Comparing Interpolation Method for the Vasicek Model

Jung Park, Christopher Ambrus, Ritwik Dhande, Roshan Oli

Introduction

The Vasicek model is a single factor short rate model that can be used to predict future interest rate changes. The model consists of three parameters which can be obtained by fitting the historical data. The initial part of the project is to fit the data to obtain the values of model parameters. We will use constant short maturity treasuries with maturities ranging from 6 months to 30 years. We want to minimize the root mean square error between the model par rate obtained from the Vasicek model and the historical par rate. Once we have obtained optimal parameters, we can use them to evaluate how the interest rate will change over the time period. This will serve as our data points for the interpolation.

The second part of the project consists of using different interpolation methods to interpolate the data points obtained from the Vasicek model and compare the result with the closed form solution of the Vasicek model. One problem with the Vasicek model is that when the time increment is not large enough, the result deviates from the closed form solution in the near-term future. Therefore, to obtain near-term interest rate, we run the Vasicek model with large enough time increment and interpolate in between data points. We will utilize cubic spline interpolation, Newton polynomial interpolation, Lagrange interpolation, and univariate polynomial regression.

Method

Vasicek Model

The Vasicek model is defined as [2]

$$\Delta r_{t+\Delta t} = (\alpha - \beta r_t)\Delta t + \sigma\epsilon\sqrt{\Delta t}.$$

The term $\alpha - \beta r_t$ represents the trend in rates, σ represents the volatility and ϵ is the standard normal random variable.

The yield to maturity on a T-year zero coupon bond is given by:

$$ytm = \frac{\ln\big(A(T)\big)}{T} + \frac{B(T)}{T} r_o,$$

$$A(T) = \exp\left\{\left(\frac{\sigma^2}{2\beta^2} - \frac{\alpha}{\beta}\right)T + \left(\frac{\alpha}{\beta^2} - \frac{\sigma^2}{\beta^3}\right)(1 - \exp(-\beta T)) + \frac{\sigma^2}{4\beta^3}(1 - \exp(-2\beta T))\right\},$$

$$B(T) = \frac{1}{\beta}(1 - \exp(-\beta T)).$$

Given the yield to maturity of a zero coupon bonds, we can obtain short rate, r_o. Our historical data consists of daily constant maturity yield from January 1st 2022 to March 15th 2024. Now, we will use this short rate to calculate discount factors from 0.5 year to 30 years at each date and use the discount factors to obtain par rate of different maturities.

$$D(T) = A(T)\exp(-B(T)r_o),$$

$$T\ year\ par\ rate = 2\left[\frac{100 - 100D(T)}{\sum D\left(\frac{i}{2}\right)}\right], i = 1 \ldots 2T.$$

Finally, we want to optimize our parameters such that the root mean square error between our model par rate and the historical par rate is minimized. Once we have obtained optimal parameters, we use the Monte Carlo method on the Vasicek model to simulate 10,000 paths to obtain how the interest rate changes over next 10 years with 1 year increment. We assume today's short rate to be 0.05. We will use these as our data points for interpolation and compare with the solution of the Vasicek model:

$$E[r_t] = r_o e^{-\beta t} + \frac{\alpha}{\beta}(1 - e^{-\beta t})$$

Interpolation Techniques

Cubic Spline Interpolation

The cubic spline method requires us to construct n-polynomials of degree 3 [1].

$$P_i(x) = a_o^{(i)} + a_1^{(i)}(x - x_i) + a_2^{(i)}(x - x_i)^2 + a_3^{(i)}(x - x_i)^3, i = 1 \ldots n, x_{i-1} < x < x_i$$

The conditions are:

$$P_i(x_i) = f(x_i), i = 1 \ldots n$$

$$P_1(x_o) = f(x_o)$$

$$P_i(x_i) = P_{i+1}(x_i), i = 1 \ldots n - 1$$

$$P_i'(x_i) = P_{i+1}'(x_i), i = 1 \ldots n - 1$$

$$P_i''(x_i) = P_{i+1}(x_i), i = 1 \ldots n - 1$$

$$P_1''(x_n) = P_n'' = 0$$

Conditions above give us 4n variables of $a_j^{(i)}$, j = 0…3 and i = 1…n.

Lagrange Polynomial Interpolation

The Lagrange Polynomial is defined as [1]

$$P(x) = \sum_{k=0}^{n} L_{k,n}(x) f(x)\text{, where}$$

$$L_{k,n}(x) = \prod_{\substack{i=0 \\ i\neq k}}^{n} \frac{x - x_i}{x_k - x_i}.$$

The Newton polynomial is defined as [1]

$$P_n(x) = f(x_0) + \sum_{i=1}^{n} \left(f[x_0 \dots x_1] \prod_{k=0}^{i-1} (x - x_k \right)\text{, where}$$

$$f[x_0 \dots x_n] = \frac{f[x_1 \dots x_n] - f[x_0 \dots x_{n-1}]}{x_n - x_0},$$

$$f[x_i, x_{i+1}] = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}.$$

Univariate Polynomial Regression

Lastly, we use univariate polynomial regression of degree 4 [1]. The model of degree k is defined as

$y = \gamma\beta$, where

$$x = \begin{bmatrix} 1 & x_1^1 & \cdots & x_1^k \\ \vdots & \vdots & & \vdots \\ 1 & x_n^1 & \dots & x_n^k \end{bmatrix},$$

$$\gamma = \begin{bmatrix} \gamma_1 \\ \gamma_1 \\ \vdots \\ \gamma_k \end{bmatrix},$$

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}.$$

We use ordinary least square estimation to obtain optimal β,

$$\hat{\beta} = (x^T x)^{-1} x^T y.$$

Results

Our model parameters α, β, σ are 0.0181325, 0.5235948, 0.0002373, respectively. The future interest rates are shown in the table below. Figures 2 to 5 show interpolation for each method.

Year	Interest Rate
1	0.046595548
2	0.039171938
3	0.036391507
4	0.035328126
5	0.034911479
6	0.034746169
7	0.034678138
8	0.034651474
9	0.034640264
10	0.034634682

Table 1. Interest rate change

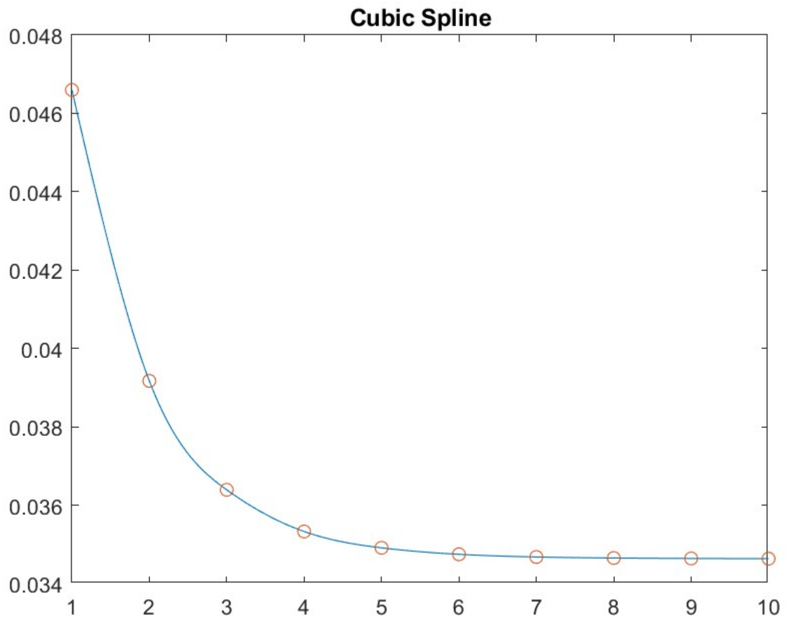

Figure 1. Cubic Spline Interpolation

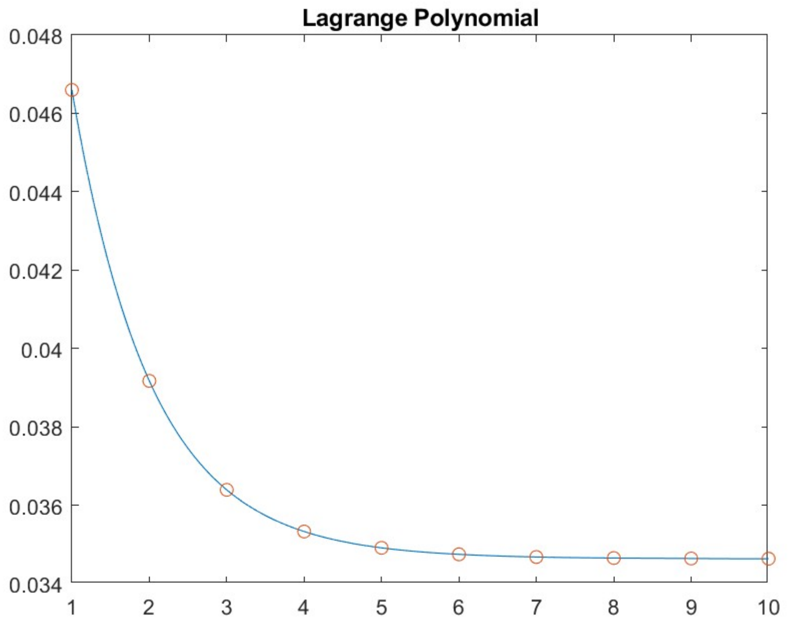

Figure 2. Lagrange Polynomial Interpolation

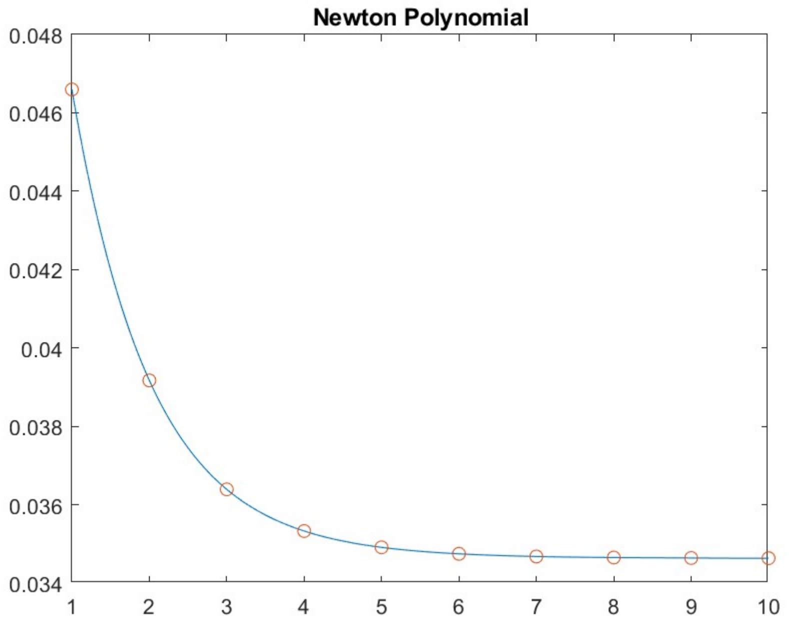

Figure 3. Newton Polynomial

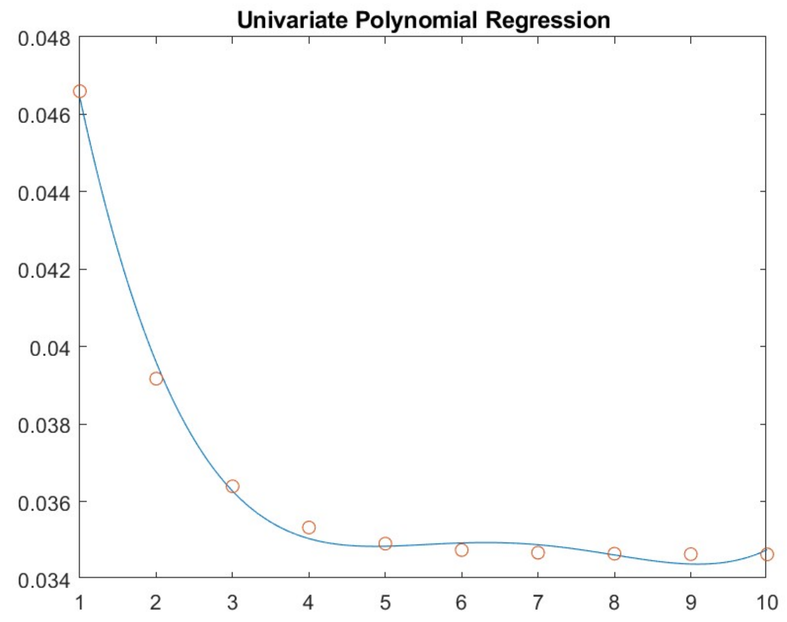

Figure 4. Univariate Polynomial Regression

Interpolation Method	1.5 Year Interest Rate	Percentage Error
Vasicek Model	0.041638248064378	
Cubic Spline	0.042459196472993	1.972%
Lagrange Polynomial	0.041989315191834	0.843%
Newton Polynomial	0.041989315191834	0.843%
Univariate Polynomial Regression	0.042473519254026	2.006%

Table 2. Interest rates after 1.5 years

Discussion

The Vasicek model has mean reverting behavior as can be seen in the figure 2 to 5. When we put all the graphs together, we observe differences before year 3 and convergence in all methods later on. We will compare the interest rate with the closed form solution of the Vasicek model. The results are shown in the Table 2 along with interest rate of each method. The Lagrange polynomial and Newton polynomial has same values since we used the same data points for both methods. We would observe difference if we add more points for the Newton polynomial method. Based on figure 5, the univariate polynomial regression does not directly go through the data points. Indeed, the percentage error is greatest of all four methods. If we increase the degree to 6, we can enforce the regression to approach our data points closer. At degree of 6, the percentage error goes down to 0.996%. Based on results in Table 2, the Lagrange polynomial method has the smallest percentage error. Therefore, we should use the Lagrange polynomial method when interpolating the Vasicek model for the best accuracy.

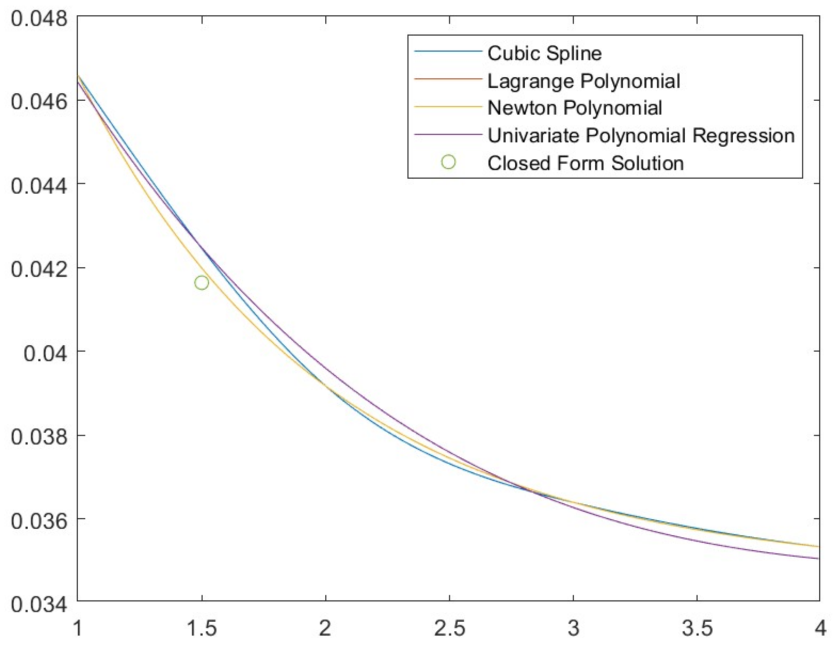

Figure 5. All Methods from Year 1 to Year 3

Reference

[1] Brandimarte, Paolo. *Numerical Methods in Finance and Economics: A MATLAB-Based Introduction*. Wiley, 2006.

[2] Etheridge, Alison M. *A Course in Financial Calculus*. Cambridge University Press, 2008.

Appendix A

Vasicek Model

```
clear
clc

%maturities
maturities = [0.5, 1,2,3,5,7,10,20,30];

%CMT, 3m,6m,1yr,2yr,3yr,5yr,7yr,10yr,20,30
cm=readmatrix('cmt_rates.xls');
cm = cm(5:end,3:12);
constantmat = rmmissing(cm);

%fminsearch to optimize RMSE
x0=[5,5,5];
x1=0;
x2=1;
A=[];
B=[];
Aeq=[];
Beq=[];
lb=[];
ub=[];
%soln = fmincon(@objfunc,x0,A,B,Aeq,Beq,lb,ub);
soln = [0.0181325124554064,0.523594883310337,0.000237301709060087];
RMSE = objfunc(soln);

%%
%Yield Curve
%paths from 1-30yrs, 10,000 paths.
alpha = soln(1);
beta = soln(2);
sigma = soln(3);
dt=1;
r0 =0.05;
ratepath = zeros(10000,30/dt);
ratepath(:,1)=r0;

%short rate simulation using monte carlo
for i =2:30/dt
    for j = 1:10000
        delta_term = (alpha-beta*ratepath(j,i-1))*dt + sigma*randn(1)*sqrt(dt);
        ratepath(j,i) = ratepath(j,i-1) + delta_term;
    end
end
avgrate = mean(ratepath);

%ytm, yield curve
for i =1:length(avgrate)
    ytm(i) = vdt(avgrate(i),alpha,beta,sigma,i/dt);
end
```

```
%%

function rtnext = rt(rate,alpha,theta,sigma,dt)
    term1 = alpha*(theta-rate)*dt;
    term2 = sigma*sqrt(dt)*randn(1);
    rtnext = rate+term1+term2;
end

function [ytm] = vdt(rate,alp,bet,sig,T)
    term1 = (  (sig^2)/(2*(bet^2))   -(alp/bet)   )*T;
    term2 = (alp/(bet^2)-(sig^2)/(bet^3))*(1-exp(-bet*T));
    term3 = (sig^2)/(4*(bet^3))*(1-exp(-2*bet*T));
    At = exp(term1+term2+term3);
    Bt = 1/bet*(1-exp(-bet*T));
    vasicekdt = At*exp(-Bt*rate);
    ytm = -log(At)/T+Bt/T*rate;
end

function [At,Bt] = shortrate(alp,bet,sig,T)
    term1 = (  (sig^2)/(2*(bet^2))   -(alp/bet)   )*T;
    term2 = (alp/(bet^2)-(sig^2)/(bet^3))*(1-exp(-bet*T));
    term3 = (sig^2)/(4*(bet^3))*(1-exp(-2*bet*T));
    At = exp(term1+term2+term3);
    Bt = 1/bet*(1-exp(-bet*T));
end

function par = parrate(time,DISCOUNT,i)
    dis= DISCOUNT;
    nomi= 100-100*dis;
    denom= sum(discount(i,1:2*time));
    par= 2*nomi/denom;
end

function rmse = objfunc(x)
    maturities2= [0.5, 1,2,3,5,7,10,20,30];

    cm=readmatrix('cmt_rates.xls');
    cm = cm(5:end,3:12);
    constantmat = rmmissing(cm);



    % obtain short rate r0 for all of them
    [At,Bt] = shortrate(x(1),x(2),x(3),0.25);

    % step3) short rate for 0.25yr at each date.
    for j=1:551
        term1 = constantmat(j,1)+log(At)/0.25;
        shortR(j,1) = term1*0.25/Bt/100;
    end

    % step4) discount rate from 0.5yr ~ 30yr for all date.
    for i = 1:551
```

```
        for j = 1:60
        [Att,Btt]=shortrate(x(1),x(2),x(3),j/2);
        discount(i,j) = Att*exp(-Btt*shortR(i));
        end
    end

    % step5) use Discount factors to obtain model par rates.
    for i = 1:551
        for j = 1:length(maturities2)
            time = maturities2(j);
            dis = discount(i,time*2);
            nomi= 100-100*dis;
            denom= sum(discount(i,1:2*time));
            modelpar(i,j) = 2*nomi/denom;
        end
    end

    %error between actual CMT and model par rates
    errormatrix = constantmat(:,2:end)-modelpar;
    %RMSE
    sqrt(sum(errormatrix.^2,'all')/(numel(errormatrix)))
    rmse = sqrt(sum(sum(errormatrix.^2))/(numel(errormatrix)));

end
```

Appendix B

Interpolation

```
clc
clear

%interpolation
y= [0.046595548     0.039171938  0.036391507  0.035328126  0.034911479  0.034746169
       0.034678138  0.034651474  0.034640264  0.034634682
];
x=1:10;

%cubic spline
%number of conditions conditions(4n)
total=(length(x)-1)*4;
A=zeros(total); % (x-xi)
B=zeros(total,1); %soln we need [a0_1,a1_1,a2_1,a3_1,,,a2_5,a3_5]
C=zeros(total,1);
n=length(x)-1;
% a0_i,a1_i,a2_i,a3_i, i=1...5

%Interpolation condition, p(x)=f(x), p1(x0) = f(x0); n+1 conditions
a=0;
for i =1:n
    xval=x(i+1);
    for j = 1:4
        a=a+1;
        A(i,a)=(x(i+1)-xval)^(j-1);
    end
    C(i)=y(i+1);
end
for j=1:4
    A(n+1,j)=(x(1)-x(2))^(j-1);
end
C(n+1)=y(1);
total_cond = n+1;

%Continuity condition, p_i(x_i)=p_i+1(x_i), n-1 conditions
a=0;
for i =1:n-1
    total_cond=total_cond+1;
    for j=1:4
        a=a+1;
        A(total_cond,a)= (x(i+1)-x(i+1))^(j-1);
        A(total_cond,a+4) = -(x(i+1)-x(i+2))^(j-1);
    end
    C(total_cond) = 0;
end

%Smoothness condition, p'_i(x_i) = p'_i+1(x_i) ,n-1 conditions
% P'_i(x) = a1+ 2*a2(x-xi)+3*a3(x-xi)^2
a=0;
for i =1:n-1
    total_cond=total_cond+1;
    for j=1:4
        a=a+1;
        if j==1
            A(total_cond,a)=0;
            A(total_cond,a+4) = 0;
        elseif j==2
            A(total_cond,a)=1;
            A(total_cond,a+4) = -1;
        else
            A(total_cond,a)= (x(i+1)-x(i+1))^(j-1);
            A(total_cond,a+4) = -(j-1)*(x(i+1)-x(i+2))^(j-2);
        end
    end
    C(total_cond) = 0;
end

%2nd derivative condition, P''_i(xi) = p''_i+1(xi), n-1 conditions
% p''_i(x) = 2*a2+6*a2(x-xi)
a=0;
for i = 1:n-1
    total_cond=total_cond+1;
    for j = 1:4
        a=a+1;
        if j==3
            A(total_cond,a)=2;
            A(total_cond,a+4)=-2;
        elseif j==4
            A(total_cond,a+4)=-6*(x(i+1)-x(i+2))^(j-3);
        end
    end
    C(total_cond)=0;
end

%natural spline condition, 2conditions
total_cond=total_cond+1;
C(total_cond)=0;
A(total_cond,3)=2;
A(total_cond,4)=6*(x(1)-x(2));

total_cond=total_cond+1;
C(total_cond)=0;
A(total_cond,end-1)=2;

B = inv(A)*C;
count=1;
for i = 1:length(x)-1
    for j=1:4
        soln(i,j)=B(count);
        count=count+1;
    end
end

%testing
testx = 1:0.01:10;
for i = 1:length(testx)
```

```
    xi = floor(testx(i)+1);
    xx = [1; (testx(i)-xi); (testx(i)-xi)^2; (testx(i)-xi)^3];
    if xi<11
        cspline(i) = soln(xi-1,:)*xx;
    else
        cspline(i) = soln(9,:)*xx;
    end
end
plot(testx,cspline)
hold on
scatter(x,y)
title('Cubic Spline')


%%
%Newton Polynomial
newton = zeros(n+1,n+1);
newton(1,:) = y;
for i =1:n
    for j = 1:n-i+1
        newton(i,j+1)
        newton(i,j)
        x(j+i)
        x(j)
        newton(i+1,j) = (newton(i,j+1)-newton(i,j))/(x(j+i)-x(j));
    end
end

newton_coefs= newton(:,1);
newton_x = x(1:9)';

for i = 1:length(testx)
    newton_soln = 0;
    xvar = testx(i)-newton_x;
    for j = 1:length(x)-1
        xcoef(j,1) = prod(xvar(1:j));
    end
    newton_soln = xcoef'*newton_coefs(2:end)+newton_coefs(1);
    newtonpoly(i) = newton_soln;
end
plot(testx,newtonpoly)
hold on
scatter(x,y)
title('Newton Polynomial')

%%
%Lagrange Polynomial


L=ones(length(x),length(testx));
for k = 1:length(x)
    for i = 1:length(x)
        if k~=i
            L(k,:)=L(k,:).*(testx-x(i))/(x(k)-x(i));
        end
    end
end

lagpoly = 0;
for i =1:length(x)
    lagpoly = lagpoly+y(i)*L(i,:);
end
plot(testx,lagpoly)
hold on
scatter(x,y)
title('Lagrange Polynomial')
%%
%Univariate Polynomial Regression
for i = 1:10
    UPRx(i,:) = [1,x(i),x(i)^2,x(i)^3,x(i)^4];
    %UPRx(i,:) = [1,x(i),x(i)^2,x(i)^3,x(i)^4,x(i)^5,x(i)^6];
end

beta = inv(UPRx'*UPRx)*UPRx'*y';
for i = 1:length(testx)
    xi = testx(i);
    UPRy(i) = [1,xi,xi^2,xi^3,xi^4]*beta;
    %UPRy(i) = [1,xi,xi^2,xi^3,xi^4,xi^5,xi^6]*beta;
end
plot(testx,UPRy)
hold on
scatter(x,y)
title('Univariate Polynomial Regression')


%%
% x = 1.5 @ index 51;
realsoln = 0.041638248064378;
model = [lagpoly(51),newtonpoly(51),cspline(51),UPRy(51)];
percenterror = abs(model-realsoln)/realsoln*100
plot(testx(1:300),cspline(1:300))
hold on
plot(testx(1:300),lagpoly(1:300))
plot(testx(1:300), newtonpoly(1:300))
plot(testx(1:300), UPRy(1:300))
scatter(1.5, realsoln)
legend('Cubic Spline','Lagrange Polynomial', 'Newton Polynomial','Univariate
Polynomial Regression','Closed Form Solution')
```