Polynomial Representations for a Wavelet Model of Interest Rates

Dennis G. Llemit
Department of Mathematics
Adamson University
1000 San Marcelino Street, Ermita, Manila

Abstract

We provide two numerical schemes in order to generate polynomial functions that approximate a new wavelet which is a modification of Morlet and Mexican Hat wavelets. This new wavelet fits historical data of Philippine 90-day T-bill rates from 1987 up to 2008 and is used as a new tool for interest rates forecasting.

1 Introduction

A wavelet is a mathematical function that is used to divide a given function or a continuous-time signal into different scale components. Since wavelet transforms are representations of functions similar to those of Fourier transforms, they are used in data compression and signal processing. Over the years, wavelets have been employed in research undertakings in a broad range of fields from electrical engineering and computer science to economics and finance. Among the popular ones are the Morlet and the Mexican Hat wavelets.

In 2010, Noemi Torre and Jose Maria Escaner IV, developed a new wavelet which we call the Torre - Escaner Wavelet. The said wavelet was constructed using the Morlet and Mexican Hat wavelets and logically inherits their intrinsic properties. The Morlet wavelet is known to offer improved detection and localization of scale over the Mexican Hat wavelet. On the other hand, the Mexican Hat wavelet provides better detection and localization of patch and gap events over the Morlet wavelet.

The Torre - Escaner wavelet is given by

$$W(t) = \frac{2}{\sqrt{3}}\pi^{-\frac{1}{4}}\left(1-(1.1t)^2\right)e^{-\frac{(1.1t)^2}{2}}e^{i7.7t} \tag{1}$$

where t stands for time and $W(t)$ for frequency.
This new wavelet fits historical data of Philippine 90-day T-bill rates and is, thus, used to model Philippine interest rates [4].

In this paper, we aim to

(1) represent the Torre - Escaner wavelet by a polynomial function $P_n(t)$ via numerical methods; and

(2) determine the root-mean-squared errors, $E_{RMS} = \frac{1}{\sqrt{n}}\|W(t)-P_n(t)\|_2$, of these polynomial representations.

We are interested in polynomial functions as approximations for the Torre - Escaner wavelet for three reasons. Firstly, polynomial functions are generally simple and easy to manipulate. Secondly, polynomial functions are algorithmically easy to implement and computationally efficient. Note that these two are the reasons why we only want a polynomial approximation of practical degree. Although a plethora of numerical schemes can easily give an approximation for (1), they are mostly higher ordered that they become impractical for our goal. Thirdly, we want to provide a numerical basis or treatment for the wavelet that models Philippine interest rates.

2 Numerical Schemes

We employ techniques and methods provided in Numerical Analysis such as Polynomial Least Squares (PLS) and Chebyshev Polynomial approximations to construct the approximating or interpolating function $P_n(t)$.

2.1 Polynomial Least Squares Approximation

For the Polynomial Least Squares (PLS) approximation, we seek to construct a degree n polynomial,

$$P_n(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_n t^n \tag{2}$$

from the Torre - Escaner wavelet $W(t)$, such that the root-mean squared error, E_{RMS}, is as small as possible,
i.e.,

$$E_{RMS} = \frac{1}{\sqrt{n}} \|W(t) - P_n(t)\|_2 < \epsilon \tag{3}$$

Let

$$\begin{aligned} G(a_0, a_1, \cdots, a_n) &= \sum_{i=1}^{m} \big[W(t_i) - P_n(t_i)\big]^2 \\ &= \sum_{i=1}^{m} \big[W(t_i) - a_0 - a_1 t_i - a_2 t_i^2 - \cdots - a_n t_i^n\big]^2 \end{aligned}$$

To minimize E_{RMS}, we set the following:

$$\begin{gathered} \frac{\partial G}{\partial a_0} = 0, \\ \frac{\partial G}{\partial a_1} = 0, \\ \vdots \\ \frac{\partial G}{\partial a_n} = 0, \end{gathered}$$

and solve the subsequent equations

$$\begin{aligned} \frac{\partial G}{\partial a_0} &= -2 \sum_{i=1}^{m} \big[W(t_i) - a_0 - a_1 t_i - a_2 t_i^2 - \cdots - a_n t_i^n\big] = 0, \\ \frac{\partial G}{\partial a_1} &= -2 \sum_{i=1}^{m} \big[W(t_i) - a_0 - a_1 t_i - a_2 t_i^2 - \cdots - a_n t_i^n\big]\, t_i = 0, \\ &\vdots \\ \frac{\partial G}{\partial a_n} &= -2 \sum_{i=1}^{m} \big[W(t_i) - a_0 - a_1 t_i - a_2 t_i^2 - \cdots - a_n t_i^n\big]\, t_i^n = 0 \end{aligned}$$

which simplifies to

$$\begin{aligned}
a_0\sum_{i=1}^{m}1 + a_1\sum_{i=1}^{m}t_i + \cdots + a_n\sum_{i=0}^{n}t_i^n &= \sum_{i=1}^{m}W(t_i),\\
a_0\sum_{i=1}^{m}t_i + a_1\sum_{i=1}^{m}t_i^2 + \cdots + a_n\sum_{i=0}^{n}t_i^{n+1} &= \sum_{i=1}^{m}W(t_i)t_i,\\
&\vdots\\
a_0\sum_{i=1}^{m}t_i^n + a_1\sum_{i=1}^{m}t_i^{n-1} + \cdots + a_n\sum_{i=0}^{n}t_i^{2n} &= \sum_{i=1}^{m}W(t_i)t_i^n. \qquad (4)
\end{aligned}$$

According to [3], (4) is a system of normal equations which is equivalent to

$$X^TXa = X^TW \tag{5}$$

where

$$X = \begin{bmatrix} 1 & t_1 & t_1^2 & \dots & t_1^n \\ 1 & t_2 & t_2^2 & \dots & t_2^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & t_m & t_m^2 & \dots & t_m^n \end{bmatrix}$$

and

$$W = \begin{bmatrix} W(t_1) \\ W(t_2) \\ \vdots \\ W(t_m) \end{bmatrix}.$$

Expression (5) is also equivalent to

$$\begin{bmatrix} \sum_{i=1}^m 1 & \sum_{i=1}^m t_i & \dots & \sum_{i=1}^m t_i^n \\ \sum_{i=1}^m t_i & \sum_{i=1}^m t_i^2 & \dots & \sum_{i=1}^m t_i^{n+1} \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{i=1}^m t_i^{n+1} & \sum_{i=1}^m t_i^{n+2} & \dots & \sum_{i=1}^m t_i^{2n} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^m W(t_i) \\ \sum_{i=1}^m W(t_i)t_i \\ \vdots \\ \sum_{i=1}^m W(t_i)t_i^{2n} \end{bmatrix},$$

which can be expressed as

$$La = b \tag{6}$$

where

$$L = \begin{bmatrix} \sum_{i=1}^{m} 1 & \sum_{i=1}^{m} t_i & \cdots & \sum_{i=1}^{m} t_i^n \\ \sum_{i=1}^{m} t_i & \sum_{i=1}^{m} t_i^2 & \cdots & \sum_{i=1}^{m} t_i^{n+1} \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{i=1}^{m} t_i^{n+1} & \sum_{i=1}^{m} t_i^{n+2} & \cdots & \sum_{i=1}^{m} t_i^{2n} \end{bmatrix},$$

$$a = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix},$$

and

$$b = \begin{bmatrix} \sum_{i=1}^{m} W(t_i) \\ \sum_{i=1}^{m} W(t_i) t_i \\ \vdots \\ \sum_{i=1}^{m} W(t_i) t_i^{2n} \end{bmatrix}.$$

Now, if L is invertible, then L^{-1} exists and matrix a can be solved.

2.2 Chebyshev Polynomial Approximation

Instead of monomial basis, Chebyshev Polynomial approximation uses different basis vectors.
Let

$$T_n(t) = \cos n\theta = \cos n\left(\arccos(t)\right) \tag{7}$$

where $-1 \leq t \leq 1$. We know that

$$\cos\,(n+1)\,\theta + \cos\,(n-1)\theta = 2\cos n\theta \cos\theta.$$

Thus, we have

$$T_{(n+1)}(t) + T_{(n-1)}(t) = 2tT_n(t)$$

which gives the recurrence relation form of the Chebyshev polynomials,

$$T_{(n+1)}(t) = 2tT_n(t) - T_{(n-1)}(t). \tag{8}$$

Hence, the basis vector is of the form $\{T_0(t), T_1(t), T_2(t), \ldots, T_n(t)\}$ which is also a basis vector of polynomials in terms of t.

According to a theorem by Weierstrass [2], if $P_n(t)$ interpolates a given function $f(t)$ on the zeros of $T_{n+1}(t)$, then

$$\|f(t) - P_n(t)\|_\infty \leq \frac{2^{-n}}{(n+1)!} \max_{t_0 \leq t \leq t_n} \left| f^{(n+1)}(t) \right| \tag{9}$$

where $t \in [-1, 1]$.

This theorem guarantees that a good fit of $f(t)$ can be found. Hence it is imperative to determine the zeros of $T_{n+1}(t)$. That can be done by simply setting (7) to zero and solving for the corresponding θ values,

$$\begin{aligned} \cos n\theta &= 0 \\ n\theta &= (2k+1)\frac{\pi}{2},\ k = 0, 1, 2, \ldots, (n-1), \\ \theta &= (2k+1)\frac{\pi}{2n},\ k = 0, 1, 2, \ldots, (n-1). \end{aligned}$$

Let

$$t_k = \cos\theta = \cos\left((2k+1)\frac{\pi}{2n}\right);\ k = 0, 1, 2, \ldots, (n-1) \tag{10}$$

be the *kth* zero of (7). Then the set generated by (10) is called the set of Chebyshev nodes of (7).

Next, we note that since $\{T_0(t), T_1(t), T_2(t), \ldots, T_n(t)\}$ is a basis for the set of polynomials, then the set is linearly independent and $T_i(t)$ is orthogonal to $T_j(t)$ for $i \neq j$ [1].

Hence, we can set

$$P_n(t) = \sum_{k=0}^{n} c_k T_k(t), \tag{11}$$

a linear combination of these basis vectors. Thus, we need to detemine each coefficient c_k to come up with the Chebyshev polynomial approximation $P_n(t)$ for $f(t)$.

$P_n(t)$ interpolates $f(t)$ at the $(n+1)$ Chebyshev nodes so that at every node t_k,

$$f(t_k) = P_n(t_k).$$

Hence,

$$\begin{aligned}\sum_{j=0}^{n} f(t_j)T_k(t_j) &= \sum_{i=0}^{n} c_i \sum_{j=0}^{n} T_i(t_j)T_k(t_j)\\ &= \sum_{i=0}^{n} c_i K_i \delta_{ik} \text{ (orthogonality of } T_i \text{ and } T_k)\\ &= \frac{1}{2}(n+1)c_k\end{aligned}$$

where $K_i = \frac{1}{2}(n+1)$.
Thus, the coefficients can be obtained by

$$c_k = \frac{2}{n+1}\sum_{j=0}^{n} f(t_j)T_k(t_j) \tag{12}$$

where $t_j = \cos\left(\frac{(j+\frac{1}{2})\pi}{n+1}\right)$.

3 Results and Discussions

After considering several choices, we come up with the common interval of interest for (1). The interval is $[-1, 1]$ because this is the only interval in which our Matlab codes give quality results. For one, the graph using *ChebPolyApprox* explodes outside $[-1, 1]$. For comparison purposes, we set a maximum root-mean square error, E_{RMS}, of 0.10 corresponding to 10 percent. This threshold might be considered large but setting a very small value might result in very high polynomial degrees.

3.1 Least Squares Approximation Results

Using the Matlab code $least\_squares(x, y, m)$, the minimum degree of the approximating polynomial is $n = m = 20$. For values larger than 20, the Matlab program returns a warning note that the solution to the matrix system (6) may be inaccurate. This is because L is a Hilbert matrix [5]. As the matrix becomes large, its determinant tends to zero. Thus, L becomes non-invertible.

For this particular degree, the Matlab program returns the polynomial representation for (1):

$$\begin{aligned} P_{20}(t) &= (4.2 - 0.01i)t^{20} + (9.32i)t^{19} + (-35.47 + 0.03i)t^{18} \\ &+ (0.01 - 67.28i)t^{17} + (144.3 - 0.06i)t^{16} + (-0.02 + 234.72i)t^{15} \\ &+ (-376.03 + 0.07i)t^{14} + (0.03 - 526.04i)t^{13} + (691.51 - 0.05i)t^{12} \\ &+ (-0.02 + 827.98i)t^{11} + (-914.14 + 0.02i)t^{10} + (0.01 - 921.35i)t^{9} \\ &+ (842.72 - 0.01i)t^{8} + 692.78it^{7} - 505.15t^{6} - 321.48it^{5} \\ &+ 174.48t^{4} + 78.11it^{3} - 27.29t^{2} - 6.68it + 0.87. \end{aligned} \tag{13}$$

Its graph is

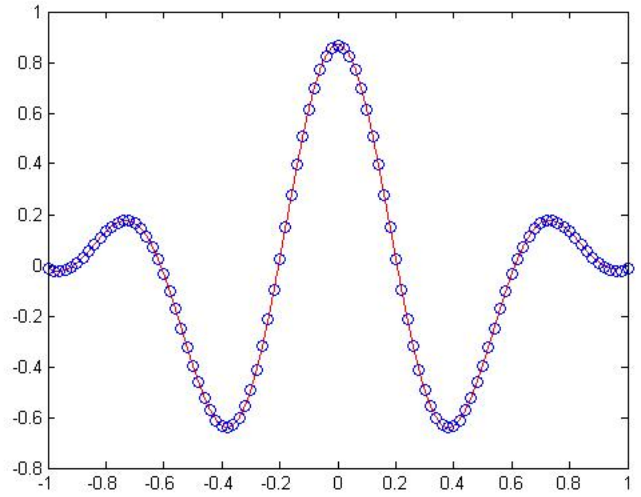

Figure 1: Graph of $P_{20}(t)$ with $W(t)$ in circles

However, its root-mean-squared error is

$$E_{RMS} = \frac{1}{\sqrt{n}} \|W(t) - P_{20}(t)\|_2 = 0.8014,$$

much bigger than the threshold root-mean squared error.

3.2 Chebyshev Polynomial Approximation Results

For the Chebyshev Polynomial approximation, the Matlab code was able to find a polynomial approximation that satisfies the threshold error. In fact, it was able to find more than one polynomial in the interval $[-1, 1]$. Since we want a polynomial of minimum degree, we only get the three lowest, in terms of degree, polynomial representations.

The lowest degree polynomial representation is

$$\begin{aligned} P_{10}(t) &= -97.36t^{10} + 54.93it^{9} + 284.32t^{8} \\ &\quad -145.35it^{7} - 304.26t^{6} + 133.73it^{5} \\ &\quad 142.08t^{4} - 48.64it^{3} - 25.73t^{2} \\ &\quad 5.28it + 0.87. \end{aligned} \tag{14}$$

Its graph is

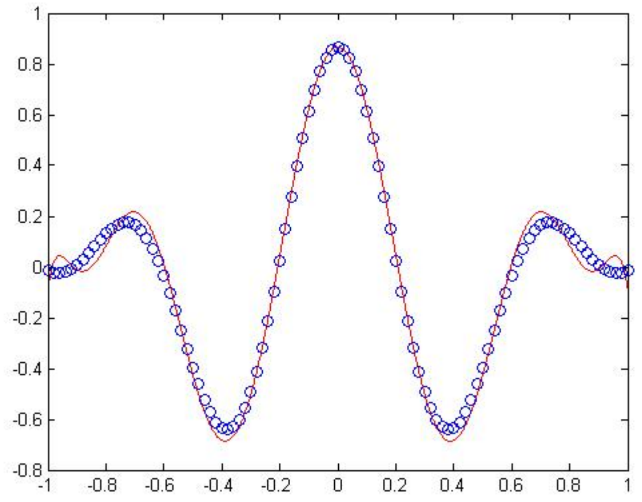

Figure 2: Graph of $P_{10}(t)$ with $W(t)$ in circles

and the corresponding root-mean-squared error is

$$E_{RMS} = \frac{1}{\sqrt{n}}\|W(t) - P_{10}(t)\|_2 = 0.0792,$$

which is less than than the threshold error 0.10.

The next lowest degree polynomial representation is

$$\begin{aligned} P_{11}(t) &= -104.19it^{11} - 72.21t^{10} + 346.99it^9 \\ &\quad +220.32t^8 - 444.54it^7 - 246.96t^6 \\ &\quad +268.81it^5 + 120.97t^4 - 73.82it^3 \\ &\quad -22.97t^2 + 6.62it + 0.81. \end{aligned} \tag{15}$$

Its graph is

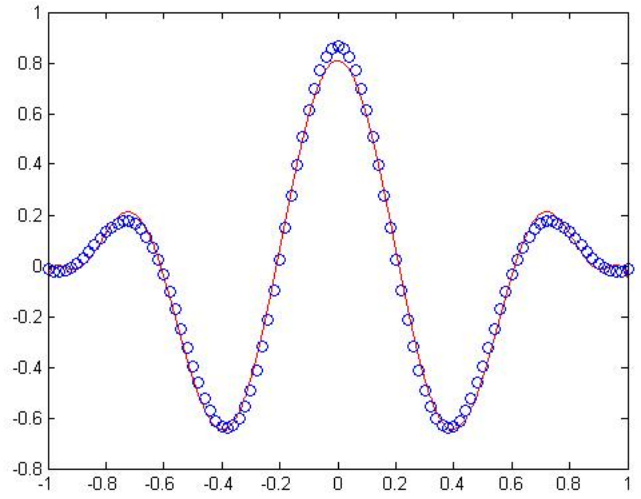

Figure 3: Graph of $P_{11}(t)$ with $W(t)$ in circles

while its root-mean-squared error is

$$E_{RMS} = \frac{1}{\sqrt{n}}\|W(t) - P_{11}(t)\|_2 = 0.0376$$

expectedly lower than $P_{10}(t)$.

Lastly, the third lowest degree polynomial representation is

$$\begin{aligned} P_{12}(t) &= 100.6t^{12} - 82.03it^{11} - 378.5t^{10} \\ &\quad +285.21it^9 + 571.23t^8 - 381.68it^7 \\ &\quad -433.14t^6 + 240.67it^5 + 165.93t^4 \\ &\quad -68.64it^3 - 26.99t^2 + 6.35it + 0.87. \end{aligned} \tag{16}$$

Its graph is

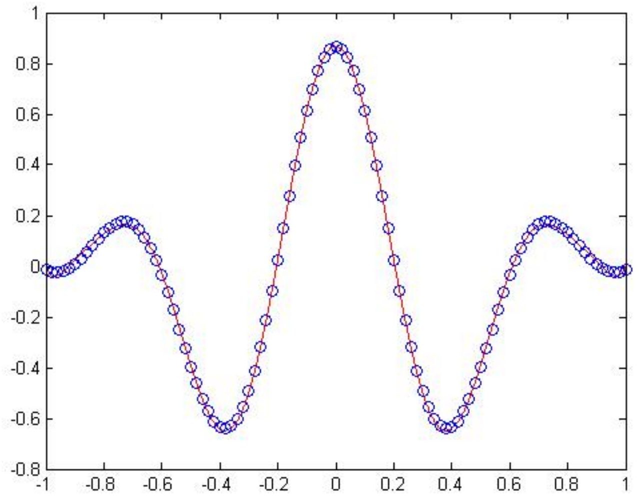

Figure 4: Graph of $P_{12}(t)$ with $W(t)$ in circles

Its root-mean-squared error is

$$E_{RMS} = \frac{1}{\sqrt{n}} \| W(t) - P_{12}(t) \|_2 = 0.0163,$$

much lower than $P_{11}(t)$ and $P_{10}(t)$.

4 Conclusions

The purpose of this paper is to find polynomial approximations to the Torre-Escaner wavelet with ideal root-mean-squared errors. We first used the Polynomial Least Squares (PLS) approximation scheme and the results showed that the only resulting polynomial representation (13) does not satisfy our threshold E_{RMS}. This means that PLS failed to provide a good polynomial approximation for (1).

On the other hand, the Chebyshev Polynomial approximation scheme was able to provide good polynomial approximations for (1) at lower polynomial degrees. This is a validation of the ”near-minimax” nature of the Chebyshev Polynomial approximation scheme.

For future works, we would like to examine other polynomial approximation schemes such as the Legendre, Gegenbauer, Jacobi and other orthogonal polynomials whether they can provide better approximations than the Chebyshev scheme.

References

[1] Bernard Kolman and Drexel Hill. *Elementary Linear Algebra.* Pearson Education, New Jersey, 2008.

[2] E. W. Cheney. *Introduction to Approximation Theory.* AMS Chelsea Publishing, Rhode Island, 1966.

[3] K. Atkinson and W. Han. *Elementary Numerical Analysis, Third Edition.* John Wiley and Sons, New York, 2004.

[4] Noemi Torre and Jose Maria Escaner IV. A New Wavelet based on Morlet and Mexican Hat Wavelets. *Matimyas Matematika*, 2011.

[5] W. Yang, W. Cao, et al. *Applied Numerical Methods Using MATLAB.* John Wiley and Sons, New Jersey, 2005.

Appendices

A Least Squares Polynomial Approximation Matlab Code

```
function least_squares(x, y, m)

% least_squares(x, y, m) fits a least-squares polynomial of
% degree m through a set of data where x and y are the
% coordinates.
% The output to least_squares(x, y, m) is the set of
% coefficients  a0, a1,...,am for the least-square
% polynomial Pm(x) = a0 + a1*x + a2*x^2 + ...+ am*x^m
% Hence, there will be m+1 values in the vector a.
% A graph displays the set of points and the fitted polynomial


n = size(x,1);
if n == 1
   n = size(x,2);
end

b = zeros(m+1,1);
for i = 1:n
   for j = 1:m+1
      % right-hand side column vector consisting of sums of
      % powers of x multiplied by y's
      b(j) = b(j) + y(i)*x(i)^(j-1);
   end
end


p = zeros(2*m+1,1);
for i = 1:n
   for j = 1:2*m+1
      % sums of powers of x
```

```
        p(j) = p(j) + x(i)^(j-1);
    end
end


for i = 1:m+1
    for j = 1:m+1
        % distributing the sums of powers of x in a matrix A
        L(i,j) = p(i+j-1);
    end
end

a = L\b

% creating an x-axis and evaluating the least-square polynomial
% this is only needed for data visualization
t = min(x):0.05:max(x);
n = size(t,2);
for i = 1:n
    f(i) = a(m+1);
    for j = m:-1:1
        f(i) = a(j) + f(i)*t(i);
    end
end

% data visualization
figure
plot(t,f)   % the least_squares polynomial
hold on
plot(x, y, 'r*')    % the data points
grid on
```

B Chebyshev Polynomial Approximation Matlab Code

```
function A =ChebPolyApprox(n, f, a, b);

for k = 1 : n + 1

    t = cos ( pi * ( k - 0.5 ) / ( n + 1 ) );

    y = ( ( 1.0 + t ) * b     ...
        + ( 1.0 - t ) * a ) ...
        /    2.0;

    d(k) = f ( y );

  end

  fac = 2.0 / ( n + 1 );
  for j = 1 : ( n + 1 )
    sum = 0.0;
    for k = 1 : ( n + 1 )
      sum = sum + d(k) * cos ( ( pi * ( j - 1 ) )
        * ( ( k - 0.5 ) / ( n + 1 ) ) );
    end
    c(j) = fac * sum;
  end

  c(1) = c(1) / 2.0;


t0 = 1;
t1 = [1 0];
if n == 0
    T = t0;
elseif n == 1;
    T = t1;
else
```

```
    for k=2:n
        T = [2*t1 0] - [0 0 t0];
        t0 = t1;
        t1 = T;
    end
end
A = zeros(n+1,n+1);

B = zeros(n+1,n+1);

for k = 0:n,
    tj = T;
    tj = fliplr(tj);
    B(k+1,1:k+1) = tj;
end
A = c*B;
A = fliplr(A);
```