Implicit Shammas Interpolation
By
Namir C. Shammas

INTRODUCTION

In the HHC 2008 conference held in Corvallis, Oregon, I introduced the Shammas polynomials as polynomials with non-integer powers. I explained that the powers of such polynomial change using various math expressons that involves the

polynomial term sequence number. Moreover, the study explained that such expressons must never produce the same powers of two or more terms. In the 2008 presentation I showed that the Shammas polynomials can have multiple terms with a smaller power range compared to regular polynomials. This feature allows better curve fitting with Shammas polynomials that avoids the instability of raising large or small numbers to high powers. If you are not familiar with the Shammas polynomials, then you can download my 2008 presentation using the following link:

http://www.namirshammas.com/NEW/ShammasPolynomials.pdf.

A Shammas polynomial has the following general form:

$$y = = a_0 + \sum_{i=1}^{n} a_i \, x^{gx(i,A,B,C)} \quad (1)$$

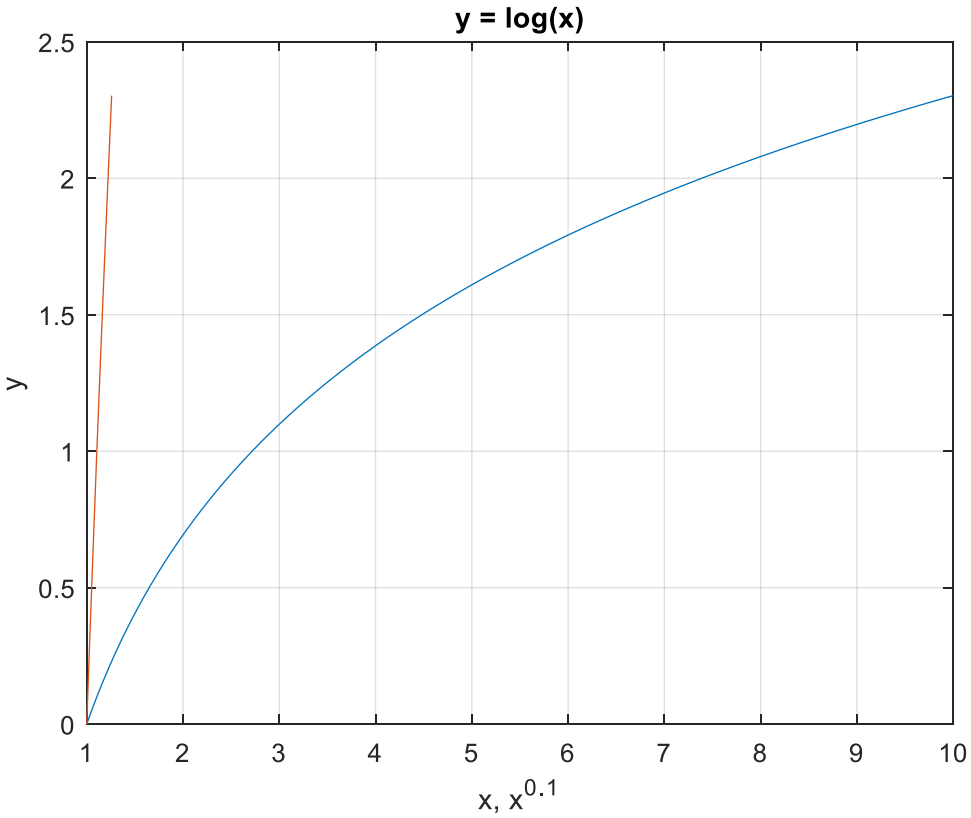
Where A, B, and C are fixed constants. Using C is optional. The function gx(i, A, B, C) supplies the math expressions used to calculate the powers of the Shammas polynomial. The power can be integers and non-integers.

I have used Shammas polynomials for function approximation in either a direct approach or as Pade-Shammas approximations. In either case I use explicit forms of the Shammas polynomials.

In this study I use an implicit (and much simpler to execute) approach in conducting interpolations with the following algorithms:

- Lagrangian interpolation.
- Newton divided-difference interpolation.
- Neville interpolation.
- Barylag 1-D barycentric lagrange interpolation).
- Rational interpolation using the Bulirsch-Stoer Method.
- Rational interpolation using the Floater-Hormann Method.
- Hermite divided difference interpolation.
- Interpolation using simple rational polynomials.
- Rational interpolation using the Schneider-Werner Method.
- Steffen interpolation.
- Stineman interpolation.
- Thiele interpolation.

The approach of implicit Shammas interpolation is surprisingly simple. You replace the values for x (in both the data and the interpolating value of x) with x^p, where p < 1, and use one of the above interpolation methods. I suggest values for p like 0.5 and 0.1. The aim is to obtain results for the interpolated y value that are more accurate than in using p = 1 (i.e. normal practice of interpolation). The caviate here is that it depends on the underlying interpolated function. The easiest way to determine if the Shammas interpolation scheme would work is to plot the lines connecting the data for (x, y), ($x^{0.5}$, y), and ($x^{0.1}$, y). If the transformed data form a straight line (or close to it), then Shammas interpolation scheme would work. The next figure shows the data for (x, ln(x)) in blue and ($x^{0.1}$, ln(x)) in red. Notice that the red points form a straight line.



The linear regression for the (x, ln(x)) data in the range of (1, 10) is:

ln(x) = 0.354256369100258 + 0.218527532581403*x

With a coefficient of determination of 0.926325408582826. Compare the above data with the linear regression for the ($x^{0.1}$, ln(x)) data in the range of (1, 10):

ln(x) = –8.67494440136651 + 8.74152632436559 *x^0.1

With a coefficient of determination of 0.999083610284736. These statistical results confirm the visual representation of the data in the above graph.

✍ The implicit Shammas interpolation is rather unique in that is it essentially a *parasitic* method. It simply feeds on transforming the x data used by the server interpolation methods. The method does not influence/alter the algorithm's steps used by these interpolation methods.

THE TESTED FUNCTIONS

This study conducts implicit Shammas interpolation for the following functions:

- The ln(x) function.
- The $\ln(x^2+1)$ function.
- The $\Gamma(x)/\ \Gamma(x+1)$ function.
- The reciprocal function.
- The square root.
- A first Pade polynomial:
  - y = (x.^3-2*x^2+3*x–4)/(x^2+x+1)
- A second Pade polynomial:
  - y = (x^2+5*x+10)/(x^3+x^2+x+1)
- An arbitrary trigonometric function:
  - y = (5*x^2 + sin(x))/(3*x^2 + cos(x))

MATLAB CODE

The next subsections show matlab code for testing the various interpolation algorithms, I obtained most of the Matlab code for the interpolation from the Matlab File Exchange by downloading the file interputils.zip. The Matlab code files in this zip (tagged as version 1.39.0.0) where coded by Joe Henning. The URL for Henning's page in Matlab's File Exchange web pages is:

https://www.mathworks.com/matlabcentral/fileexchange/36800-interpolation-utilities?s_tid=srchtitle

The Implicit Shammas and Lagragian Interpolation

The next listing shows the test code for applying the implicit Shammas interpolation approach to Lagrangian interpolation.

```
clc
clear
```



```
ExcelFilename = 'lagragngeInterpol.xlsx';
% test f(x) = log(x)
fprintf('---------------------------------- log(x)\n');
x=1:5;
y = log(x);

xint = 1.5:4.5;
yintc = log(xint);
fprintf('xint = \n');
disp(xint);
fprintf('yintc = \n');
disp(yintc);
yint = zeros(length(xint),1);
percerr = zeros(length(xint),1);
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x,y,xint(i));
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr');
yint1 = yint;
percerr1 = percerr;

p = 0.5;
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint2 = yint;
percerr2 = percerr;

p = 0.1;
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint3 = yint;
percerr3 = percerr;
```



```
n = length(xint);
X = x(1:n)';
Y = y(1:n)';
zVarNames =
{'X','Y','Xint','Yint','Yint1','PercnErr1','Yint2','PercnErr2','Yint3','Percn
Err3'};
T =
table(X,Y,xint',yintc',yint1,percerr1,yint2,percerr2,yint3,percerr3,'Variable
Names',zVarNames);
writetable(T, ExcelFilename, 'Sheet','log');

% test f(x) = log(x^2+1)
fprintf('---------------------------------- log(x^2+1)\n');
x=1:5;
y = log(x.^2+1);

xint = 1.5:4.5;
yintc = log(xint.^2+1);
fprintf('xint = \n');
disp(xint);
fprintf('yintc = \n');
disp(yintc);
yint = zeros(length(xint),1);
percerr = zeros(length(xint),1);
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x,y,xint(i));
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr');
yint1 = yint;
percerr1 = percerr;

p = 0.5;
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint2 = yint;
percerr2 = percerr;

p = 0.1;
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
```



```
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint3 = yint;
percerr3 = percerr;

n = length(xint);
X = x(1:n)';
Y = y(1:n)';
zVarNames =
{'X','Y','Xint','Yint','Yint1','PercnErr1','Yint2','PercnErr2','Yint3','Percn
Err3'};
T =
table(X,Y,xint',yintc',yint1,percerr1,yint2,percerr2,yint3,percerr3,'Variable
Names',zVarNames);
writetable(T, ExcelFilename, 'Sheet','log(x2 plus 1)');


% test f(x) = gamma(x)/gamma(x+1)
fprintf('----------------------------------- gamma(x)/gamma(x+1)\n');
x=2:6;
y = gamma(x)./gamma(x+1);

xint = 2.5:5.5;
yintc = gamma(xint)./gamma(xint+1);
fprintf('xint = \n');
disp(xint);
fprintf('yintc = \n');
disp(yintc);
yint = zeros(length(xint),1);
percerr = zeros(length(xint),1);
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x,y,xint(i));
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr');
yint1 = yint;
percerr1 = percerr;

p = 0.5;
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
```



```
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint2 = yint;
percerr2 = percerr;

p = 0.1;
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint3 = yint;
percerr3 = percerr;

n = length(xint);
X = x(1:n)';
Y = y(1:n)';
zVarNames =
{'X','Y','Xint','Yint','Yint1','PercnErr1','Yint2','PercnErr2','Yint3','Percn
Err3'};
T =
table(X,Y,xint',yintc',yint1,percerr1,yint2,percerr2,yint3,percerr3,'Variable
Names',zVarNames);
writetable(T, ExcelFilename, 'Sheet','gamma ratio');



% test f(x) = 1/x
fprintf('---------------------------------- 1/x\n');
x=1:5;
y = 1./x;

xint = 1.5:4.5;
yintc = 1./xint;
fprintf('xint = \n');
disp(xint);
fprintf('yintc = \n');
disp(yintc);
yint = zeros(length(xint),1);
percerr = zeros(length(xint),1);
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x,y,xint(i));
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
```



```
fprintf('%%error = \n');
disp(percerr');
yint1 = yint;
percerr1 = percerr;

p = 0.5;
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint2 = yint;
percerr2 = percerr;

p = 0.1;
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint3 = yint;
percerr3 = percerr;

n = length(xint);
X = x(1:n)';
Y = y(1:n)';
T =
table(X,Y,xint',yintc',yint1,percerr1,yint2,percerr2,yint3,percerr3,'Variable
Names',zVarNames);
writetable(T, ExcelFilename, 'Sheet','recip');

% test f(x) = sqrt(x)
fprintf('---------------------------------- sqrt(x)\n');
x=2:6;
y = sqrt(x);

xint = 2.5:5.5;
yintc = sqrt(xint);
fprintf('xint = \n');
disp(xint);
fprintf('yintc = \n');
disp(yintc);
yint = zeros(length(xint),1);
percerr = zeros(length(xint),1);
for i=1:length(xint)
```



```
  yint(i) = lagragngeInterpol(x,y,xint(i));
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr');
yint1 = yint;
percerr1 = percerr;

p = 0.5;
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint2 = yint;
percerr2 = percerr;

p = 0.1;
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint3 = yint;
percerr3 = percerr;

n = length(xint);
X = x(1:n)';
Y = y(1:n)';
T =
table(X,Y,xint',yintc',yint1,percerr1,yint2,percerr2,yint3,percerr3,'Variable
Names',zVarNames);
writetable(T, ExcelFilename, 'Sheet','sqrt');


% test f(x) = Pade1(x)
fprintf('---------------------------------- Pade1(x)\n');
x=2:0.5:4;
y = pade1(x);

xint = 2.25:0.5:3.75;
```



```
yintc = pade1(xint);
fprintf('xint = \n');
disp(xint);
fprintf('yintc = \n');
disp(yintc);
yint = zeros(length(xint),1);
percerr = zeros(length(xint),1);
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x,y,xint(i));
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr');
yint1 = yint;
percerr1 = percerr;

p = 0.5;
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint2 = yint;
percerr2 = percerr;

p = 0.1;
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint3 = yint;
percerr3 = percerr;
n = length(xint);
X = x(1:n)';
Y = y(1:n)';
T =
table(X,Y,xint',yintc',yint1,percerr1,yint2,percerr2,yint3,percerr3,'Variable
Names',zVarNames);
writetable(T, ExcelFilename, 'Sheet','pade1');
```



```
% test f(x) = pade2(x)
fprintf('---------------------------------- pade2(x)\n');
x=2:0.5:4;
y = pade2(x);

xint = 2.25:0.5:3.75;
yintc = pade2(xint);
fprintf('xint = \n');
disp(xint);
fprintf('yintc = \n');
disp(yintc);
yint = zeros(length(xint),1);
percerr = zeros(length(xint),1);
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x,y,xint(i));
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr');
yint1 = yint;
percerr1 = percerr;

p = 0.5;
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint2 = yint;
percerr2 = percerr;

p = 0.1;
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint3 = yint;
percerr3 = percerr;
n = length(xint);
X = x(1:n)';
```



```
Y = y(1:n)';
T =
table(X,Y,xint',yintc',yint1,percerr1,yint2,percerr2,yint3,percerr3,'Variable
Names',zVarNames);
writetable(T, ExcelFilename, 'Sheet','pade2');

% test f(x) = trigFx1(x)
fprintf('---------------------------------- trigXf1(x)\n');
x=2:0.5:4;
y = trigFx1(x);

xint = 2.25:0.5:3.75;
yintc = trigFx1(xint);
fprintf('xint = \n');
disp(xint);
fprintf('yintc = \n');
disp(yintc);
yint = zeros(length(xint),1);
percerr = zeros(length(xint),1);
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x,y,xint(i));
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr');
yint1 = yint;
percerr1 = percerr;

p = 0.5;
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint2 = yint;
percerr2 = percerr;

p = 0.1;
for i=1:length(xint)
  yint(i) = lagragngeInterpol(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
```



```
fprintf('%%error = \n');
disp(percerr')
yint3 = yint;
percerr3 = percerr;
n = length(xint);
X = x(1:n)';
Y = y(1:n)';
T =
table(X,Y,xint',yintc',yint1,percerr1,yint2,percerr2,yint3,percerr3,'Variable
Names',zVarNames);
writetable(T, ExcelFilename, 'Sheet','trig fx1');

function y = pade1(x)
  y = (x.^3-2*x.^2+3*x-4)./(x.^2+x+1);
end

function y = pade2(x)
  y = (x.^2+5*x+10)./(x.^3+x.^2+x+1);
end

function y = trigFx1(x)
  y = (5*x.^2 + sin(x))./(3*x.^2 + cos(x));
end
```

The above code tests the various sample functions using the Lagrangian interpolation. Each test involves the following tasks:

1. Assign the values for x and calculate the corresponding values of y.
2. Assign the values for xint (the interpolating values of x) an calculate the corresponding values of the interpolated y values--yintc. The xint and yintc arrays have one row less than the arrays x and y to avoid extrapolation (i.e. $x_i < xint_i < x_{i+1}$, for i=1 to n).
3. Perform the regular interpolations for the array of xint and store the results in yint. Also store the percent errors of yint in array percerr.
4. Display the results and copy the yint and percerr values.
5. Repeat steps 3 and 4 for the implicit Shammas interpolation. Use p = 0.5.
6. Repeat steps 3 and 4 for the implicit Shammas interpolation. Use p = 0.1.
7. Build a Matlab table T from all the values for array x, y, xint, yintc, and the three sets of yint and percerr arrays. The table includes the name of the columns found in the cell variable zVarNames. The values for arrays x and y are truncated by excluding the last array elements. This step is required by Matlab which expects that all arrays in the table T to have the same number of rows.
8. Write the table T to an Excel file using a sheet name whose name is based on the tested function. The name of the Excel file is related to the algorithm tested and starts with the leading string “test_”.

The code for the Matlab function lagragngeInterpol is:

```
function yint = lagragngeInterpol(x,y,xint)
%LAGRAGNGEINTERPOL Summary of this function goes here
%   Detailed explanation goes here
  yint = 0;
  n = length(x);
  for i=1:n
    prod = y(i);
    for j=1:n
      if i~=j
        prod = prod * (xint - x(j)) / (x(i) - x(j));
      end
    end
    yint = yint + prod;
  end
end
```


Results for ln(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0	1.5	0.405465108	0.397709356	-1.912803726	0.404708133	-0.186692922	0.405463366	-0.00042964
2	0.693147181	2.5	0.916290732	0.918487214	0.239714487	0.916436294	0.01588601	0.916290973	2.6319E-05
3	1.098612289	3.5	1.252762968	1.251114332	-0.131600048	1.252681061	-0.006538173	1.25276286	-8.63028E-06
4	1.386294361	4.5	1.504077397	1.507162487	0.205115127	1.504198805	0.00807197	1.504077531	8.94061E-06

Results for ln(x^2+1)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0.693147181	1.5	1.178654996	1.183124949	0.379241835	1.181863998	0.272259607	1.177849663	-0.068326475
2	1.609437912	2.5	1.981001469	1.980025156	-0.049283812	1.980324349	-0.034180666	1.981047903	0.002343996
3	2.302585093	3.5	2.583997552	2.584575466	0.022365088	2.584382421	0.014894306	2.583991546	-0.00023244
4	2.833213344	4.5	3.056356895	3.055475126	-0.028850339	3.055791126	-0.01851123	3.056353761	-0.000102554

Results for 1/x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	1	1.5	0.666666667	0.684895833	2.734375	0.671777056	0.766558348	0.667423942	0.113591357
2	0.5	2.5	0.4	0.3953125	-1.171875	0.399104056	-0.223985955	0.399904072	-0.023982021
3	0.333333333	3.5	0.285714286	0.2890625	1.171875	0.286192723	0.167452956	0.285755013	0.014254402
4	0.25	4.5	0.222222222	0.216145833	-2.734375	0.22153751	-0.308120542	0.222173492	-0.021928423

Results for √x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.414213562	2.5	1.58113883	1.580836269	-0.01913566	1.58113883	-4.213E-14	1.581138627	-1.28307E-05
3	1.732050808	3.5	1.870828693	1.870930393	0.00543608	1.870828693	2.37376E-14	1.870828735	2.25044E-06
4	2	4.5	2.121320344	2.121236363	-0.003958879	2.121320344	-2.09346E-14	2.12132032	-1.12032E-06
5	2.236067977	5.5	2.34520788	2.345375099	0.007130238	2.34520788	0	2.345207914	1.47025E-06

Results for pade1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.285714286	2.25	0.483082707	0.482827108	-0.052910023	0.482994794	-0.018198206	0.483083458	0.00015561
2.5	0.679487179	2.75	0.877071823	0.877160386	0.010097583	0.87709954	0.003160158	0.877072606	8.92452E-05
3	1.076923077	3.25	1.279535865	1.279462065	-0.00576774	1.279514811	-0.001645462	1.279534833	-8.06298E-05
3.5	1.485074627	3.75	1.693521595	1.693668505	0.008674842	1.693559922	0.002263188	1.693523943	0.000138669

Results for pade2(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.6	2.25	1.335448057	1.336574015	0.084313147	1.335576946	0.009651369	1.335330855	-0.008776272
2.5	1.133004926	2.75	0.975182482	0.974769466	-0.042352666	0.975134294	-0.004941386	0.975209492	0.002769717
3	0.85	3.25	0.749125596	0.749484524	0.047912958	0.749166015	0.005395446	0.749109839	-0.002103407
3.5	0.666666667	3.75	0.598383927	0.597645685	-0.123372497	0.598305402	-0.01312282	0.598406771	0.003817635

Results for Gamma Ratio

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.5	2.5	0.4	0.401822917	0.455729167	0.400462841	0.115710356	0.400063248	0.015811988
3	0.333333333	3.5	0.285714286	0.28515625	-0.1953125	0.285604889	-0.038288949	0.285702242	-0.004215261
4	0.25	4.5	0.222222222	0.22265625	0.1953125	0.222290958	0.030931148	0.22222861	0.002874599
5	0.2	5.5	0.181818182	0.180989583	-0.455729167	0.181708897	-0.060106915	0.181809353	-0.004855873

Results for TrigFx1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.8050382	2.25	1.79201788	1.79179439	-0.01247132	1.79198221	-0.00199066	1.79204165	0.00132631
2.5	1.77440119	2.75	1.75498847	1.75506652	0.0044469	1.75499807	0.00054683	1.75497984	-0.00049219
3	1.73552891	3.25	1.71712341	1.71705753	-0.00383685	1.71711708	-0.00036865	1.71713075	0.00042714
3.5	1.70045215	3.75	1.68590999	1.68604316	0.0078988	1.68592017	0.00060368	1.68589545	-0.00086265

Calculating the Results Summary

The next subsection presents the results summary. The values in this summary are calculated based on the following steps:

1. Calculate the ratio of percent errors relative to the percent error at p = 1. If a percent error for p < 1 is 0, replace it with 1e-99 to get a defined, albeit large, ratio.
2. Average the percent errors ratios for each tested function. The average of the ratios at p=1 is always 1.
3. Rank the average ratio for each p value.
4. Total the ranks for first, second, and third places for p=1, 0.5, and 0.1. The result is a 3 by 3 matrix. The totals for each row and column in that matrix is 8, which is the number of tested functions.

All subsections show similarly calculated results summary. These summaries tell you whether or not using p < 1 benefits the various interpolation methods.

Results Summary

	Rank 1 Count	Rank 2 Count	Rank 3 Count
p = 1	0	0	8
p = 0.5	2	6	0
p = 0.1	6	2	0

The above table shows that using p = 0.1 and to a lesser exetent p = 0.5 rank better than regular Lagrange interpolation.

Implicit Shammas and Newton Divided-Difference Interpolation

The following matlab code performs the implicit Shammas interpolation using Newton Divided-Difference method.

```
clc
clear

ExcelFilename = 'newtonDivDiff.xlsx';
% test f(x) = log(x)
fprintf('---------------------------------- log(x)\n');
x=1:5;
y = log(x);

xint = 1.5:4.5;
yintc = log(xint);
fprintf('xint = \n');
disp(xint);
fprintf('yintc = \n');
disp(yintc);
yint = zeros(length(xint),1);
percerr = zeros(length(xint),1);
for i=1:length(xint)
  yint(i) = newtonDivDiff(x,y,xint(i));
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
```



```
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr');
yint1 = yint;
percerr1 = percerr;

p = 0.5;
for i=1:length(xint)
  yint(i) = newtonDivDiff(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint2 = yint;
percerr2 = percerr;

p = 0.1;
for i=1:length(xint)
  yint(i) = newtonDivDiff(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint3 = yint;
percerr3 = percerr;

n = length(xint);
X = x(1:n)';
Y = y(1:n)';
zVarNames =
{'X','Y','Xint','Yint','Yint1','PercnErr1','Yint2','PercnErr2','Yint3','Percn
Err3'};
T =
table(X,Y,xint',yintc',yint1,percerr1,yint2,percerr2,yint3,percerr3,'Variable
Names',zVarNames);
writetable(T, ExcelFilename, 'Sheet','log');

% test f(x) = log(x^2+1)
fprintf('---------------------------------- log(x^2+1)\n');
x=1:5;
y = log(x.^2+1);

xint = 1.5:4.5;
yintc = log(xint.^2+1);
```



```
fprintf('xint = \n');
disp(xint);
fprintf('yintc = \n');
disp(yintc);
yint = zeros(length(xint),1);
percerr = zeros(length(xint),1);
for i=1:length(xint)
  yint(i) = newtonDivDiff(x,y,xint(i));
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr');
yint1 = yint;
percerr1 = percerr;

p = 0.5;
for i=1:length(xint)
  yint(i) = newtonDivDiff(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint2 = yint;
percerr2 = percerr;

p = 0.1;
for i=1:length(xint)
  yint(i) = newtonDivDiff(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint3 = yint;
percerr3 = percerr;

n = length(xint);
X = x(1:n)';
Y = y(1:n)';
zVarNames =
{'X','Y','Xint','Yint','Yint1','PercnErr1','Yint2','PercnErr2','Yint3','Percn
Err3'};
```



```
T =
table(X,Y,xint',yintc',yint1,percerr1,yint2,percerr2,yint3,percerr3,'Variable
Names',zVarNames);
writetable(T, ExcelFilename, 'Sheet','log(x2 plus 1)');


% test f(x) = gamma(x)/gamma(x+1)
fprintf('---------------------------------- gamma(x)/gamma(x+1)\n');
x=2:6;
y = gamma(x)./gamma(x+1);

xint = 2.5:5.5;
yintc = gamma(xint)./gamma(xint+1);
fprintf('xint = \n');
disp(xint);
fprintf('yintc = \n');
disp(yintc);
yint = zeros(length(xint),1);
percerr = zeros(length(xint),1);
for i=1:length(xint)
  yint(i) = newtonDivDiff(x,y,xint(i));
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr');
yint1 = yint;
percerr1 = percerr;

p = 0.5;
for i=1:length(xint)
  yint(i) = newtonDivDiff(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint2 = yint;
percerr2 = percerr;

p = 0.1;
for i=1:length(xint)
  yint(i) = newtonDivDiff(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
```



```
fprintf('%%error = \n');
disp(percerr')
yint3 = yint;
percerr3 = percerr;

n = length(xint);
X = x(1:n)';
Y = y(1:n)';
zVarNames =
{'X','Y','Xint','Yint','Yint1','PercnErr1','Yint2','PercnErr2','Yint3','Percn
Err3'};
T =
table(X,Y,xint',yintc',yint1,percerr1,yint2,percerr2,yint3,percerr3,'Variable
Names',zVarNames);
writetable(T, ExcelFilename, 'Sheet','gamma ratio');



% test f(x) = 1/x
fprintf('---------------------------------- 1/x\n');
x=1:5;
y = 1./x;

xint = 1.5:4.5;
yintc = 1./xint;
fprintf('xint = \n');
disp(xint);
fprintf('yintc = \n');
disp(yintc);
yint = zeros(length(xint),1);
percerr = zeros(length(xint),1);
for i=1:length(xint)
  yint(i) = newtonDivDiff(x,y,xint(i));
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr');
yint1 = yint;
percerr1 = percerr;

p = 0.5;
for i=1:length(xint)
  yint(i) = newtonDivDiff(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
```



```
yint2 = yint;
percerr2 = percerr;

p = 0.1;
for i=1:length(xint)
  yint(i) = newtonDivDiff(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint3 = yint;
percerr3 = percerr;

n = length(xint);
X = x(1:n)';
Y = y(1:n)';
T =
table(X,Y,xint',yintc',yint1,percerr1,yint2,percerr2,yint3,percerr3,'Variable
Names',zVarNames);
writetable(T, ExcelFilename, 'Sheet','recip');

% test f(x) = sqrt(x)
fprintf('---------------------------------- sqrt(x)\n');
x=2:6;
y = sqrt(x);

xint = 2.5:5.5;
yintc = sqrt(xint);
fprintf('xint = \n');
disp(xint);
fprintf('yintc = \n');
disp(yintc);
yint = zeros(length(xint),1);
percerr = zeros(length(xint),1);
for i=1:length(xint)
  yint(i) = newtonDivDiff(x,y,xint(i));
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr');
yint1 = yint;
percerr1 = percerr;

p = 0.5;
for i=1:length(xint)
  yint(i) = newtonDivDiff(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
```



```
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint2 = yint;
percerr2 = percerr;

p = 0.1;
for i=1:length(xint)
  yint(i) = newtonDivDiff(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint3 = yint;
percerr3 = percerr;

n = length(xint);
X = x(1:n)';
Y = y(1:n)';
T =
table(X,Y,xint',yintc',yint1,percerr1,yint2,percerr2,yint3,percerr3,'Variable
Names',zVarNames);
writetable(T, ExcelFilename, 'Sheet','sqrt');


% test f(x) = Pade1(x)
fprintf('---------------------------------- Pade1(x)\n');
x=2:0.5:4;
y = pade1(x);

xint = 2.25:0.5:3.75;
yintc = pade1(xint);
fprintf('xint = \n');
disp(xint);
fprintf('yintc = \n');
disp(yintc);
yint = zeros(length(xint),1);
percerr = zeros(length(xint),1);
for i=1:length(xint)
  yint(i) = newtonDivDiff(x,y,xint(i));
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr');
```



```
yint1 = yint;
percerr1 = percerr;

p = 0.5;
for i=1:length(xint)
  yint(i) = newtonDivDiff(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint2 = yint;
percerr2 = percerr;

p = 0.1;
for i=1:length(xint)
  yint(i) = newtonDivDiff(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint3 = yint;
percerr3 = percerr;
n = length(xint);
X = x(1:n)';
Y = y(1:n)';
T =
table(X,Y,xint',yintc',yint1,percerr1,yint2,percerr2,yint3,percerr3,'Variable
Names',zVarNames);
writetable(T, ExcelFilename, 'Sheet','pade1');

% test f(x) = pade2(x))
fprintf('---------------------------------- pade2(x)\n');
x=2:0.5:4;
y = pade2(x);

xint = 2.25:0.5:3.75;
yintc = pade2(xint);
fprintf('xint = \n');
disp(xint);
fprintf('yintc = \n');
disp(yintc);
yint = zeros(length(xint),1);
percerr = zeros(length(xint),1);
for i=1:length(xint)
  yint(i) = newtonDivDiff(x,y,xint(i));
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
```



```
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr');
yint1 = yint;
percerr1 = percerr;

p = 0.5;
for i=1:length(xint)
  yint(i) = newtonDivDiff(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint2 = yint;
percerr2 = percerr;

p = 0.1;
for i=1:length(xint)
  yint(i) = newtonDivDiff(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint3 = yint;
percerr3 = percerr;
n = length(xint);
X = x(1:n)';
Y = y(1:n)';
T =
table(X,Y,xint',yintc',yint1,percerr1,yint2,percerr2,yint3,percerr3,'Variable
Names',zVarNames);
writetable(T, ExcelFilename, 'Sheet','pade2');

% test f(x) = trigFx1(x)
fprintf('----------------------------------- trigXf1(x)\n');
x=2:0.5:4;
y = trigFx1(x);

xint = 2.25:0.5:3.75;
yintc = trigFx1(xint);
fprintf('xint = \n');
disp(xint);
fprintf('yintc = \n');
disp(yintc);
```



```
yint = zeros(length(xint),1);
percerr = zeros(length(xint),1);
for i=1:length(xint)
  yint(i) = newtonDivDiff(x,y,xint(i));
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr');
yint1 = yint;
percerr1 = percerr;

p = 0.5;
for i=1:length(xint)
  yint(i) = newtonDivDiff(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint2 = yint;
percerr2 = percerr;

p = 0.1;
for i=1:length(xint)
  yint(i) = newtonDivDiff(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
fprintf('------------------ using power %f\n', p);
fprintf('xint = \n');
disp(xint);
fprintf('yint = \n');
disp(yint');
fprintf('%%error = \n');
disp(percerr')
yint3 = yint;
percerr3 = percerr;
n = length(xint);
X = x(1:n)';
Y = y(1:n)';
T =
table(X,Y,xint',yintc',yint1,percerr1,yint2,percerr2,yint3,percerr3,'Variable
Names',zVarNames);
writetable(T, ExcelFilename, 'Sheet','trig fx1');

function y = pade1(x)
  y = (x.^3-2*x.^2+3*x-4)./(x.^2+x+1);
end

function y = pade2(x)
```



```
  y = (x.^2+5*x+10)./(x.^3+x.^2+x+1);
end

function y = trigFx1(x)
  y = (5*x.^2 + sin(x))./(3*x.^2 + cos(x));
end
```

The above test code resembles the one for the Languagian version. The difference is in the loops that calculate arrays yint. These loops call function newtonDivDiff. Other than that, the test code performs the same tasks.

The code for Newton Divided-Different interpolation is:

```
function yint = newtonDivDiff(x,y,xint)
%NEWTONDIVDIFF Summary of this function goes here
%   Detailed explanation goes here
  n = length(x);
  D = zeros(n,n);

  for i = 1:n
    D(i,1) = y(i);
  end

  for i = 2:n
      for j = 2:i
         D(i,j)=(D(i,j-1)-D(i-1,j-1))/(x(i)-x(i-j+1));
      end
   end

   yint = D(n,n);
   for i = n-1:-1:1
      yint = yint*(xint-x(i)) + D(i,i);
   end
end
```


Results for ln(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0	1.5	0.40546511	0.39770936	-1.91280373	0.40470813	-0.18669292	0.40546337	-0.00042964
2	0.69314718	2.5	0.91629073	0.91848721	0.23971449	0.91643629	0.01588601	0.91629097	2.6319E-05
3	1.09861229	3.5	1.25276297	1.25111433	-0.13160005	1.25268106	-0.00653817	1.25276286	-8.6303E-06
4	1.38629436	4.5	1.5040774	1.50716249	0.20511513	1.50419881	0.00807197	1.50407753	8.9406E-06

Results for ln(x^2+1)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0.69314718	1.5	1.178655	1.18312495	0.37924183	1.181864	0.27225961	1.17784966	-0.06832647
2	1.60943791	2.5	1.98100147	1.98002516	-0.04928381	1.98032435	-0.03418067	1.9810479	0.002344
3	2.30258509	3.5	2.58399755	2.58457547	0.02236509	2.58438242	0.01489431	2.58399155	-0.00023244
4	2.83321334	4.5	3.0563569	3.05547513	-0.02885034	3.05579113	-0.01851123	3.05635376	-0.00010255

Results for 1/x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	1	1.5	0.666666667	0.684895833	2.734375	0.671777056	0.766558348	0.667423942	0.113591357
2	0.5	2.5	0.4	0.3953125	-1.171875	0.399104056	-0.223985955	0.399904072	-0.023982021
3	0.333333333	3.5	0.285714286	0.2890625	1.171875	0.286192723	0.167452956	0.285755013	0.014254402
4	0.25	4.5	0.222222222	0.216145833	-2.734375	0.22153751	-0.308120542	0.222173492	-0.021928423

Results for √x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.414213562	2.5	1.58113883	1.580836269	-0.01913566	1.58113883	0	1.581138627	-1.28307E-05
3	1.732050808	3.5	1.870828693	1.870930393	0.00543608	1.870828693	0	1.870828735	2.25044E-06
4	2	4.5	2.121320344	2.121236363	-0.003958879	2.121320344	0	2.12132032	-1.12032E-06
5	2.236067977	5.5	2.34520788	2.345375099	0.007130238	2.34520788	0	2.345207914	1.47025E-06

Results for pade1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.285714286	2.25	0.483082707	0.482827108	-0.052910023	0.482994794	-0.018198206	0.483083458	0.00015561
2.5	0.679487179	2.75	0.877071823	0.877160386	0.010097583	0.87709954	0.003160158	0.877072606	8.92452E-05
3	1.076923077	3.25	1.279535865	1.279462065	-0.00576774	1.279514811	-0.001645462	1.279534833	-8.06298E-05
3.5	1.485074627	3.75	1.693521595	1.693668505	0.008674842	1.693559922	0.002263188	1.693523943	0.000138669

Results for pade2 (x)

2	1.6	2.25	1.335448057	1.336574015	0.084313147	1.335576946	0.009651369	1.335330855	-0.008776272
2.5	1.133004926	2.75	0.975182482	0.974769466	-0.042352666	0.975134294	-0.004941386	0.975209492	0.002769717
3	0.85	3.25	0.749125596	0.749484524	0.047912958	0.749166015	0.005395446	0.749109839	-0.002103407
3.5	0.666666667	3.75	0.598383927	0.597645685	-0.123372497	0.598305402	-0.01312282	0.598406771	0.003817635

Results for Gamma Ratio

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.5	2.5	0.4	0.401822917	0.455729167	0.400462841	0.115710356	0.400063248	0.015811988
3	0.333333333	3.5	0.285714286	0.28515625	-0.1953125	0.285604889	-0.038288949	0.285702242	-0.004215261
4	0.25	4.5	0.222222222	0.22265625	0.1953125	0.222290958	0.030931148	0.22222861	0.002874599
5	0.2	5.5	0.181818182	0.180989583	-0.455729167	0.181708897	-0.060106915	0.181809353	-0.004855873

Results for TrigFx1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.805038197	2.25	1.792017881	1.791794392	-0.01247132	1.791982208	-0.001990663	1.792041648	0.001326305
2.5	1.774401191	2.75	1.754988474	1.755066517	0.004446901	1.754998071	0.000546828	1.754979836	-0.000492194
3	1.735528911	3.25	1.717123415	1.717057531	-0.003836847	1.717117085	-0.000368651	1.717130749	0.00042714
3.5	1.700452151	3.75	1.685909995	1.686043161	0.007898805	1.685920172	0.000603681	1.685895451	-0.000862647

Results Summary

	Rank 1 Count	Rank 2 Count	Rank 3 Count
p = 1	0	0	8
p = 0.5	2	6	0
p = 0.1	6	2	0

The above table shows that using p = 0.1 and to a lesser exetent p = 0.5 rank better than regular Newton divided-difference interpolation.

The Implicit Shammas and Barycentric Lagrange Interpolation

The following matlab code performs the implicit Shammas interpolation using Barycentric Lagrange method. The code, found in file test_barylag.m, is very similar to the two test scripts that I resented earlier. Here is a sample for loop that calculates the values for arrays yint and percerr:

```
p = 0.1;
for i=1:length(xint)
  [yint(i),~]= barylag(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
```

The code for function barylag is:

```
function [yi, u] = barylag(x, y, xi)

% BARYLAG 1-D barycentric lagrange interpolation
%    BARYLAG(X,Y,XI) interpolates to find YI, the values of the
%    underlying function Y at the points in the array XI, using the
%    second form (or true form) of the barycentric interpolation
%    formula.  X and Y must be vectors of length N.
%
%    [YI,U] = BARYLAG() also returns the barycentric interpolation
%    weights U.
%
%    Ref:
%    Barycentric Lagrange Interpolation
%    Jean-Paul Berrut and Lloyd N. Trefethen
%    SIAM Review, Vol. 46, No. 3
%    pgs. 501-517

% Joe Henning - Fall 2011

n = length(x);

if n ~= length(y)
   fprintf('??? Bad input to barylag ==> X and Y must be of the same
length\n');
   yi = [];
   u = [];
   return;
end

for i = 1:length(xi)
   % Find the right place in the table by means of a bisection.
   klo = 1;
   khi = n;
   while (khi-klo > 1)
      k = fix((khi+klo)/2.0);
      if (x(k) > xi(i))
```



```
         khi = k;
      else
         klo = k;
      end
   end

   h = x(khi) - x(klo);
   if (h == 0.0)
      fprintf('??? Bad input to barylag ==> X values must be distinct\n');
      yi(i) = NaN;
      continue;
   end

   isiny = 0;
   for k = 1:n
      if (xi(i) == x(k))
         yi(i) = y(k);
         isiny = 1;
         break
      end
   end

   if (isiny)
      continue
   end

   % Evaluate lagrange polynomial
   num = 0;
   den = 0;
   for k = 1:n
      u(k,1) = 1;
      for m = 1:n
         if (k ~= m)
            u(k,1) = u(k,1)/(x(k)-x(m));
         end
      end
      num = num + u(k,1)*y(k)/(xi(i)-x(k));
      den = den + u(k,1)/(xi(i)-x(k));
   end
   yi(i) = num/den;
end
```


Results for ln(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0	1.5	0.405465108	0.397709356	-1.912803726	0.404708133	-0.186692922	0.405463366	-0.00042964
2	0.693147181	2.5	0.916290732	0.918487214	0.239714487	0.916436294	0.01588601	0.916290973	2.6319E-05
3	1.098612289	3.5	1.252762968	1.251114332	-0.131600048	1.252681061	-0.006538173	1.25276286	-8.63028E-06
1	0	1.5	0.405465108	0.397709356	-1.912803726	0.404708133	-0.186692922	0.405463366	-0.00042964

Results for ln(x^2+1)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0.693147181	1.5	1.178654996	1.183124949	0.379241835	1.181863998	0.272259607	1.177849663	-0.068326475
2	1.609437912	2.5	1.981001469	1.980025156	-0.049283812	1.980324349	-0.034180666	1.981047903	0.002343996
3	2.302585093	3.5	2.583997552	2.584575466	0.022365088	2.584382421	0.014894306	2.583991546	-0.00023244
4	2.833213344	4.5	3.056356895	3.055475126	-0.028850339	3.055791126	-0.01851123	3.056353761	-0.000102554

Results for 1/x!!

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	1	1.5	0.666666667	0.684895833	2.734375	0.671777056	0.766558348	0.667423942	0.113591357
2	0.5	2.5	0.4	0.3953125	-1.171875	0.399104056	-0.223985955	0.399904072	-0.023982021
3	0.333333333	3.5	0.285714286	0.2890625	1.171875	0.286192723	0.167452956	0.285755013	0.014254402
4	0.25	4.5	0.222222222	0.216145833	-2.734375	0.22153751	-0.308120542	0.222173492	-0.021928423

Results for √x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.414213562	2.5	1.58113883	1.580836269	-0.01913566	1.58113883	-1.40433E-14	1.581138627	-1.28307E-05
3	1.732050808	3.5	1.870828693	1.870930393	0.00543608	1.870828693	0	1.870828735	2.25044E-06
4	2	4.5	2.121320344	2.121236363	-0.003958879	2.121320344	0	2.12132032	-1.12032E-06
5	2.236067977	5.5	2.34520788	2.345375099	0.007130238	2.34520788	1.8936E-14	2.345207914	1.47025E-06

Results for pade1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.285714286	2.25	0.483082707	0.482827108	-0.052910023	0.482994794	-0.018198206	0.483083458	0.00015561
2.5	0.679487179	2.75	0.877071823	0.877160386	0.010097583	0.87709954	0.003160158	0.877072606	8.92452E-05
3	1.076923077	3.25	1.279535865	1.279462065	-0.00576774	1.279514811	-0.001645462	1.279534833	-8.06298E-05
3.5	1.485074627	3.75	1.693521595	1.693668505	0.008674842	1.693559922	0.002263188	1.693523943	0.000138669

Results for pade2(x)

2	1.6	2.25	1.335448057	1.336574015	0.084313147	1.335576946	0.009651369	1.335330855	-0.008776272
2.5	1.133004926	2.75	0.975182482	0.974769466	-0.042352666	0.975134294	-0.004941386	0.975209492	0.002769717
3	0.85	3.25	0.749125596	0.749484524	0.047912958	0.749166015	0.005395446	0.749109839	-0.002103407
3.5	0.666666667	3.75	0.598383927	0.597645685	-0.123372497	0.598305402	-0.01312282	0.598406771	0.003817635

Results for Gamma Ratio

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.5	2.5	0.4	0.401822917	0.455729167	0.400462841	0.115710356	0.400063248	0.015811988
3	0.333333333	3.5	0.285714286	0.28515625	-0.1953125	0.285604889	-0.038288949	0.285702242	-0.004215261
4	0.25	4.5	0.222222222	0.22265625	0.1953125	0.222290958	0.030931148	0.22222861	0.002874599
5	0.2	5.5	0.181818182	0.180989583	-0.455729167	0.181708897	-0.060106915	0.181809353	-0.004855873

Results for TrigFx1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.5	2.5	0.4	0.401822917	0.455729167	0.400462841	0.115710356	0.400063248	0.015811988
3	0.333333333	3.5	0.285714286	0.28515625	-0.1953125	0.285604889	-0.038288949	0.285702242	-0.004215261
4	0.25	4.5	0.222222222	0.22265625	0.1953125	0.222290958	0.030931148	0.22222861	0.002874599
5	0.2	5.5	0.181818182	0.180989583	-0.455729167	0.181708897	-0.060106915	0.181809353	-0.004855873

Results Summary

	Rank 1 Count	Rank 2 Count	Rank 3 Count
p = 1	0	0	8
p = 0.5	2	6	0
p = 0.1	6	2	0

The above table shows that using p = 0.1 and to a lesser exetent p = 0.5 rank better than regular barycentric Lagrange interpolation.

The Implicit Shammas and Rational interpolation using the Bulirsch-Stoer Method

The following matlab code performs the implicit Shammas interpolation using the rational interpolation using the Bulirsch-Stoer Method. The code, found in file test_bulirschstoer.m, is very similar to the two test scripts that I resented earlier. Here is a sample for loop that calculates the values for arrays yint and percerr:

```
p = 0.1;
for i=1:length(xint)
  [yint(i),~]= bulirschstoer(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
```

The code for function bulirschstoer is:

```
function [yi, R] = bulirschstoer(x, y, xi)

% BULIRSCHSTOER Rational interpolation using the Bulirsch-Stoer Method
%    BULIRSCHSTOER(X,Y,XI) interpolates to find YI, the value of the
%    underlying function Y at the points in the array XI, using the
%    Bulirsch-Stoer Method.  X and Y must be vectors of length N.
%
%    [YI,R] = BULIRSCHSTOER() also returns the rational polynomial
%    table R calculated for the last XI.

% Joe Henning - Fall 2011

n = length(x);

if n ~= length(y)
   fprintf('??? Bad input to bulirschstoer ==> X and Y must be of the same
length\n');
   yi = [];
   R = [];
   return;
end

tol = n * max(y) * eps(class(y));

for k = 1:length(xi)
   isiny = 0;
   for i = 1:n
      if (xi(k) == x(i))
         yi(k) = y(i);
         isiny = 1;
         break
      end
   end

   if (isiny)
```

```
      continue
   end

   R = zeros(n,n);
   R(:,1) = y(:);

   for i = 1:n-1
      for j = 1:(n-i)
         if (i == 1)
            R(j,i+1) = R(j,i) + (R(j,i) - R(j+1,i))/(((xi(k)-x(j+i))/(xi(k)-
x(j)))*(1 - (R(j,i) - R(j+1,i))/(R(j,i)+tol)) - 1);
         else
            R(j,i+1) = R(j,i) + (R(j,i) - R(j+1,i))/(((xi(k)-x(j+i))/(xi(k)-
x(j)))*(1 - (R(j,i) - R(j+1,i))/(R(j,i) - R(j+1,i-1)+tol)) - 1);
         end
      end
   end

   yi(k) = R(1,n);
end
```


Results for ln(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0	1.5	0.405465108	0.404967467	-0.12273339	0.405432489	-0.008044914	0.405465055	-1.30735E-05
2	0.693147181	2.5	0.916290732	0.916356283	0.007153932	0.916294918	0.000456886	0.916290739	7.36123E-07
3	1.098612289	3.5	1.252762968	1.252735455	-0.002196196	1.252761199	-0.000141273	1.252762966	-2.28165E-07
4	1.386294361	4.5	1.504077397	1.504109417	0.002128904	1.504079494	0.00013943	1.5040774	2.2654E-07

Results for ln(x^2+1)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0.693147181	1.5	1.178654996	1.187636339	0.761999274	1.176633467	-0.17151157	1.177216676	-0.122030628
2	1.609437912	2.5	1.981001469	2.069947936	4.489974844	1.981187884	0.009410126	1.981133304	0.006654986
3	2.302585093	3.5	2.583997552	2.583205798	-0.030640678	2.583934905	-0.002424423	2.583954011	-0.001685029
4	2.833213344	4.5	3.056356895	3.056888455	0.01739195	3.056419537	0.002049541	3.056399711	0.001400872

Results for 1/x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	1	1.5	0.666666667	0.666666667	0	0.666666667	6.66134E-14	0.666750584	0.012587538
2	0.5	2.5	0.4	0.4	-1.38778E-14	0.4	-1.38778E-14	0.39999371	-0.001572397
3	0.333333333	3.5	0.285714286	0.285714286	0	0.285714286	1.94289E-14	0.285716199	0.000669724
4	0.25	4.5	0.222222222	0.222222222	1.249E-14	0.222222222	3.747E-14	0.222220426	-0.000808466

Results for √x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.414213562	2.5	1.58113883	1.581104127	-0.00219481	1.58113883	0	1.581138188	-4.06048E-05
3	1.732050808	3.5	1.870828693	1.870836656	0.000425643	1.870828693	0	1.870828838	7.73283E-06
4	2	4.5	2.121320344	2.121315573	-0.0002249	2.121320344	0	2.121320257	-4.1007E-06
5	2.236067977	5.5	2.34520788	2.34521507	0.000306602	2.34520788	0	2.345208013	5.66601E-06

Results for pade1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.285714286	2.25	0.483082707	0.48402054	0.194135034	0.482518896	-0.116711077	0.482668721	-0.085696564
2.5	0.679487179	2.75	0.877071823	0.876946788	-0.014256005	0.877425247	0.040295888	0.877265835	0.022120396
3	1.076923077	3.25	1.279535865	1.279597056	0.004782274	1.276316309	-0.251619083	1.279268443	-0.020899906
3.5	1.485074627	3.75	1.693521595	1.693438251	-0.004921298	1.69287265	-0.038319252	1.695382053	0.109857345

Results for pade2(x)

2	1.6	2.25	1.335448057	1.335461347	0.000995137	1.335484723	0.002745555	1.33549494	0.003510679
2.5	1.133004926	2.75	0.975182482	0.975180078	-0.000246471	0.975175793	-0.000685899	0.975173906	-0.000879373
3	0.85	3.25	0.749125596	0.749126735	0.000152064	0.749128809	0.00042884	0.749129733	0.000552189
3.5	0.666666667	3.75	0.598383927	0.598382547	-0.000230633	0.598379974	-0.000660577	0.598378811	-0.000854988

Results for Gamma Ratio

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.5	2.5	0.4	0.4	-1.38778E-14	0.4	-1.38778E-14	0.400006129	0.00153213
3	0.333333333	3.5	0.285714286	0.285714286	0	0.285714286	0	0.285713458	-0.00028974
4	0.25	4.5	0.222222222	0.222222222	1.249E-14	0.222222222	3.747E-14	0.222222564	0.000153864
5	0.2	5.5	0.181818182	0.181818182	0	0.181818182	-1.52656E-14	0.181817793	-0.0002137

Results for TrigFx1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.805038197	2.25	1.792017881	1.791692602	-0.018151513	1.791707391	-0.017326257	1.791712391	-0.017047236
2.5	1.774401191	2.75	1.754988474	1.755081713	0.005312758	1.755077264	0.005059266	1.755075758	0.004973446
3	1.735528911	3.25	1.717123415	1.717064098	-0.003454455	1.717066827	-0.003295517	1.717067758	-0.003241297
3.5	1.700452151	3.75	1.685909995	1.685996224	0.005114682	1.685992547	0.004896611	1.68599128	0.004821445

Results Summary

	Rank 1 Count	Rank 2 Count	Rank 3 Count
p = 1	4	0	4
p = 0.5	1	6	1
p = 0.1	3	2	3

The above table shows that the ordinary rational interpolation using the Bulirsch-Stoer method does better than using fractional powers with the x variables.

The Implicit Shammas and Rational interpolation using the Floater-Hormann Method

The following matlab code performs the implicit Shammas interpolation using the rational interpolation using the Floater-Hormann Method. The code, found in file test_floaterhormann.m, is very similar to the two test scripts that I resented earlier. Here is a sample for loop that calculates the values for arrays yint and percerr:

```
p = 0.1;
for i=1:length(xint)
  yint(i) = floaterhormann(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
```

The code for function floaterhormann is:

```
function [yi, u, ypi, yppi] = floaterhormann(x, y, xi, c)

% FLOATERHORMANN Rational interpolation using the Floater-Hormann Method
%    FLOATERHORMANN(X,Y,XI,C) interpolates to find YI, the values of
%    the underlying function Y at the points in the array XI, using
%    Floater-Hormann rational interpolation.  X and Y must be vectors
%    of length N.
%
%    C specifies the order of the interpolating polynomial (0 <= C <= N).
%    The default is C = 0 (Berrut rational interpolation). Use length(x)-1.
%
%    [YI,U] = FLOATERHORMANN() also returns the barycentric
%    interpolation weights U.
%
%    [YI,U,YPI,YPPI] = FLOATERHORMANN() also returns the interpolated
%    derivative YPI and the interpolated second derivative YPPI.
%
%    Ref:
%    Barycentric Rational Interpolation with no Poles and High Rates of
Approximation
%    Michael S. Floater and Kai Hormann
%    Ifl Technical Report Series, Ifl-06-06

% Joe Henning - Fall 2011

if (nargin < 4)
   c = 0;
end

n = length(x);

if n ~= length(y)
   fprintf('??? Bad input to floaterhormann ==> X and Y must be of the same
length\n');
   yi = [];
```

```
   u = [];
   ypi = [];
   yppi = [];
   return;
end

if (n-1 < c)
   fprintf('??? Bad input to floaterhormann ==> C <= N-1\n');
   yi = [];
   u = [];
   ypi = [];
   yppi = [];
   return
end

% calculate Floater-Hormann coefficients
u = [];
for k = 1:n
   u(k,1) = 0;
   for m = k-c:k
      prod = 1;
      if (m < 1 || m > n-c)
         continue
      end
      for j = m:m+c
         if (j ~= k)
            prod = prod/(x(k)-x(j));
         end
      end
      u(k,1) = u(k,1) + (-1)^m*prod;
   end
end

for i = 1:length(xi)
   % Find the right place in the table by means of a bisection.
   klo = 1;
   khi = n;
   while (khi-klo > 1)
      k = fix((khi+klo)/2.0);
      if (x(k) > xi(i))
         khi = k;
      else
         klo = k;
      end
   end

   h = x(khi) - x(klo);
   if (h == 0.0)
      fprintf('??? Bad input to floaterhormann ==> X values must be
distinct\n');
      yi(i) = NaN;
      ypi(i) = NaN;
      yppi(i) = NaN;
      continue;
   end

   isiny = 0;
```



```
   for k = 1:n
      if (xi(i) == x(k))
         yi(i) = y(k);
         num = 0;
         for j = 1:n
            if (j ~= k)
               num = num + u(j)*(y(k)/(x(k)-x(j)) + y(j)/(x(j)-x(k)));
            end
         end
         ypi(i) = -num/u(k);
         num = 0;
         for j = 1:n
            if (j ~= k)
               num = num + u(j)*(ypi(i)-(y(k)/(x(k)-x(j)) + y(j)/(x(j)-
x(k))))/(x(k)-x(j));
            end
         end
         yppi(i) = -2*num/u(k);
         isiny = 1;
         break
      end
   end

   if (isiny)
      continue
   end

   % Evaluate polynomial
   num = 0;
   den = 0;
   for k = 1:n
      num = num + y(k)*u(k)/(xi(i)-x(k));
      den = den + u(k)/(xi(i)-x(k));
   end
   yi(i) = num/den;

   num = 0;
   den = 0;
   for k = 1:n
      term = yi(i)/(xi(i)-x(k)) + y(k)/(x(k)-xi(i));
      num = num + term*u(k)/(xi(i)-x(k));
      den = den + u(k)/(xi(i)-x(k));
   end
   ypi(i) = num/den;

   num = 0;
   den = 0;
   for k = 1:n
      term = (ypi(i)-(yi(i)/(xi(i)-x(k)) + y(k)/(x(k)-xi(i))))/(xi(i)-x(k));
      num = num + term*u(k)/(xi(i)-x(k));
      den = den + u(k)/(xi(i)-x(k));
   end
   yppi(i) = 2*num/den;
end
```


Results for ln(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0	1.5	0.405465108	0.217125065	-46.45037006	0.209691977	-48.2835952	0.196595424	-51.51360263
2	0.693147181	2.5	0.916290732	1.077097129	17.54971339	1.054228766	15.05395931	1.041023499	13.61279371
3	1.098612289	3.5	1.252762968	1.120029317	-10.59527263	1.139349056	-9.053102226	1.151888689	-8.052144102
4	1.386294361	4.5	1.504077397	1.60583729	6.765602156	1.583446901	5.276956099	1.569272076	4.334529581

Results for ln(x^2+1)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0.693147181	1.5	1.178654996	0.949209882	-19.46668999	0.927446718	-21.31313057	0.897646948	-23.84141662
2	1.609437912	2.5	1.981001469	2.209688351	11.54400366	2.183923015	10.2433819	2.170605315	9.571110835
3	2.302585093	3.5	2.583997552	2.381684602	-7.829455964	2.406604007	-6.865081789	2.422049245	-6.26735531
4	2.833213344	4.5	3.056356895	3.217668772	5.277913628	3.18488496	4.20527017	3.164303957	3.531886651

Results for 1/x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	1	1.5	0.666666667	0.818139963	22.72099448	0.813683636	22.05254533	0.81284913	21.92736955
2	0.5	2.5	0.4	0.297826087	-25.54347826	0.318052588	-20.48685294	0.331030314	-17.24242141
3	0.333333333	3.5	0.285714286	0.358695652	25.54347826	0.344120517	20.44218104	0.334040009	16.91400318
4	0.25	4.5	0.222222222	0.171731123	-22.72099448	0.185274513	-16.62646919	0.193994789	-12.70234487

Results for √x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.414213562	2.5	1.58113883	1.49487757	-5.455641093	1.486843088	-5.963786362	1.478164882	-6.512644322
3	1.732050808	3.5	1.870828693	1.959905376	4.761348951	1.955952576	4.550062907	1.954087391	4.450364582
4	2	4.5	2.121320344	2.038085113	-3.9237464	2.042922851	-3.695693221	2.045861409	-3.55716829
5	2.236067977	5.5	2.34520788	2.41515704	2.982642187	2.406139185	2.598119591	2.400163009	2.343294558

Results for pade1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.285714286	2.25	0.483082707	0.366126988	-24.21028896	0.356392197	-26.22542842	0.347252386	-28.11740494
2.5	0.679487179	2.75	0.877071823	1.008978043	15.03938636	1.008799474	15.01902665	1.009394261	15.08684178
3	1.076923077	3.25	1.279535865	1.146341869	-10.40955548	1.148264891	-10.25926492	1.149176542	-10.18801631
3.5	1.485074627	3.75	1.693521595	1.813392512	7.07820422	1.805058216	6.586076103	1.799235492	6.242252712

Results for pade2(x)

2	1.6	2.25	1.335448057	1.453579393	8.845820363	1.456510318	9.065291657	1.459581321	9.295252143
2.5	1.133004926	2.75	0.975182482	0.871098694	-10.67326263	0.876810002	-10.08759708	0.880771548	-9.681360697
3	0.85	3.25	0.749125596	0.833873609	11.31292452	0.828244269	10.56146974	0.824159568	10.01620725
3.5	0.666666667	3.75	0.598383927	0.535174767	-10.56331172	0.542610648	-9.320651144	0.547907024	-8.435537934

Results for Gamma Ratio

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.5	2.5	0.4	0.444475138	11.11878453	0.445144819	11.28620478	0.446271241	11.56781033
3	0.333333333	3.5	0.285714286	0.25	-12.5	0.25390368	-11.13371186	0.256491294	-10.22804701
4	0.25	4.5	0.222222222	0.25	12.5	0.246681613	11.00672571	0.244347616	9.95642699
5	0.2	5.5	0.181818182	0.16160221	-11.11878453	0.165313691	-9.077470082	0.167821087	-7.698402296

Results for TrigFx1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.805038197	2.25	1.792017881	1.799457157	0.415134064	1.800257122	0.459774507	1.801004324	0.501470625
2.5	1.774401191	2.75	1.754988474	1.745329326	-0.550382433	1.745363339	-0.548444351	1.745332749	-0.550187373
3	1.735528911	3.25	1.717123415	1.726711597	0.558386311	1.726438751	0.542496623	1.726269296	0.532628085
3.5	1.700452151	3.75	1.685909995	1.678057629	-0.465764244	1.678768543	-0.42359628	1.67926857	-0.393937082

Results Summary

	Rank 1 Count	Rank 2 Count	Rank 3 Count
p = 1	0	0	8
p = 0.5	1	7	0
p = 0.1	7	1	0

The above table shows that using p = 0.1 benefits the rational interpolation using the Floater-Hormann method.

The Implicit Shammas and Hermite divided difference interpolation

The following matlab code performs the implicit Shammas interpolation using the Hermite divided difference interpolation. The code, found in file test_hermitediv.m, is remarkably similar to the two test scripts that I resented earlier. Here is a sample for loop that calculates the values for arrays yint and percerr:

```
p = 0.5;
for i=1:length(xint)
  yint(i) = hermitediv(x.^p,y,[],[],xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
```

The code for function hermitediv is:

```
function [yi, p, pval] = hermitediv(x, y, yp, ypp, xi)

% HERMITEDIV Hermite divided difference interpolation
%    HERMITEDIV(X,Y,YP,YPP,XI) interpolates to find YI, the values of
%    the underlying function Y at the points in the array XI, using
%    divided difference hermite interpolation.  X and Y must be
%    vectors of length N.  YP can be used to specify the first
%    derivative of Y, and YPP can be used to specify the second
%    derivative.
%
%    [YI,P,PVAL] = HERMITEDIV() also returns P, the coefficients of
%    the interpolating Hermite polynomial calculated via divided
%    difference, and PVAL, which specifies the maximum degree of the
%    interpolating polynomial.
%
%    See also: DIVDIFF

% Joe Henning - Fall 2011

if (nargin < 5)
   help hermitediv;
   yi = NaN;
   p = NaN;
   pval = NaN;
   return
end

if (isempty(yp) && isempty(ypp))
   order = 1;
   yp = [];
   ypp = [];
   pval = length(x)*(0+1) - 1;
elseif (isempty(ypp))
   order = 2;
   ypp = [];
```



```
   pval = length(x)*(1+1) - 1;
else
   order = 3;
   pval = length(x)*(2+1) - 1;
end

x = reshape(repmat(x,order,1),1,length(x)*order);
y = reshape(repmat(y,order,1),1,length(y)*order);
yp = reshape(repmat(yp,order,1),1,length(yp)*order);
ypp = reshape(repmat(ypp,order,1),1,length(ypp)*order);

n = length(x)-1;
D = zeros(n+1,n+1);
D(:,1) = y(:);

tol = n * max(x) * eps(class(x));

if (order == 1)
   for i = 1:n
      for j = 1:i
         D(i+1,j+1) = (D(i+1,j)-D(i,j))/(x(i+1)-x(i-j+1));
      end
   end
elseif (order == 2)
   for i = 1:n
      for j = 1:i
         h = (x(i+1)-x(i-j+1));
         if (j == 1 && h < tol)
            D(i+1,j+1) = yp(i)/1;
         else
            D(i+1,j+1) = (D(i+1,j)-D(i,j))/h;
         end
      end
   end
else
   for i = 1:n
      for j = 1:i
         h = (x(i+1)-x(i-j+1));
         if (j == 1 && h < tol)
            D(i+1,j+1) = yp(i)/1.0;
         elseif (j == 2 && h < tol)
            D(i+1,j+1) = ypp(i)/2.0;
         else
            D(i+1,j+1) = (D(i+1,j)-D(i,j))/h;
         end
      end
   end
end

p = diag(D);

for k = 1:length(xi)
   % Evaluate polynomial with the difference coefficients
   yi(k) = 0;
   for i = 1:length(p)
      term = p(i);
      for j = 1:i
```



```
         if (j == 1)
            term = term*1;
         else
            term = term*(xi(k)-x(j-1));
         end
      end
      yi(k) = yi(k) + term;
   end
end
```


Results for ln(x)

1	0	1.5	0.405465108	0.397709356	-1.912803726	0.404708133	-0.186692922	0.405463366	-0.00042964
2	0.693147181	2.5	0.916290732	0.918487214	0.239714487	0.916436294	0.01588601	0.916290973	2.6319E-05
3	1.098612289	3.5	1.252762968	1.251114332	-0.131600048	1.252681061	-0.006538173	1.25276286	-8.63028E-06
4	1.386294361	4.5	1.504077397	1.507162487	0.205115127	1.504198805	0.00807197	1.504077531	8.94061E-06

Results for ln(x^2+1)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0.693147181	1.5	1.178654996	1.183124949	0.379241835	1.181863998	0.272259607	1.177849663	-0.068326475
2	1.609437912	2.5	1.981001469	1.980025156	-0.049283812	1.980324349	-0.034180666	1.981047903	0.002343996
3	2.302585093	3.5	2.583997552	2.584575466	0.022365088	2.584382421	0.014894306	2.583991546	-0.00023244
4	2.833213344	4.5	3.056356895	3.055475126	-0.028850339	3.055791126	-0.01851123	3.056353761	-0.000102554

Results for 1/x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	1	1.5	0.666666667	0.684895833	2.734375	0.671777056	0.766558348	0.667423942	0.113591357
2	0.5	2.5	0.4	0.3953125	-1.171875	0.399104056	-0.223985955	0.399904072	-0.023982021
3	0.333333333	3.5	0.285714286	0.2890625	1.171875	0.286192723	0.167452956	0.285755013	0.014254402
4	0.25	4.5	0.222222222	0.216145833	-2.734375	0.22153751	-0.308120542	0.222173492	-0.021928423

Results for √x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.414213562	2.5	1.58113883	1.580836269	-0.01913566	1.58113883	0	1.581138627	-1.28307E-05
3	1.732050808	3.5	1.870828693	1.870930393	0.00543608	1.870828693	0	1.870828735	2.25044E-06
4	2	4.5	2.121320344	2.121236363	-0.003958879	2.121320344	0	2.12132032	-1.12032E-06
5	2.236067977	5.5	2.34520788	2.345375099	0.007130238	2.34520788	0	2.345207914	1.47025E-06

Results for pade1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.285714286	2.25	0.483082707	0.482827108	-0.052910023	0.482994794	-0.018198206	0.483083458	0.00015561
2.5	0.679487179	2.75	0.877071823	0.877160386	0.010097583	0.87709954	0.003160158	0.877072606	8.92452E-05
3	1.076923077	3.25	1.279535865	1.279462065	-0.00576774	1.279514811	-0.001645462	1.279534833	-8.06298E-05
3.5	1.485074627	3.75	1.693521595	1.693668505	0.008674842	1.693559922	0.002263188	1.693523943	0.000138669

Results for pade2(x)

2	1.6	2.25	1.335448057	1.336574015	0.084313147	1.335576946	0.009651369	1.335330855	-0.008776272
2.5	1.133004926	2.75	0.975182482	0.974769466	-0.042352666	0.975134294	-0.004941386	0.975209492	0.002769717
3	0.85	3.25	0.749125596	0.749484524	0.047912958	0.749166015	0.005395446	0.749109839	-0.002103407
3.5	0.666666667	3.75	0.598383927	0.597645685	-0.123372497	0.598305402	-0.01312282	0.598406771	0.003817635

Results for Gamma Ratio

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.5	2.5	0.4	0.401822917	0.455729167	0.400462841	0.115710356	0.400063248	0.015811988
3	0.333333333	3.5	0.285714286	0.28515625	-0.1953125	0.285604889	-0.038288949	0.285702242	-0.004215261
4	0.25	4.5	0.222222222	0.22265625	0.1953125	0.222290958	0.030931148	0.22222861	0.002874599
5	0.2	5.5	0.181818182	0.180989583	-0.455729167	0.181708897	-0.060106915	0.181809353	-0.004855873

Results for TrigFx1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.805038197	2.25	1.792017881	1.791794392	-0.01247132	1.791982208	-0.001990663	1.792041648	0.001326305
2.5	1.774401191	2.75	1.754988474	1.755066517	0.004446901	1.754998071	0.000546828	1.754979836	-0.000492194
3	1.735528911	3.25	1.717123415	1.717057531	-0.003836847	1.717117085	-0.000368651	1.717130749	0.00042714
3.5	1.700452151	3.75	1.685909995	1.686043161	0.007898805	1.685920172	0.000603681	1.685895451	-0.000862647

Results Summary

	Rank 1 Count	Rank 2 Count	Rank 3 Count
p = 1	0	0	8
p = 0.5	2	6	0
p = 0.1	6	2	0

The above table shows that using p = 0.1 (and to a lesser extent p = 0.5) benefits the Hermite divided difference interpolation.

The Implicit Shammas and Neville Interpolation

The following matlab code performs the implicit Shammas interpolation using the Neville interpolation method. The code, found in file test_nevilleInterpol.m, is very similar to the two test scripts that I resented earlier. Here is a sample for loop that calculates the values for arrays yint and percerr:

```
p = 0.1;
for i=1:length(xint)
  yint(i) = nevilleInterpol(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
```

The code for function nevilleInterpol is:

```
function yint = nevilleInterpol(x,y,xint)
%NEVILLE Summary of this function goes here
% Neville's algorithm as a function
%
% inputs:
%    n = order of interpolation (n+1 = # of points)
%    x(1),...,x(n+1)     x coords
%    y(1),...,y(n+1)     yint coords
%    xint=evaluation point for interpolating polynomial p
%
% output:  yint
  n = length(x)-1;
  for i = n:-1:1
    for j = 1:i
        y(j) = (xint-x(j))*y(j+1) - (xint-x(j+n+1-i))*y(j);
        y(j) = y(j)/(x(j+n+1-i)-x(j));
    end
  end
  yint = y(1);
end
```


Results for ln(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0	1.5	0.405465108	0.397709356	-1.912803726	0.404708133	-0.186692922	0.405463366	-0.00042964
2	0.693147181	2.5	0.916290732	0.918487214	0.239714487	0.916436294	0.01588601	0.916290973	2.6319E-05
3	1.098612289	3.5	1.252762968	1.251114332	-0.131600048	1.252681061	-0.006538173	1.25276286	-8.63028E-06
4	1.386294361	4.5	1.504077397	1.507162487	0.205115127	1.504198805	0.00807197	1.504077531	8.94061E-06

Results for ln(x^2+1)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0.693147181	1.5	1.178654996	1.183124949	0.379241835	1.181863998	0.272259607	1.177849663	-0.068326475
2	1.609437912	2.5	1.981001469	1.980025156	-0.049283812	1.980324349	-0.034180666	1.981047903	0.002343996
3	2.302585093	3.5	2.583997552	2.584575466	0.022365088	2.584382421	0.014894306	2.583991546	-0.00023244
4	2.833213344	4.5	3.056356895	3.055475126	-0.028850339	3.055791126	-0.01851123	3.056353761	-0.000102554

Results for 1/x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	1	1.5	0.666666667	0.684895833	2.734375	0.671777056	0.766558348	0.667423942	0.113591357
2	0.5	2.5	0.4	0.3953125	-1.171875	0.399104056	-0.223985955	0.399904072	-0.023982021
3	0.333333333	3.5	0.285714286	0.2890625	1.171875	0.286192723	0.167452956	0.285755013	0.014254402
4	0.25	4.5	0.222222222	0.216145833	-2.734375	0.22153751	-0.308120542	0.222173492	-0.021928423

Results for √x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.414213562	2.5	1.58113883	1.580836269	-0.01913566	1.58113883	1.40433E-14	1.581138627	-1.28307E-05
3	1.732050808	3.5	1.870828693	1.870930393	0.00543608	1.870828693	1.18688E-14	1.870828735	2.25044E-06
4	2	4.5	2.121320344	2.121236363	-0.003958879	2.121320344	0	2.12132032	-1.12032E-06
5	2.236067977	5.5	2.34520788	2.345375099	0.007130238	2.34520788	-3.78721E-14	2.345207914	1.47025E-06

Results for pade1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.285714286	2.25	0.483082707	0.482827108	-0.052910023	0.482994794	-0.018198206	0.483083458	0.00015561
2.5	0.679487179	2.75	0.877071823	0.877160386	0.010097583	0.87709954	0.003160158	0.877072606	8.92452E-05
3	1.076923077	3.25	1.279535865	1.279462065	-0.00576774	1.279514811	-0.001645462	1.279534833	-8.06298E-05
3.5	1.485074627	3.75	1.693521595	1.693668505	0.008674842	1.693559922	0.002263188	1.693523943	0.000138669

Results for pade2(x)

2	1.6	2.25	1.335448057	1.336574015	0.084313147	1.335576946	0.009651369	1.335330855	-0.008776272
2.5	1.133004926	2.75	0.975182482	0.974769466	-0.042352666	0.975134294	-0.004941386	0.975209492	0.002769717
3	0.85	3.25	0.749125596	0.749484524	0.047912958	0.749166015	0.005395446	0.749109839	-0.002103407
3.5	0.666666667	3.75	0.598383927	0.597645685	-0.123372497	0.598305402	-0.01312282	0.598406771	0.003817635

Results for Gamma Ratio

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.5	2.5	0.4	0.401822917	0.455729167	0.400462841	0.115710356	0.400063248	0.015811988
3	0.333333333	3.5	0.285714286	0.28515625	-0.1953125	0.285604889	-0.038288949	0.285702242	-0.004215261
4	0.25	4.5	0.222222222	0.22265625	0.1953125	0.222290958	0.030931148	0.22222861	0.002874599
5	0.2	5.5	0.181818182	0.180989583	-0.455729167	0.181708897	-0.060106915	0.181809353	-0.004855873

Results for TrigFx1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.805038197	2.25	1.792017881	1.791794392	-0.01247132	1.791982208	-0.001990663	1.792041648	0.001326305
2.5	1.774401191	2.75	1.754988474	1.755066517	0.004446901	1.754998071	0.000546828	1.754979836	-0.000492194
3	1.735528911	3.25	1.717123415	1.717057531	-0.003836847	1.717117085	-0.000368651	1.717130749	0.00042714
3.5	1.700452151	3.75	1.685909995	1.686043161	0.007898805	1.685920172	0.000603681	1.685895451	-0.000862647

Results Summary

	Rank 1 Count	Rank 2 Count	Rank 3 Count
p = 1	0	0	8
p = 0.5	2	6	0
p = 0.1	6	2	0

The above table shows that using p = 0.1 (and to a lesser extent p = 0.5) benefits the Neville interpolation.

The Implicit Shammas and Interpolation using simple rational polynomials

The following matlab code performs the implicit Shammas interpolation with interpolation using simple rational polynomials. The code, found in file test_ratint.m, is remarkably similar to the two test scripts that I resented earlier. Here is a sample for loop that calculates the values for arrays yint and percerr:

```
p = 0.1;
for i=1:length(xint)
  [yint(i),~]= ratint(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
```

The code for function ratint is:

```
function [yi, R] = ratint(x, y, xi)

% RATINT Interpolation using simple rational polynomials
%    RATINT(X,Y,XI) interpolates to find YI, the value of the
%    underlying function Y at the points in the array XI, using
%    rational polynomials.  X and Y must be vectors of length N.
%
%    [YI,R] = RATINT() also returns the rational polynomial table R
%    calculated for the last XI.

% Joe Henning - Fall 2011

n = length(x);

if n ~= length(y)
   fprintf('??? Bad input to ratint ==> X and Y must be of the same
length\n');
   yi = [];
   ypi = [];
   return;
end

tol = n * max(y) * eps(class(y));

for k = 1:length(xi)
   isiny = 0;
   for i = 1:n
      if (xi(k) == x(i))
         yi(k) = y(i);
         isiny = 1;
         break
      end
   end

   if (isiny)
      continue
   end
```



```
   R = zeros(n,n);
   R(:,1) = y(:);

   % Evaluate rational polynomial
   for i = 1:n-1
      for j = 1:(n-i)
         R(j,i+1) = (x(j+i)-x(j))/((xi(k)-x(j))/(R(j+1,i)+tol) + (x(j+i)-
xi(k))/(R(j,i)+tol));
      end
   end

   yi(k) = R(1,n);
end
```


Results for ln(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.693147181	2.5	0.916290732	0.901152595	-1.652110697	0.908379056	-0.863446033	0.912129889	-0.454096379
3	1.098612289	3.5	1.252762968	1.260217255	0.595027721	1.255639783	0.229637545	1.253929881	0.093147109
4	1.386294361	4.5	1.504077397	1.496478826	-0.505198151	1.501745334	-0.155049385	1.503297791	-0.051832834
5	1.609437912	5.5	1.704748092	1.722753521	1.056192947	1.70922078	0.262366487	1.706018557	0.074525075

Results for ln(x^2+1)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0.693147181	1.5	1.178654996	1.108909304	-5.917396733	1.14872214	-2.539577444	1.168448674	-0.865929625
2	1.609437912	2.5	1.981001469	2.031269335	2.537497671	1.994875302	0.700344436	1.984342253	0.168641191
3	2.302585093	3.5	2.583997552	2.528114467	-2.162660149	2.572147425	-0.458596682	2.581722834	-0.088030974
4	2.833213344	4.5	3.056356895	3.204838084	4.858110282	3.079638269	0.761736093	3.060030885	0.120208146

Results for 1/x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	1	1.5	0.666666667	0.666666667	6.66134E-13	0.666666667	7.32747E-13	0.666901651	0.035247645
2	0.5	2.5	0.4	0.4	1.09635E-12	0.4	1.08247E-12	0.399987278	-0.003180508
3	0.333333333	3.5	0.285714286	0.285714286	1.57374E-12	0.285714286	1.57374E-12	0.285717365	0.001077721
4	0.25	4.5	0.222222222	0.222222222	2.02338E-12	0.222222222	2.03587E-12	0.222219803	-0.001088452

Results for √x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.414213562	2.5	1.58113883	1.57910617	-0.128556698	1.580785312	-0.022358472	1.58112043	-0.001163706
3	1.732050808	3.5	1.870828693	1.871729357	0.048142521	1.870949317	0.00644763	1.870833745	0.000270011
4	2	4.5	2.121320344	2.120402791	-0.043253858	2.121220933	-0.004686237	2.121316827	-0.000165758
5	2.236067977	5.5	2.34520788	2.34738051	0.092641241	2.345403919	0.008359145	2.345213913	0.000257249

Results for pade1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.285714286	2.25	0.483082707	0.45752357	-5.290840692	0.463278064	-4.099638107	0.467275279	-3.272199
2.5	0.679487179	2.75	0.877071823	0.898365531	2.427817967	0.89048166	1.528932581	0.886108177	1.030286633
3	1.076923077	3.25	1.279535865	1.249823312	-2.322135249	1.26331692	-1.267564704	1.269862663	-0.755993033
3.5	1.485074627	3.75	1.693521595	1.793875218	5.925736258	1.73897746	2.684103077	1.717261748	1.401821683

Results for pade2(x)

2	1.6	2.25	1.335448057	1.335445761	-0.00017192	1.335466578	0.001386892	1.33543141	-0.001246535
2.5	1.133004926	2.75	0.975182482	0.975183463	0.000100633	0.97517902	-0.000354962	0.975184845	0.0002423
3	0.85	3.25	0.749125596	0.749124832	-0.000101969	0.749127287	0.000225655	0.749124687	-0.000121411
3.5	0.666666667	3.75	0.598383927	0.598385252	0.000221452	0.59838182	-0.000351987	0.598384844	0.000153278

Results for Gamma Ratio

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.5	2.5	0.4	0.4	5.68989E-13	0.4	5.41234E-13	0.400015366	0.003841391
3	0.333333333	3.5	0.285714286	0.285714286	7.96585E-13	0.285714286	7.77156E-13	0.285712614	-0.000585076
4	0.25	4.5	0.222222222	0.222222222	1.01169E-12	0.222222222	1.04916E-12	0.222222805	0.000262344
5	0.2	5.5	0.181818182	0.181818182	1.22125E-12	0.181818182	1.23651E-12	0.181817606	-0.000316862

Results for TrigFx1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.805038197	2.25	1.792017881	1.791831135	-0.01042098	1.791989013	-0.001610899	1.792009344	-0.000476365
2.5	1.774401191	2.75	1.754988474	1.7550485	0.003420283	1.754993166	0.000267354	1.754986864	-9.17882E-05
3	1.735528911	3.25	1.717123415	1.717076236	-0.002747538	1.717122423	-5.77554E-05	1.71712713	0.000216331
3.5	1.700452151	3.75	1.685909995	1.686000296	0.005356227	1.685908167	-0.000108438	1.685899692	-0.000611123

Results Summary

	Rank 1 Count	Rank 2 Count	Rank 3 Count
p = 1	2	1	5
p = 0.5	2	5	1
p = 0.1	4	2	2

The above table shows that using p = 0.1 (and to a lesser extent p = 0.5) somewhat benefits the rational polynomials interpolation.

The Implicit Shammas and Rational interpolation using the Schneider-Werner Method

The following matlab code performs the implicit Shammas interpolation using the rational interpolation using the Schneider-Werner Method. The code, found in file test_schwerner.m, is very similar to the two test scripts that I resented earlier. Here is a sample for loop that calculates the values for arrays yint and percerr:

```
p = 0.1;
for i=1:length(xint)
  yint(i) = schwerner(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
```

The code for function schwerner is:

```
function [yi, u, ypi, yppi] = schwerner(x, y, xi, p, q)

% SCHWERNER Rational interpolation using the Schneider-Werner Method
%    SCHWERNER(X,Y,XI,P,Q) interpolates to find YI, the values of the
%    underlying function Y at the points in the array XI, using
%    the Schneider Werner method and the second form (or true form)
%    of the barycentric interpolation formula.  X and Y must be
%    vectors of length N.
%
%    P and Q specify the degrees of the numerator and denominator
%    of the rational function and must satisfy P <= Q and P+Q=N-1.
%    If not specified, values for P and Q are calculated.
%
%    [YI,U] = SCHWERNER() also returns the barycentric interpolation
%    weights U.
%
%    [YI,U,YPI,YPPI] = SCHWERNER() also returns the interpolated
%    derivative YPI and the interpolated second derivative YPPI.
%
%    Ref:
%    Recent developments in barycentric rational interpolation
%    Jean-Paul Berrut, Richard Baltensperger and Hans D. Mittelmann
%    Trends and Applications in Constructive Approximation
%    International Series of Numerical Mathematics Vol. 1??
%    2005 Birkhauser Verlag Basel (ISBN 3-7643-7124-2)

% Joe Henning - Fall 2011

x = x(:).';
y = y(:).';
n = length(x);

if n ~= length(y)
   fprintf('??? Bad input to schwerner ==> X and Y must be of the same
length\n');
```



```
   yi = [];
   u = [];
   ypi = [];
   yppi = [];
   return;
end

if (nargin < 4)
   p = floor((n-1)/2.0);
   q = ceil((n-1)/2.0);
elseif (nargin < 5)
   q = (n-1) - p;
end

if (p > q)
   fprintf('??? Bad input to schwerner ==> P <= Q\n');
   yi = [];
   u = [];
   ypi = [];
   yppi = [];
   return
end

if ((p+q) ~= (n-1))
   fprintf('??? Bad input to schwerner ==> P+Q = (N-1)\n');
   yi = [];
   u = [];
   ypi = [];
   yppi = [];
   return
end

if (p < 0)
   fprintf('??? Bad input to schwerner ==> P >= 0\n');
   yi = [];
   u = [];
   ypi = [];
   yppi = [];
   return
end

% construct A
A = ones(1,n);
for i = 1:p-1
   A = [A; x.^i];
end
A = [A; y];
for i = 1:q-1
   A = [A; y.*x.^i];
end

% calculate the kernel of A
%u = null(A);
u = kernel(A);

% check the dimension of the kernel
if (size(u,2) > 1)
```



```
   fprintf('The kernel of A has a dimension larger than 1; recalling with
P=%d-1=%d\n',p,p-1);
   [yi, u, ypi, yppi] = schwerner(x, y, xi, p-1);
   return
end

% remove unattainable points
[N,M] = size(u);
tol = reshape(eps*max([ones(size(u(:))) abs(u(:))],[],2),N,M);
i = find(abs(u) < tol);
if ~isempty(i)
   for k = 1:length(i)
      fprintf('(%f,%f) is an unattainable point; it will be
eliminated\n',x(i(k)),y(i(k)));
   end
   k = find(abs(u) >= tol);
   [yi, u, ypi, yppi] = schwerner(x(k), y(k), xi);
   return
end

for i = 1:length(xi)
   % Find the right place in the table by means of a bisection.
   klo = 1;
   khi = n;
   while (khi-klo > 1)
      k = fix((khi+klo)/2.0);
      if (x(k) > xi(i))
         khi = k;
      else
         klo = k;
      end
   end

   h = x(khi) - x(klo);
   if (h == 0.0)
      fprintf('??? Bad input to schwerner ==> X values must be distinct\n');
      yi(i) = NaN;
      ypi(i) = NaN;
      yppi(i) = NaN;
      continue;
   end

   isiny = 0;
   for k = 1:n
      if (xi(i) == x(k))
         yi(i) = y(k);
         num = 0;
         for j = 1:n
            if (j ~= k)
               num = num + u(j)*(y(k)/(x(k)-x(j)) + y(j)/(x(j)-x(k)));
            end
         end
         ypi(i) = -num/u(k);
         num = 0;
         for j = 1:n
            if (j ~= k)
```

```
                num = num + u(j)*(ypi(i)-(y(k)/(x(k)-x(j)) + y(j)/(x(j)-
x(k))))/(x(k)-x(j));
             end
          end
          yppi(i) = -2*num/u(k);
          isiny = 1;
          break
       end
    end

    if (isiny)
       continue
    end

    % Evaluate polynomial
    num = 0;
    den = 0;
    for k = 1:n
       num = num + y(k)*u(k)/(xi(i)-x(k));
       den = den + u(k)/(xi(i)-x(k));
    end
    yi(i) = num/den;

    num = 0;
    den = 0;
    for k = 1:n
       term = yi(i)/(xi(i)-x(k)) + y(k)/(x(k)-xi(i));
       num = num + term*u(k)/(xi(i)-x(k));
       den = den + u(k)/(xi(i)-x(k));
    end
    ypi(i) = num/den;

    num = 0;
    den = 0;
    for k = 1:n
       term = (ypi(i)-(yi(i)/(xi(i)-x(k)) + y(k)/(x(k)-xi(i))))/(xi(i)-x(k));
       num = num + term*u(k)/(xi(i)-x(k));
       den = den + u(k)/(xi(i)-x(k));
    end
    yppi(i) = 2*num/den;
end


function Y = kernel(X)
% Calculate the kernel or null space of matrix X
[m,n] = size(X);
[U,S,V] = svd(X);
s = [];
for i = 1:m
   s = [s; S(i,i)];
end
%tol = eps*max([ones(size(s)) abs(s)],[],2);
tol = eps*max([max(m,n) max(abs(s))]);
r = sum(s > tol);
Y = V(:,r+1:n);
```


Results for ln(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0	1.5	0.405465108	0.404967467	-0.12273339	0.405432489	-0.008044914	0.405465055	-1.30735E-05
2	0.693147181	2.5	0.916290732	0.916356283	0.007153932	0.916294918	0.000456886	0.916290739	7.36123E-07
3	1.098612289	3.5	1.252762968	1.252735455	-0.002196196	1.252761199	-0.000141273	1.252762966	-2.28165E-07
4	1.386294361	4.5	1.504077397	1.504109417	0.002128904	1.504079494	0.00013943	1.5040774	2.2654E-07

Results for ln(x^2+1)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0.693147181	1.5	1.178654996	1.187636339	0.761999274	1.176633467	-0.17151157	1.177216676	-0.122030628
2	1.609437912	2.5	1.981001469	2.069947936	4.489974842	1.981187884	0.009410126	1.981133304	0.006654986
3	2.302585093	3.5	2.583997552	2.583205798	-0.030640678	2.583934905	-0.002424423	2.583954011	-0.001685029
4	2.833213344	4.5	3.056356895	3.056888455	0.01739195	3.056419537	0.002049541	3.056399711	0.001400872

Results for 1/x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	1	1.5	0.666666667	0.666666667	-2.66454E-13	0.666666667	1.33227E-13	0.666750584	0.012587538
2	0.5	2.5	0.4	0.4	5.55112E-14	0.4	-2.77556E-14	0.39999371	-0.001572397
3	0.333333333	3.5	0.285714286	0.285714286	-3.88578E-14	0.285714286	0	0.285716199	0.000669724
4	0.25	4.5	0.222222222	0.222222222	8.74301E-14	0.222222222	2.498E-14	0.222220426	-0.000808466

Results for √x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.414213562	2.5	1.58113883	1.581104127	-0.00219481	1.58113883	0	1.581138188	-4.06048E-05
3	1.732050808	3.5	1.870828693	1.870836656	0.000425643	1.870828693	-1.18688E-14	1.870828838	7.73283E-06
4	2	4.5	2.121320344	2.121315573	-0.0002249	2.121320344	-2.09346E-14	2.121320257	-4.1007E-06
5	2.236067977	5.5	2.34520788	2.34521507	0.000306602	2.34520788	0	2.345208013	5.66601E-06

Results for pade1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.285714286	2.25	0.483082707	0.48402054	0.194135034	0.482518896	-0.116711077	0.482668721	-0.085696564
2.5	0.679487179	2.75	0.877071823	0.876946788	-0.014256005	0.877425247	0.040295888	0.877265835	0.022120396
3	1.076923077	3.25	1.279535865	1.279597056	0.004782274	1.276316309	-0.251619083	1.279268443	-0.020899906
3.5	1.485074627	3.75	1.693521595	1.693438251	-0.004921298	1.69287265	-0.038319252	1.695382053	0.109857345

Results for pade2(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.6	2.25	1.335448057	1.335461347	0.000995137	1.335484723	0.002745555	1.33549494	0.003510679
2.5	1.133004926	2.75	0.975182482	0.975180078	-0.000246471	0.975175793	-0.000685899	0.975173906	-0.000879373
3	0.85	3.25	0.749125596	0.749126735	0.000152064	0.749128809	0.00042884	0.749129733	0.000552189
3.5	0.666666667	3.75	0.598383927	0.598382547	-0.000230633	0.598379974	-0.000660577	0.598378811	-0.000854988

Results for Gamma Ratio

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.5	2.5	0.4	0.4	9.71445E-14	0.4	-2.77556E-14	0.400006129	0.00153213
3	0.333333333	3.5	0.285714286	0.285714286	-1.94289E-14	0.285714286	-1.94289E-14	0.285713458	-0.00028974
4	0.25	4.5	0.222222222	0.222222222	1.249E-14	0.222222222	3.747E-14	0.222222564	0.000153864
5	0.2	5.5	0.181818182	0.181818182	-4.57967E-14	0.181818182	-3.05311E-14	0.181817793	-0.0002137

Results for TrigFx1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.805038197	2.25	1.792017881	1.791692602	-0.018151513	1.791707391	-0.017326257	1.791712391	-0.017047236
2.5	1.774401191	2.75	1.754988474	1.755081713	0.005312758	1.755077264	0.005059266	1.755075758	0.004973446
3	1.735528911	3.25	1.717123415	1.717064098	-0.003454455	1.717066827	-0.003295517	1.717067758	-0.003241297
3.5	1.700452151	3.75	1.685909995	1.685996224	0.005114682	1.685992547	0.004896611	1.68599128	0.004821445

Results Summary

	Rank 1 Count	Rank 2 Count	Rank 3 Count
p = 1	2	2	4
p = 0.5	3	4	1
p = 0.1	3	2	3

The above table shows that using p = 0.1 and p = 0.5 somewhat benefit the Schneider-Werner Method.

The Implicit Shammas and Steffen interpolation

The following matlab code performs the implicit Shammas interpolation using the Steffen interpolation. The code, found in file test_steffen.m, is very similar to the two test scripts that I resented earlier. Here is a sample for loop that calculates the values for arrays yint and percerr:

```
p = 0.1;
for i=1:length(xint)
  yint(i) = steffen(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
```

The code for function steffen is:

```
function yi = steffen(x, y, xi, yp)

% STEFFEN 1-D Steffen interpolation
%    STEFFEN(X,Y,XI) interpolates to find YI, the values of the
%    underlying function Y at the points in the array XI, using the
%    method of Steffen.  X and Y must be vectors of length N.
%
%    Steffen's method is based on a third-order polynomial.  The
%    slope at each grid point is calculated in a way to guarantee
%    a monotonic behavior of the interpolating function.  As such,
%    the curve is smooth up to the first derivative.
%
%    Ref:
%    A Simple Method for Monotonic Interpolation in One Dimension
%    M. Steffen
%    Astron. Astrophys. 239, , 1990
%    pgs. 443-450

% Joe Henning - Summer 2014

if nargin < 4
   yp = [];
end

n = length(x);

if n ~= length(y)
   fprintf('??? Bad input to steffen ==> X and Y must be of the same
length\n');
   yi = [];
   return;
end

if (isempty(yp))
   % calculate slopes
   yp = zeros(1,n);
```



```
   for i = 2:n-1
      hi = x(i+1) - x(i);
      him1 = x(i) - x(i-1);
      si = (y(i+1) - y(i))/hi;
      sim1 = (y(i) - y(i-1))/him1;
      pi = (sim1*hi + si*him1)/(him1 + hi);

      if (sim1*si <= 0)
         yp(i) = 0;
      elseif (abs(pi) > 2*abs(sim1))
         if (sign(sim1) ~= sign(si))
            error('Sign si not equal to sim1');
         end
         a = sign(sim1);
         yp(i) = 2*a*min([abs(sim1),abs(si)]);
      elseif (abs(pi) > 2*abs(si))
         if (sign(sim1) ~= sign(si))
            error('Sign si not equal to sim1');
         end
         a = sign(sim1);
         yp(i) = 2*a*min([abs(sim1),abs(si)]);
      else
         yp(i) = pi;
      end
   end

   % first point
   h1 = x(2) - x(1);
   h2 = x(3) - x(2);
   s1 = (y(2) - y(1))/h1;
   s2 = (y(3) - y(2))/h2;
   p1 = s1*(1 + h1/(h1 + h2)) - s2*h1/(h1 + h2);
   if (p1*s1 <= 0)
      yp(1) = 0;
   elseif (abs(p1) > 2*abs(s1))
      yp(1) = 2*s1;
   else
      yp(1) = p1;
   end

   % last point
   hnm1 = x(n) - x(n-1);
   hnm2 = x(n-1) - x(n-2);
   snm1 = (y(n) - y(n-1))/hnm1;
   snm2 = (y(n-1) - y(n-2))/hnm2;
   pn = snm1*(1 + hnm1/(hnm1 + hnm2)) - snm2*hnm1/(hnm1 + hnm2);
   if (pn*snm1 <= 0)
      yp(n) = 0;
   elseif (abs(pn) > 2*abs(snm1))
      yp(n) = 2*snm1;
   else
      yp(n) = pn;
   end
end

for i = 1:length(xi)
```



```
   % Find the right place in the table by means of a bisection.
   klo = 1;
   khi = n;
   while (khi-klo > 1)
      k = fix((khi+klo)/2.0);
      if (x(k) > xi(i))
         khi = k;
      else
         klo = k;
      end
   end

   h = x(khi) - x(klo);
   if (h == 0.0)
      fprintf('??? Bad x input to steffen ==> x values must be distinct\n');
      yi(i) = NaN;
      continue;
   end

   isiny = 0;
   for k = 1:n
      if (xi(i) == x(k))
         yi(i) = y(k);
         isiny = 1;
         break
      end
   end

   if (isiny)
      continue
   end

   s = (y(khi) - y(klo))/h;

   a = (yp(klo) + yp(khi) - 2*s)/h/h;
   b = (3*s - 2*yp(klo) - yp(khi))/h;
   c = yp(klo);
   d = y(klo);

   t = xi(i) - x(klo);
   t2 = t*t;
   t3 = t2*t;

   yi(i) = a*t3 + b*t2 + c*t + d;
end
```


Results for ln(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0	1.5	0.405465108	0.382533849	-5.655544291	0.399223064	-1.539477587	0.405199498	-0.065507603
2	0.693147181	2.5	0.916290732	0.921221304	0.538101263	0.917155366	0.094362457	0.916314206	0.002561914
3	1.098612289	3.5	1.252762968	1.253848422	0.086644778	1.252956251	0.015428473	1.252768273	0.000423463
4	1.386294361	4.5	1.504077397	1.505933452	0.123401571	1.504518847	0.029350222	1.504094345	0.001126799

Results for ln(x^2+1)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0.693147181	1.5	1.178654996	1.17918549	0.045008427	1.192130651	1.14330779	1.190413955	0.997659061
2	1.609437912	2.5	1.981001469	1.980115408	-0.044727938	1.978512118	-0.125661255	1.979675709	-0.0669237
3	2.302585093	3.5	2.583997552	2.584665718	0.025857813	2.58369397	-0.011748572	2.583709682	-0.011140518
4	2.833213344	4.5	3.056356895	3.058873073	0.082326047	3.056555271	0.006490606	3.056006736	-0.011456762

Results for 1/x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	1	1.5	0.666666667	0.708333333	6.25	0.688995766	3.349364905	0.676985274	1.5477911
2	0.5	2.5	0.4	0.390625	-2.34375	0.396494025	-0.876493796	0.398838945	-0.290263873
3	0.333333333	3.5	0.285714286	0.284375	-0.46875	0.285197712	-0.180800803	0.285538621	-0.061482775
4	0.25	4.5	0.222222222	0.220833333	-0.625	0.221563959	-0.296218432	0.221945053	-0.124725934

Results for √x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.414213562	2.5	1.58113883	1.579368192	-0.111985013	1.58113883	0	1.580822949	-0.019978105
3	1.732050808	3.5	1.870828693	1.871135983	0.016425322	1.870828693	0	1.870848259	0.001045816
4	2	4.5	2.121320344	2.121441953	0.005732721	2.121320344	0	2.121328103	0.000365772
5	2.236067977	5.5	2.34520788	2.345609637	0.017130966	2.34520788	0	2.345267844	0.002556884

Results for pade1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.285714286	2.25	0.483082707	0.482142857	-0.194552529	0.482069168	-0.209806514	0.481194261	-0.390915677
2.5	0.679487179	2.75	0.877071823	0.877306462	0.02675254	0.877237492	0.018888903	0.877189572	0.013425234
3	1.076923077	3.25	1.279535865	1.279608141	0.005648582	1.279602826	0.005233254	1.279595885	0.004690783
3.5	1.485074627	3.75	1.693521595	1.6934763	-0.002674596	1.693624596	0.006082061	1.694090791	0.033610205

Results for pade2(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.6	2.25	1.335448057	1.343503695	0.603216085	1.339802329	0.326053241	1.337453851	0.150196297
2.5	1.133004926	2.75	0.975182482	0.973773604	-0.144473215	0.974644918	-0.05512439	0.975043055	-0.014297501
3	0.85	3.25	0.749125596	0.748488663	-0.08502359	0.748853103	-0.036374804	0.749026712	-0.013199961
3.5	0.666666667	3.75	0.598383927	0.596691176	-0.282886969	0.597434771	-0.158619912	0.597876174	-0.084854007

Results for Gamma Ratio

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.5	2.5	0.4	0.40625	1.5625	0.403292314	0.823078427	0.40150682	0.37670506
3	0.333333333	3.5	0.285714286	0.284375	-0.46875	0.285197712	-0.180800803	0.285538621	-0.061482775
4	0.25	4.5	0.222222222	0.221875	-0.15625	0.222086833	-0.060925015	0.222175765	-0.020905575
5	0.2	5.5	0.181818182	0.18125	-0.3125	0.181545676	-0.149878091	0.181702293	-0.063738675

Results for TrigFx1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.805038197	2.25	1.792017881	1.790749103	-0.070801608	1.790914774	-0.061556688	1.791125783	-0.049781716
2.5	1.774401191	2.75	1.754988474	1.755242536	0.014476527	1.755121911	0.007603298	1.755028305	0.002269581
3	1.735528911	3.25	1.717123415	1.71723355	0.006413946	1.717196268	0.004242742	1.71715693	0.001951814
3.5	1.700452151	3.75	1.685909995	1.686032338	0.007256809	1.686139245	0.013598039	1.686195293	0.01692248

Results Summary

	Rank 1 Count	Rank 2 Count	Rank 3 Count
p = 1	1	0	7
p = 0.5	2	6	0
p = 0.1	5	2	1

The above table shows that using p = 0.1 (and to a lesser extent p = 0.5) benefits the Steffen interpolation.

The Implicit Shammas and Stineman interpolation

The following matlab code performs the implicit Shammas interpolation using the Stineman interpolation. The code, found in file test_ stineman.m, is very similar to the two test scripts that I resented earlier. Here is a sample for loop that calculates the values for arrays yint and percerr:

```
p = 0.1;
for i=1:length(xint)
  yint(i) = stineman(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
```

The code for function stineman is:

```
function yi = stineman(x, y, xi, yp)

% STINEMAN 1-D Stineman interpolation
%    STINEMAN(X,Y,XI) interpolates to find YI, the values of the
%    underlying function Y at the points in the array XI, using the
%    method of Stineman.  X and Y must be vectors of length N.
%
%    Stineman's method is based on a rational function which satisfies
%    three conditions outlined in his paper.  The method produces
%    monotonic curves and avoids problems seen in other interpolation
%    methods near abrupt changes of slope.  The curve is smooth up to
%    the first derivative.
%
%    Ref:
%    A Consistently Well-Behaved Method of Interpolation
%    Russel W. Stineman
%    Creative Computing, Volume 6, Number 7, 1980
%    pgs. 54-57

% Joe Henning - Summer 2014

if nargin < 4
   yp = [];
end

n = length(x);

if n ~= length(y)
   fprintf('??? Bad input to stineman ==> X and Y must be of the same
length\n');
   yi = [];
   return;
end

if (isempty(yp))
   % calculate slopes
```



```
   yp = zeros(1,n);

   for i = 2:n-1
      a = y(i) - y(i-1);
      b = y(i+1) - y(i);
      c = x(i) - x(i-1);
      d = x(i+1) - x(i);
      yp(i) = (a*(d*d + b*b) + b*(c*c + a*a))/(c*(d*d + b*b) + d*(c*c +
a*a));
   end

   % first point
   s = (y(2)-y(1))/(x(2)-x(1));
   if (s > 0 & s > yp(2))
      yp(1) = 2*s - yp(2);
   elseif (s < 0 & s < yp(2))
      yp(1) = 2*s - yp(2);
   else
      yp(1) = s + abs(s)*(s - yp(2))/(abs(s) + abs(s - yp(2)));
   end

   % last point
   s = (y(n)-y(n-1))/(x(n)-x(n-1));
   if (s > 0 & s > yp(n-1))
      yp(n) = 2*s - yp(n-1);
   elseif (s < 0 & s < yp(n-1))
      yp(n) = 2*s - yp(n-1);
   else
      yp(n) = s + abs(s)*(s - yp(n-1))/(abs(s) + abs(s - yp(n-1)));
   end
end

for i = 1:length(xi)
   % Find the right place in the table by means of a bisection.
   klo = 1;
   khi = n;
   while (khi-klo > 1)
      k = fix((khi+klo)/2.0);
      if (x(k) > xi(i))
         khi = k;
      else
         klo = k;
      end
   end

   h = x(khi) - x(klo);
   if (h == 0.0)
      fprintf('??? Bad input to stineman ==> X values must be distinct\n');
      yi(i) = NaN;
      continue;
   end

   isiny = 0;
   for k = 1:n
      if (xi(i) == x(k))
         yi(i) = y(k);
         isiny = 1;
```



```
         break
      end
   end

   if (isiny)
      continue
   end

   slo = (y(khi)-y(klo))/(x(khi)-x(klo));

   % ordinate corresponding to xi(i)
   y0 = y(klo) + slo*(xi(i) - x(klo));

   % vertical distances
   dylo = y(klo) + yp(klo)*(xi(i) - x(klo)) - y0;
   dyhi = y(khi) + yp(khi)*(xi(i) - x(khi)) - y0;

   % test product
   prod = dylo*dyhi;
   if (abs(prod) < eps)
      yi(i) = y0;
   elseif (prod > 0)
      yi(i) = y0 + prod/(dylo + dyhi);
   else   % prod < 0
      yi(i) = y0 + prod*(xi(i) - x(klo) + xi(i) - x(khi))/((dylo -
dyhi)*(x(khi) - x(klo)));
   end
end
```


Results for ln(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0	1.5	0.405465108	0.386830976	-4.595742343	0.4028026	-0.656655331	0.405427207	-0.009347645
2	0.693147181	2.5	0.916290732	0.916472213	0.019806036	0.916204638	-0.009395927	0.916280204	-0.00114892
3	1.098612289	3.5	1.252762968	1.252840129	0.006159195	1.252756643	-0.00050495	1.252760586	-0.00019017
4	1.386294361	4.5	1.504077397	1.505280893	0.080015567	1.504180499	0.006854819	1.504072906	-0.000298601

Results for ln(x^2+1)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0.693147181	1.5	1.178654996	1.182202727	0.300998237	1.192149279	1.144888237	1.196575421	1.520413091
2	1.609437912	2.5	1.981001469	1.979235341	-0.089153274	1.975457893	-0.279837033	1.981367548	0.018479513
3	2.302585093	3.5	2.583997552	2.583808094	-0.007331998	2.583613333	-0.01486919	2.584277992	0.010852929
4	2.833213344	4.5	3.056356895	3.057616891	0.041225411	3.056062425	-0.009634694	3.056004696	-0.011523517

Results for 1/x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	1	1.5	0.666666667	0.704268293	5.640243902	0.678509968	1.776495172	0.664436906	-0.334464157
2	0.5	2.5	0.4	0.400222222	0.055555556	0.400618228	0.154557031	0.401849974	0.462493505
3	0.333333333	3.5	0.285714286	0.285722485	0.002869605	0.28570621	-0.002826436	0.285997829	0.099240234
4	0.25	4.5	0.222222222	0.221435344	-0.354095057	0.221898604	-0.145628269	0.222468149	0.110666913

Results for √x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.414213562	2.5	1.58113883	1.579452064	-0.106680454	1.58113883	0	1.581801142	0.041888284
3	1.732050808	3.5	1.870828693	1.870884541	0.002985203	1.870828693	0	1.870790736	-0.002028887
4	2	4.5	2.121320344	2.121343066	0.001071166	2.121320344	0	2.121305256	-0.000711249
5	2.236067977	5.5	2.34520788	2.345523469	0.013456743	2.34520788	0	2.345102649	-0.004487046

Results for pade1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.285714286	2.25	0.483082707	0.482145544	-0.193996305	0.482755623	-0.067707569	0.483557691	0.098323617
2.5	0.679487179	2.75	0.877071823	0.8775227	0.051407087	0.877287659	0.024608702	0.877309591	0.02710925
3	1.076923077	3.25	1.279535865	1.279610236	0.005812315	1.279608219	0.005654699	1.27963684	0.007891502
3.5	1.485074627	3.75	1.693521595	1.69345996	-0.003639467	1.693176645	-0.02036878	1.692898454	-0.036795539

Results for pade2(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.6	2.25	1.335448057	1.339527555	0.305477825	1.333705904	-0.130454586	1.332732875	-0.203316158
2.5	1.133004926	2.75	0.975182482	0.975776911	0.060955743	0.976169271	0.101190229	0.976043358	0.088278503
3	0.85	3.25	0.749125596	0.749274962	0.019938652	0.749471241	0.046139828	0.749490882	0.048761581
3.5	0.666666667	3.75	0.598383927	0.597627944	-0.126337359	0.598695653	0.052094658	0.599082962	0.116820623

Results for Gamma Ratio

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.5	2.5	0.4	0.406143345	1.535836177	0.402665434	0.666358459	0.399457804	-0.135549124
3	0.333333333	3.5	0.285714286	0.285722485	0.002869605	0.28570621	-0.002826436	0.285997829	0.099240234
4	0.25	4.5	0.222222222	0.222222992	0.00034626	0.222210384	-0.005327396	0.222291192	0.031036373
5	0.2	5.5	0.181818182	0.181482623	-0.184557555	0.181670151	-0.081416748	0.181907361	0.049048306

Results for TrigFx1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.805038197	2.25	1.792017881	1.790683253	-0.074476265	1.790743631	-0.071106972	1.790397683	-0.090411909
2.5	1.774401191	2.75	1.754988474	1.754965051	-0.001334668	1.754518674	-0.026769424	1.754597728	-0.02226491
3	1.735528911	3.25	1.717123415	1.717339049	0.012557864	1.717504455	0.022190593	1.717371951	0.014473985
3.5	1.700452151	3.75	1.685909995	1.686108194	0.011756223	1.686182125	0.016141438	1.686263578	0.020972815

Results Summary

	Rank 1 Count	Rank 2 Count	Rank 3 Count
p = 1	1	1	6
p = 0.5	3	5	0
p = 0.1	4	2	2

The above table shows that using p = 0.1 and p = 0.5 somewhat benefit the Stineman Method.

The Implicit Shammas and Interpolation using Thiele rational polynomial

The following matlab code performs the implicit Shammas interpolation with Interpolation using Thiele rational polynomial. The code, found in file test_ thiele.m, is very similar to the two test scripts that I resented earlier. Here is a sample for loop that calculates the values for arrays yint and percerr:

```
p = 0.1;
for i=1:length(xint)
  yint(i) = thiele(x.^p,y,xint(i)^p);
  percerr(i) = 100*(yint(i) - yintc(i))/yintc(i);
end
```

The code for function thiele is:

```
function [yi, T] = thiele(x, y, xi, p)
% THIELE Interpolation using rational polynomials
%    THIELE(X,Y,XI,P) interpolates to find YI, the values of the
%    underlying function Y at the points in the array XI, using
%    rational polynomials.  X and Y must be vectors of length N.
%
%    P specifies the degree of the numerator.  It defaults to a
%    value of 1 (Thiele interpolation).
%
%    [YI,T] = THIELE() also returns the rational polynomial
%    table T calculated for the last XI.
%
%    Ref:
%    Some Techniques for Rational Interpolation
%    F. M. Larkin
%    The Computer Journal (1967), 10 (2)
%    pgs. 178-187
% Joe Henning - Fall 2014
  if (nargin < 4)
    p = 1;
  end
  n = length(x);
  if n ~= length(y)
    fprintf('??? Bad input to thiele ==> X and Y must be of the same
length\n');
    yi = [];
    T = [];
    return;
  end
  if (p < 1 || p >= (n-1))
    fprintf('??? Bad input to thiele ==> 1 <= P < (N-1)\n');
    yi = NaN;
    T = [];
    return
  end
```



```
  q = (n - 1) - p;
  tol = n * max(y) * eps(class(y));
  for k = 1:length(xi)
    isiny = 0;
    for i = 1:n
      if (xi(k) == x(i))
        yi(k) = y(i);
        isiny = 1;
        break
      end
    end
    if (isiny)
      continue
    end
    T = zeros(n,n);
    T(:,1) = y(:);
    % Evaluate rational polynomial
    for i = 1:n-1
      for j = 1:(n-i)
        if (i <= p)
          T(j,i+1) = ((xi(k)-x(j))*T(j+1,i) + (x(j+i)-xi(k))*T(j,i))/(x(j+i)-
x(j));
        else
          T(j,i+1) = T(j+1,i-1) + (x(j+i)-x(j))/((xi(k)-x(j))/(T(j+1,i)-
T(j+1,i-1)+tol) + (x(j+i)-xi(k))/(T(j,i)-T(j+1,i-1)+tol));
        end
      end
    end
    yi(k) = T(1,n);
  end

end
```


Results for ln(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0	1.5	0.405465108	0.404967467	-0.12273339	0.405432489	-0.008044914	0.405465055	-1.30735E-05
2	0.693147181	2.5	0.916290732	0.916356283	0.007153932	0.916294918	0.000456886	0.916290739	7.36124E-07
3	1.098612289	3.5	1.252762968	1.252735455	-0.002196196	1.252761199	-0.000141273	1.252762966	-2.28164E-07
4	1.386294361	4.5	1.504077397	1.504109417	0.002128904	1.504079494	0.00013943	1.5040774	2.26541E-07

Results for ln(x^2+1)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	0.693147181	1.5	1.178654996	1.187636339	0.761999274	1.176633467	-0.17151157	1.177216676	-0.122030628
2	1.609437912	2.5	1.981001469	2.069947937	4.489974846	1.981187884	0.009410126	1.981133304	0.006654986
3	2.302585093	3.5	2.583997552	2.583205798	-0.030640678	2.583934905	-0.002424423	2.583954011	-0.001685029
4	2.833213344	4.5	3.056356895	3.056888455	0.01739195	3.056419537	0.002049541	3.056399711	0.001400872

Results for 1/x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
1	1	1.5	0.666666667	0.666666667	5.4956E-13	0.666666667	6.16174E-13	0.666750584	0.012587538
2	0.5	2.5	0.4	0.4	8.60423E-13	0.4	8.46545E-13	0.39999371	-0.001572397
3	0.333333333	3.5	0.285714286	0.285714286	1.18516E-12	0.285714286	1.22402E-12	0.285716199	0.000669724
4	0.25	4.5	0.222222222	0.222222222	1.51129E-12	0.222222222	1.54876E-12	0.222220426	-0.000808466

Results for √x

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.414213562	2.5	1.58113883	1.581104127	-0.00219481	1.58113883	5.19603E-13	1.581138188	-4.06048E-05
3	1.732050808	3.5	1.870828693	1.870836656	0.000425643	1.870828693	4.27276E-13	1.870828838	7.73283E-06
4	2	4.5	2.121320344	2.121315573	-0.0002249	2.121320344	3.76822E-13	2.121320257	-4.1007E-06
5	2.236067977	5.5	2.34520788	2.34521507	0.000306602	2.34520788	3.21912E-13	2.345208013	5.66601E-06

Results for pade1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.285714286	2.25	0.483082707	0.48402054	0.194135034	0.482518896	-0.116711077	0.482668721	-0.085696564
2.5	0.679487179	2.75	0.877071823	0.876946788	-0.014256005	0.877425247	0.040295888	0.877265835	0.022120396
3	1.076923077	3.25	1.279535865	1.279597056	0.004782274	1.276316309	-0.251619083	1.279268443	-0.020899906
3.5	1.485074627	3.75	1.693521595	1.693438251	-0.004921298	1.69287265	-0.038319252	1.695382053	0.109857345

Results for pade2(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.6	2.25	1.335448057	1.335461347	0.000995137	1.335484723	0.002745555	1.33549494	0.003510679
2.5	1.133004926	2.75	0.975182482	0.975180078	-0.000246471	0.975175793	-0.000685899	0.975173906	-0.000879373
3	0.85	3.25	0.749125596	0.749126735	0.000152064	0.749128809	0.00042884	0.749129733	0.000552189
3.5	0.666666667	3.75	0.598383927	0.598382547	-0.000230633	0.598379974	-0.000660577	0.598378811	-0.000854988

Results for Gamma Ratio

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	0.5	2.5	0.4	0.4	4.16334E-13	0.4	4.02456E-13	0.400006129	0.00153213
3	0.333333333	3.5	0.285714286	0.285714286	5.82867E-13	0.285714286	6.02296E-13	0.285713458	-0.00028974
4	0.25	4.5	0.222222222	0.222222222	7.74381E-13	0.222222222	7.99361E-13	0.222222564	0.000153864
5	0.2	5.5	0.181818182	0.181818182	9.312E-13	0.181818182	9.312E-13	0.181817793	-0.0002137

Results for TrigFx1(x)

X	Y	Xint	Yint	Yint1	PercnErr1	Yint2	PercnErr2	Yint3	PercnErr3
2	1.805038197	2.25	1.792017881	1.791692602	-0.018151513	1.791707391	-0.017326257	1.791712391	-0.017047236
2.5	1.774401191	2.75	1.754988474	1.755081713	0.005312758	1.755077264	0.005059266	1.755075758	0.004973446
3	1.735528911	3.25	1.717123415	1.717064098	-0.003454455	1.717066827	-0.003295517	1.717067758	-0.003241297
3.5	1.700452151	3.75	1.685909995	1.685996224	0.005114682	1.685992547	0.004896611	1.68599128	0.004821445

Results Summary

	Rank 1 Count	Rank 2 Count	Rank 3 Count
p = 1	4	0	4
p = 0.5	1	6	1
p = 0.1	3	2	3

The above table shows the the regular Thiele rational polynomial interpolation is generally better than the ones using p < 1.

CONCLUSIONS

Raising the x variables to powers less than 1 seems to benefit the interpolation methods presented in this study, with the exception of the Bulirsch-Stoer and Thiele methods. Using fractional powers (the smaller, the better in general) tend to linearize the curves connecting the (x^p, y) data points.

The implicit Shammas interpolation scheme seems to work well with the three most popular interpolation algorithms—Lagrange, Newton divided-difference, and Neville.

DOCUMENT HISTORY

Date	*Version*	*Comments*
11/21/2020	1.00.00	Initial release.
12/2/2020	1.01/00	Edited text to include the linearization effect of the transformation of x.

