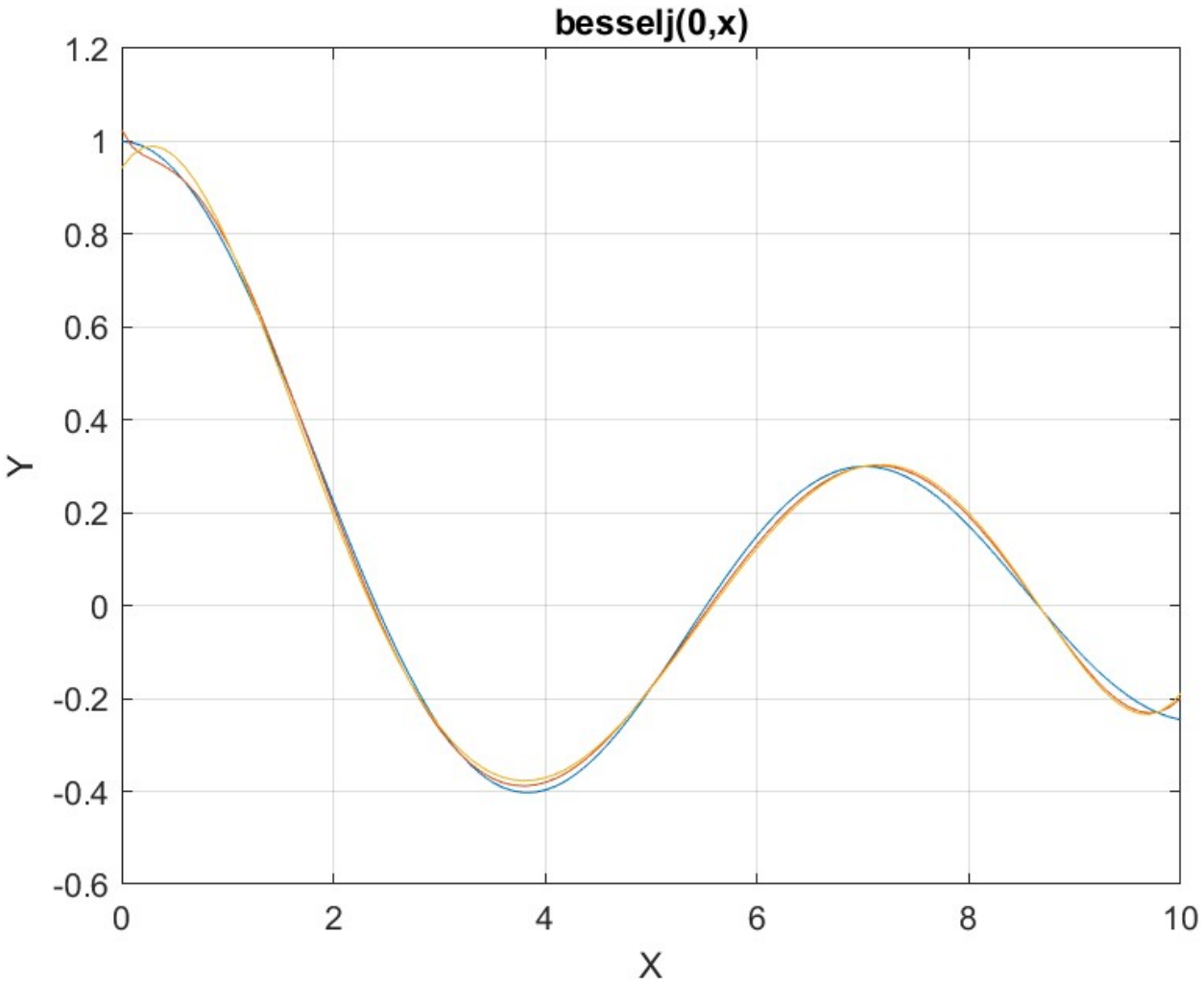


Figure 12. The graph from file besselj_0_x_halton_random_fx4.jpg.

The above graph shows some deviation of both polynomials from the Bessel function curve.

Conclusion for Using the Halton Quasi-Random Search Optimization Method

The random search optimization method did well with the power functions $p_2(i)$, $p_3(i)$, and $p_4(i)$. This is the same pattern as with the PSO and random search methods. So we are seeing a pattern where the power functions $p_2(i)$, $p_3(i)$, and $p_4(i)$ yield better least-square fits.

Testing Function P1(i) with the Sobol Quasi-Random Search

The next MATLAB script (found in file testBessel1Sobol.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial, using power function $p_1(i)$, and a sixth order classical polynomial.

```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_sobol_random_fx1";
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)";
sEqnFx = "sqrt(x)";
fprintf("%s\n", sEqn);
fprintf("pwrFx = %s\n", sEqnFx);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(@(x)x/2, order, 0.1);
fprintf("       Lb              Ub\n");
fprintf("--------------------------------\n")
for i=1:order
  fprintf("    %f         %f\n", Lb(i), Ub(i))
end

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

SSE = quantShammasPoly(bestX);
% calculate adjusted value of the coefficient of determination
glbRsqr = 1 - (1 - glbRsqr)*(n-1)/(n-order-1);
fprintf("Adjusted Rsqr = %f\n", glbRsqr);
fprintf("Quantum Shammas Polynomial Powers\n");
```



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

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

QSPpwr = bestX;
Coeff = flip(c);
T = array2table(Lb);
writetable(T,xlFile,"Sheet","Sheet1","Range","A1");
T = array2table(Ub);
writetable(T,xlFile,"Sheet","Sheet1","Range","A4");
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A7");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A10");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A13");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A16");

format short
diary off

function [Lb,Ub] = makeLimits(fx, order, gap)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  pwr = [];
  for i=1:order+1
    pwr =  [pwr fx(i)];
  end
  diff = pwr(2) - pwr(1);
```



```
  delta = (diff - gap)/2;
  Lb(1) = pwr(1) - delta;
  for i=2:order
    Ub(i-1) = pwr(i-1) + delta;
    diff = pwr(i+1) - pwr(i);
    delta = (diff - gap)/2;
    Lb(i) = pwr(i) - delta;
  end
  Ub(order) = pwr(order) + delta;
end

function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

The above code has uses the sobolRandomSearch() function to perform random search optimization. The above code also copies the console output to a diary text file. It writes the summary results to an Excel table, shown below:

Lb1	Lb2	Lb3	Lb4	Lb5	Lb6	
0.3	0.8	1.3	1.8	2.3	2.8	
Ub1	Ub2	Ub3	Ub4	Ub5	Ub6	
0.7	1.2	1.7	2.2	2.7	3.2	
QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
0.764087615	1.298586226	1.854904649	2.391193168	2.952480221	3.442384343	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.106890589	-3.635105105	11.25832927	-13.67411279	7.331746178	-1.753592103	0.195720609
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.988993813	0.996718149					

Table 13. Summary of the results appearing in file besselj_0_x_sobol_random_fx1.xlsx.

The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is less than the one for the classical polynomial. This condition indicates that the Quantum Shammas Polynomial performed with the power function $p_1(i)$.

Here is the graph (from file besselj_0_x_sobol_random_fx1.jpg) for the Bessel function and the two fitted polynomials:

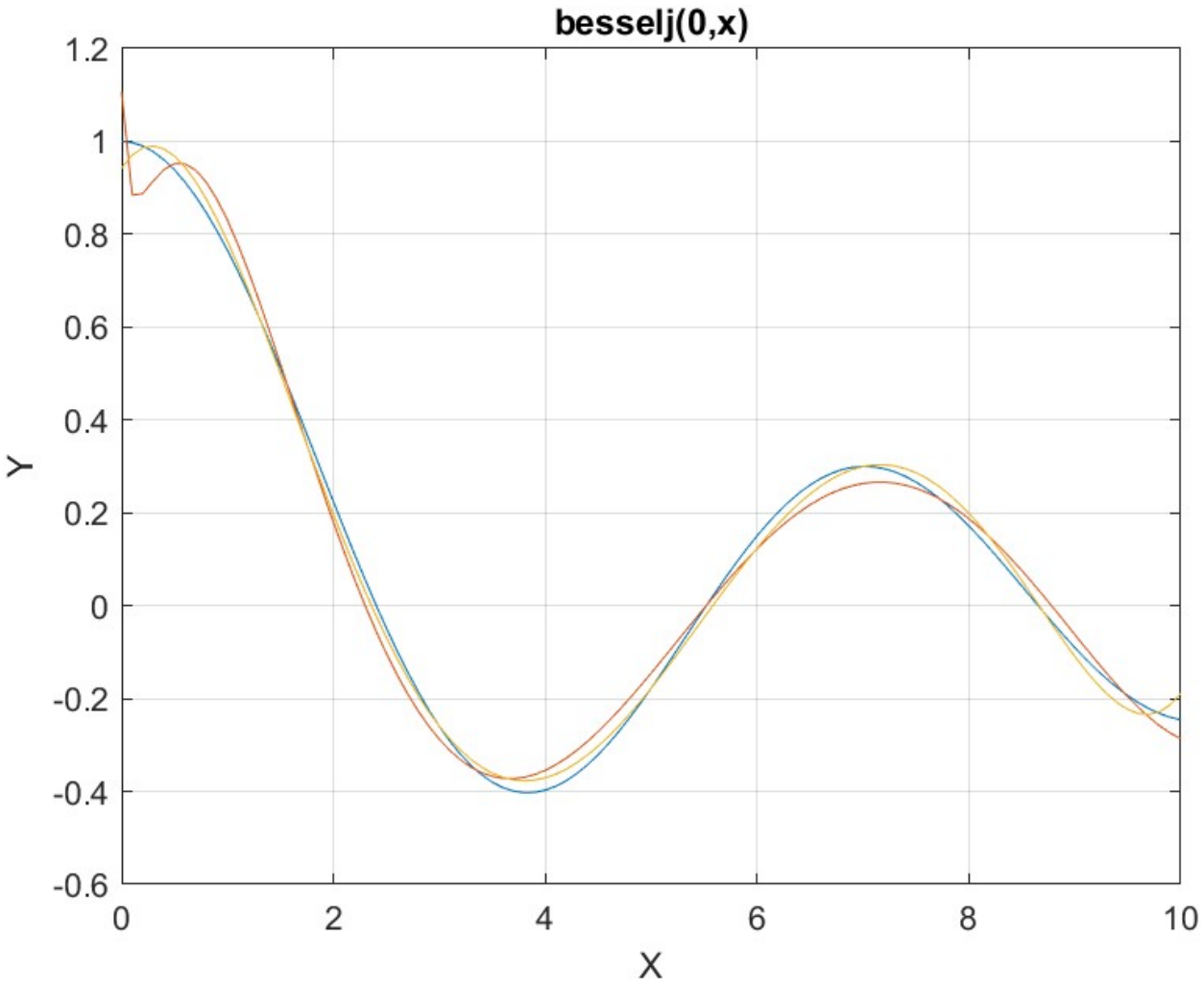


Figure 13. The graph from file besselj_0_x_sobol_random_fx1.jpg.

The above graph shows the deviation of both polynomials from the Bessel function curve. This is not surprising since I chose a wide range to fit, making the test a bit more difficult.

Testing Function P2(i) with the Sobol Quasi-Random Search

The next MATLAB script (found in file testBessel2Sobol.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial, using power function $p_2(i)$, and a sixth order classical polynomial.


```
clc
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_sobol_random_fx2";
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)";
sEqnFx = "x^1.5/ln(x+1)";
fprintf("%s\n", sEqn);
fprintf("pwrFx = %s\n", sEqnFx);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(@(x)x^1.5./log(1+x), order, 0.1);
fprintf("       Lb              Ub\n");
fprintf("-------------------------------\n")
for i=1:order
  fprintf("    %f         %f\n", Lb(i), Ub(i))
end

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

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



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

QSPpwr = bestX;
Coeff = flip(c);
T = array2table(Lb);
writetable(T,xlFile,"Sheet","Sheet1","Range","A1");
T = array2table(Ub);
writetable(T,xlFile,"Sheet","Sheet1","Range","A4");
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A7");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A10");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A13");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A16");

format short
diary off

function [Lb,Ub] = makeLimits(fx, order, gap)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  pwr = [];
  for i=1:order+1
    pwr =  [pwr fx(i)];
  end
  diff = pwr(2) - pwr(1);
  delta = (diff - gap)/2;
  Lb(1) = pwr(1) - delta;
  for i=2:order
    Ub(i-1) = pwr(i-1) + delta;
    diff = pwr(i+1) - pwr(i);
    delta = (diff - gap)/2;
    Lb(i) = pwr(i) - delta;
  end
  Ub(order) = pwr(order) + delta;
end
```



```
function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

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

Lb1	Lb2	Lb3	Lb4	Lb5	Lb6	
0.926769902	2.037702145	3.187007761	4.386085965	5.633433594	6.925917454	
Ub1	Ub2	Ub3	Ub4	Ub5	Ub6	
1.95862018	3.111388492	4.309455571	5.555272988	6.846299405	8.179547167	
QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
1.694800877	3.347641783	4.649943666	5.25281853	5.297897545	6.305177209	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.003075911	-0.197124493	-0.083273828	0.107118903	-0.462757634	0.399139844	-0.000503918
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.999982971	0.996718149					

Table 14. Summary of the results appearing in file besselj_0_x_sobol_random_fx2.xlsx.

The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is higher than the one for the classical polynomial. This condition indicates that the Quantum Shammas Polynomial performed well when using the power function $p_2(i)$.

Here is the graph (from file besselj_0_x_sobol_random_fx2.jpg) for the Bessel function and the two fitted polynomials:

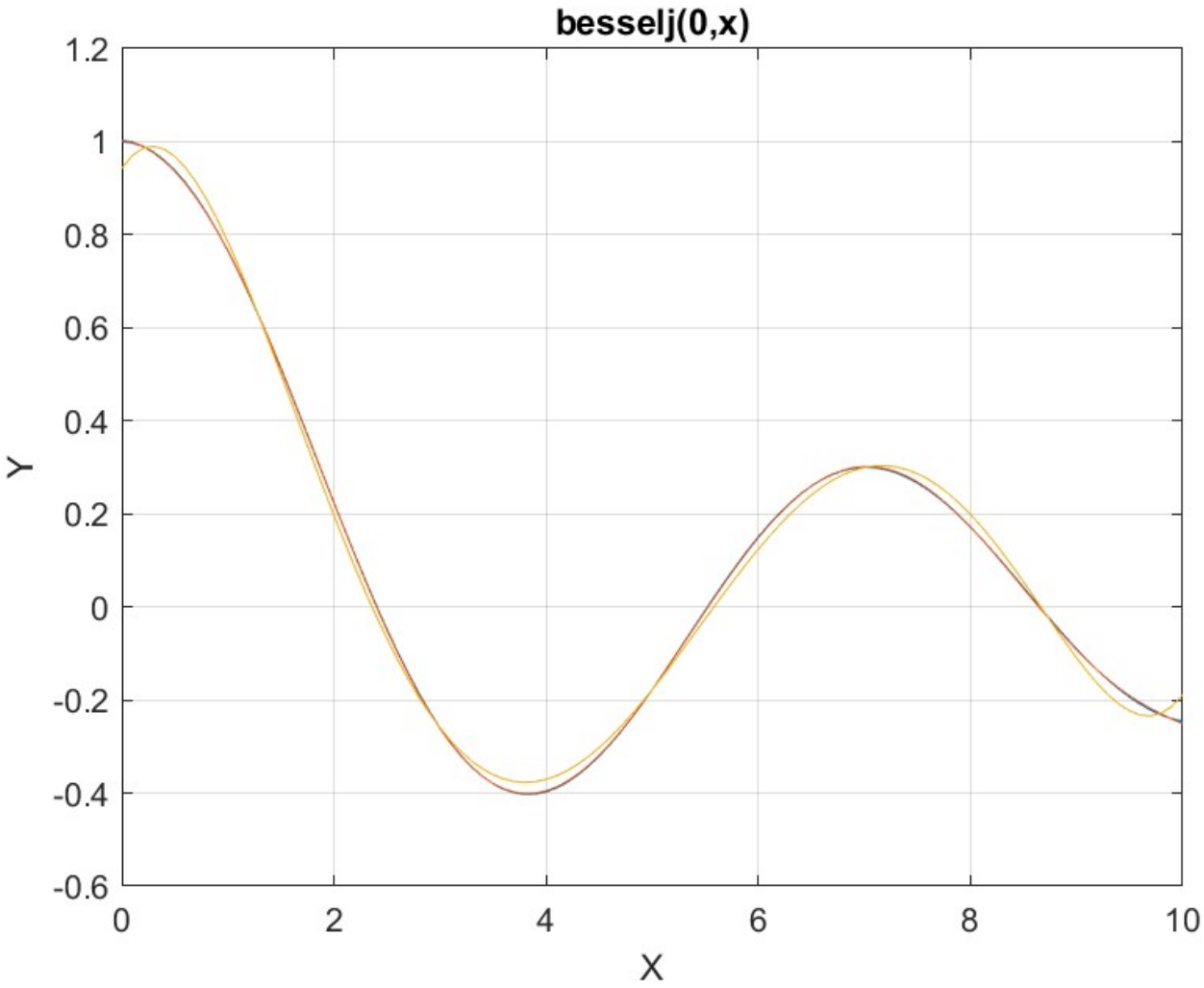


Figure 14. The graph from file besselj_0_x_sobol_random_fx2.jpg.

The above graph shows the deviation of the regular polynomial from the Bessel function curve. The Quantum Shammas Polynomial fits the Bessel function well.

Testing Function P3(i) with the Sobol Quasi-Random Search

The next MATLAB script (found in file testBessel3Sobol.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial, using power function $p_3(i)$, and a sixth order classical polynomial.

```
clc
clear
```



```
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_sobol_random_fx3";
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)";
sEqnFx = "x^0.75";
fprintf("%s\n", sEqn);
fprintf("%pwrFx = s\n", sEqnFx);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(@(x)3*x/4, order, 0.1);
fprintf("       Lb              Ub\n");
fprintf("--------------------------------\n")
for i=1:order
  fprintf("    %f         %f\n", Lb(i), Ub(i))
end

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

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

figure(1)
```



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

QSPpwr = bestX;
Coeff = flip(c);
T = array2table(Lb);
writetable(T,xlFile,"Sheet","Sheet1","Range","A1");
T = array2table(Ub);
writetable(T,xlFile,"Sheet","Sheet1","Range","A4");
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A7");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A10");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A13");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A16");

format short
diary off

function [Lb,Ub] = makeLimits(fx, order, gap)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  pwr = [];
  for i=1:order+1
    pwr =  [pwr fx(i)];
  end
  diff = pwr(2) - pwr(1);
  delta = (diff - gap)/2;
  Lb(1) = pwr(1) - delta;
  for i=2:order
    Ub(i-1) = pwr(i-1) + delta;
    diff = pwr(i+1) - pwr(i);
    delta = (diff - gap)/2;
    Lb(i) = pwr(i) - delta;
  end
  Ub(order) = pwr(order) + delta;
end

function r = rsqr(y,ycalc)
```



```
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

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

Lb1	Lb2	Lb3	Lb4	Lb5	Lb6	
0.425	1.175	1.925	2.675	3.425	4.175	
Ub1	Ub2	Ub3	Ub4	Ub5	Ub6	
1.075	1.825	2.575	3.325	4.075	4.825	
QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
1.181412047	1.958224634	2.812045149	2.818717478	3.16253982	3.793264624	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.026077431	-0.912606336	3.673186811	-640.0062674	644.1576375	-7.352136564	0.194108847
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.998620885	0.996718149					

Table 15. Summary of the results appearing in file besselj_0_x_sobol_random_fx3.xlsx.

The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is slightly higher than the one for the classical polynomial. This condition indicates that the Quantum Shammas Polynomial performed well when using the power function $p_3(i)$.

Here is the graph (from file besselj_0_x_sobol_random_fx3.jpg) for the Bessel function and the two fitted polynomials:

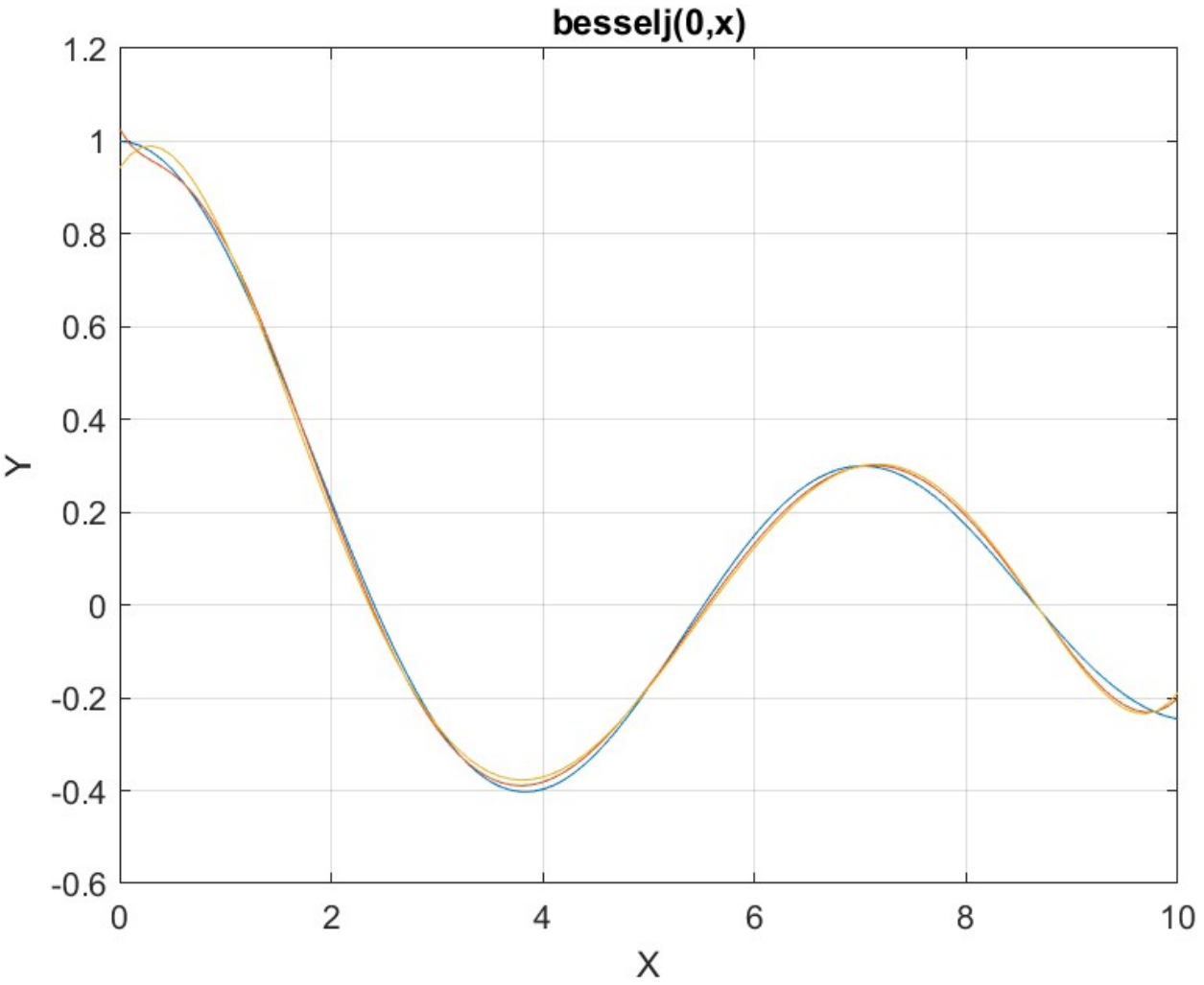


Figure 15. The graph from file besselj_0_x_sobol_random_fx3.jpg.

The above graph shows some deviation of both polynomials from the Bessel function curve.

Testing Function P4(i) with the Sobol Quasi-Random Search

The next MATLAB script (found in file testBessel4Sobol.m) tests fitting Bessel J(0, x) for x in the range (0, 10) and samples at 0.1 steps. The curve fits use a sixth order Quantum Shammas Polynomial, using power function $p_4(i)$, and a sixth order classical polynomial.

```
clc
```



```
clear
close all

global xData yData yCalc glbRsqr QSPcoeff
zFilename = "besselj_0_x_sobol_random_fx4";
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)";
sEqnFx = "0.85*log(1+x)*sqrt(x)";
fprintf("%s\n", sEqn);
fprintf("pwrFx = %s\n", sEqnFx);
fprintf("x=0:0.1:10\n")
xData= 0:0.1:10;
xData = xData';
n = length(xData);
yData = besselj(0,xData);
order = 6;
[Lb,Ub] = makeLimits(@(x)0.85*log(1+x).*sqrt(x), order, 0.1);
fprintf("       Lb              Ub\n");
fprintf("--------------------------------\n")
for i=1:order
  fprintf("    %f         %f\n", Lb(i), Ub(i))
end

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

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



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

QSPpwr = bestX;
Coeff = flip(c);
T = array2table(Lb);
writetable(T,xlFile,"Sheet","Sheet1","Range","A1");
T = array2table(Ub);
writetable(T,xlFile,"Sheet","Sheet1","Range","A4");
T1 = array2table(QSPpwr);
writetable(T1,xlFile,"Sheet","Sheet1","Range","A7");
T2 = array2table(QSPcoeff);
writetable(T2,xlFile,"Sheet","Sheet1","Range","A10");
T3 = array2table(Coeff);
writetable(T3,xlFile,"Sheet","Sheet1","Range","A13");
r_sqr = [glbRsqr r];
T4 = array2table(r_sqr);
writetable(T4,xlFile,"Sheet","Sheet1","Range","A16");

format short
diary off

function [Lb,Ub] = makeLimits(fx, order, gap)
  Lb = zeros(1,order);
  Ub = zeros(1,order);
  pwr = [];
  for i=1:order+1
    pwr =  [pwr fx(i)];
  end
  diff = pwr(2) - pwr(1);
  delta = (diff - gap)/2;
  Lb(1) = pwr(1) - delta;
  for i=2:order
    Ub(i-1) = pwr(i-1) + delta;
    diff = pwr(i+1) - pwr(i);
    delta = (diff - gap)/2;
    Lb(i) = pwr(i) - delta;
  end
  Ub(order) = pwr(order) + delta;
end
```



```
function r = rsqr(y,ycalc)
  n = length(y);
  ymean = mean(y);
  SStot = sum((y - ymean).^2);
  SSE = sum((y - ycalc).^2);
  r = 1 - SSE / SStot;
end
```

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

Lb1	Lb2	Lb3	Lb4	Lb5	Lb6	
0.273451886	1.010451094	1.743421416	2.451305888	3.132525411	3.789054658	
Ub1	Ub2	Ub3	Ub4	Ub5	Ub6	
0.904898321	1.630791983	2.338503439	3.020783014	3.678517742	4.313973158	
QSPpwr1	QSPpwr2	QSPpwr3	QSPpwr4	QSPpwr5	QSPpwr6	
0.977685513	1.783664938	2.541833776	3.240401888	3.513866017	3.597946164	
QSPcoeff1	QSPcoeff2	QSPcoeff3	QSPcoeff4	QSPcoeff5	QSPcoeff6	QSPcoeff7
1.026404268	-0.580421626	1.864092524	-3.377288723	5.605248918	-8.895167791	5.13812797
Coeff1	Coeff2	Coeff3	Coeff4	Coeff5	Coeff6	Coeff7
0.942551329	0.346766161	-0.688054603	0.203338833	-0.020739115	0.000528234	1.54357E-05
r_sqr1	r_sqr2					
0.998536206	0.996718149					

Table 16. Summary of the results appearing in file besselj_0_x_sobol_random_fx4.xlsx.

The adjusted coefficient of determination for the fitted Quantum Shammas Polynomial is slightly higher than the one for the classical polynomial. This condition indicates that the Quantum Shammas Polynomial performed well when using the power function $p_4(i)$.

Here is the graph (from file besselj_0_x_sobol_random_fx4.jpg) for the Bessel function and the two fitted polynomials:

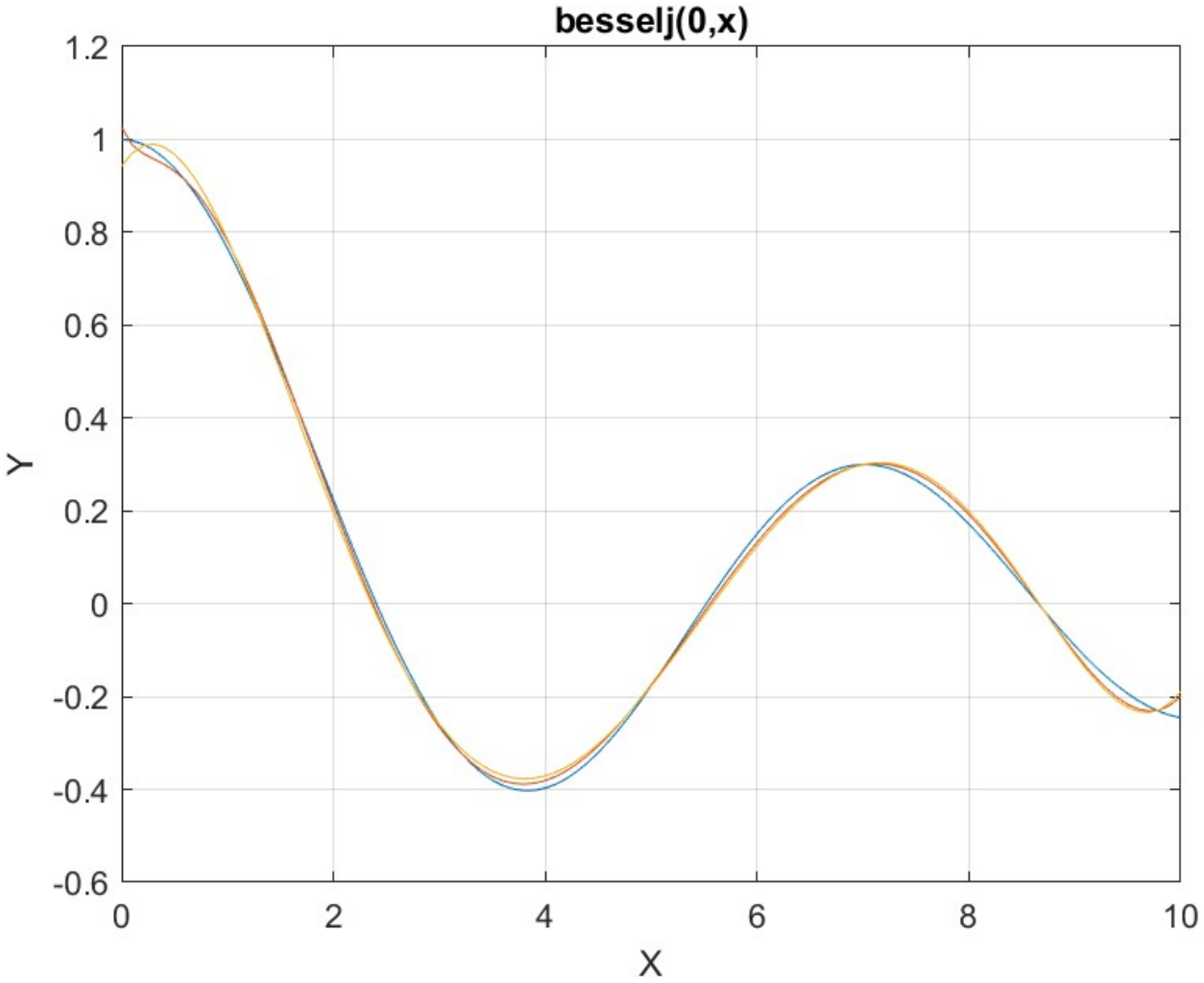


Figure 16. The graph from file besselj_0_x_sobol_random_fx4.jpg.

The above graph shows some deviation of both polynomials from the Bessel function curve.

Conclusion for Using the Sobol Quasi-Random Search Optimization Method

The random search optimization method did well with the power functions $p_2(i)$, $p_3(i)$, and $p_4(i)$. This is the same pattern as with the previous optimization methods. So we are seeing a pattern where the power functions $p_2(i)$, $p_3(i)$, and $p_4(i)$ yield better least-square fits.

Conclusion for Part 1E

The Quantum Shammas Polynomials did well with the power functions $p_2(i)$, $p_3(i)$, and $p_4(i)$. The results so far are encouraging. Of course, they show that choosing a good power function makes all the difference

Next is Part 2

Part 2 of this study looks at the Quantum Shammas Padé Polynomials.

Document History

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

