```
%
 ==========================================================================
% AUTHOR ..... [Lishan Huang]
% UPDATED .... [Jan 15]
%
```

sinTayorPoly function

sinTayorPoly.m

```
%
 ==========================================================================
% AUTHOR ..... [Lishan Huang]
% UPDATED .... [Jan 15]
%
% Evaluate the truncated Taylor series for sin(x) about the point x0 =
 0
%
% INPUT
% x .... Vector of values to evaluate the Taylor series at
% n .... Integer of last term to evaluate in Taylor series
%
% OUTPUT
% T : Evaluated Taylor series at points given by x to n terms
%
 ==========================================================================
function T = sinTayorPoly(x, n)
% Initialize sum as 0
T = 0;
% Loop over terms in series
for k = 0:n
    T = T +(-1)^k * x.^(2*k+1)/factorial(2*k+1);
end
end
```

sinHorner function

sinHorner.m

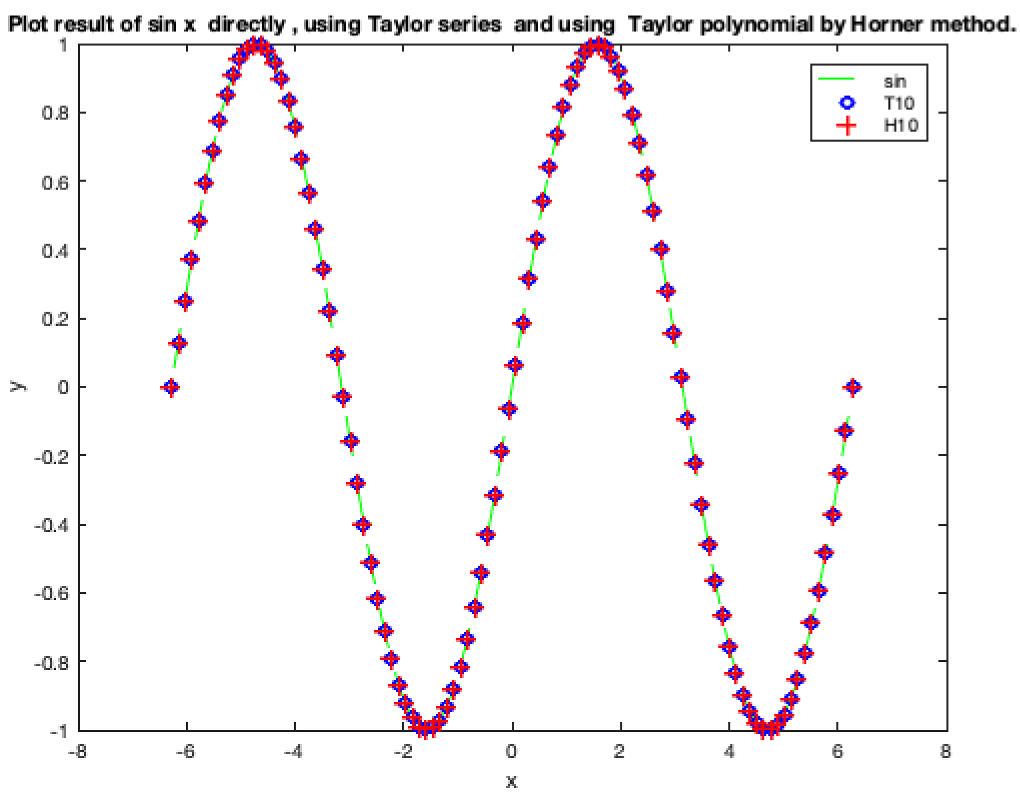
```
%
 ==========================================================================
% AUTHOR ..... [Lishan Huang]
% UPDATED .... [Jan 15]
%
% Evaluate the truncated Taylor series for sin(x) about the point x0 =
 0 by
% using Horner?s method.
%
% INPUT
% x .... Vector of values to evaluate the Taylor series at
% n .... Integer of last term to evaluate in Taylor series
```

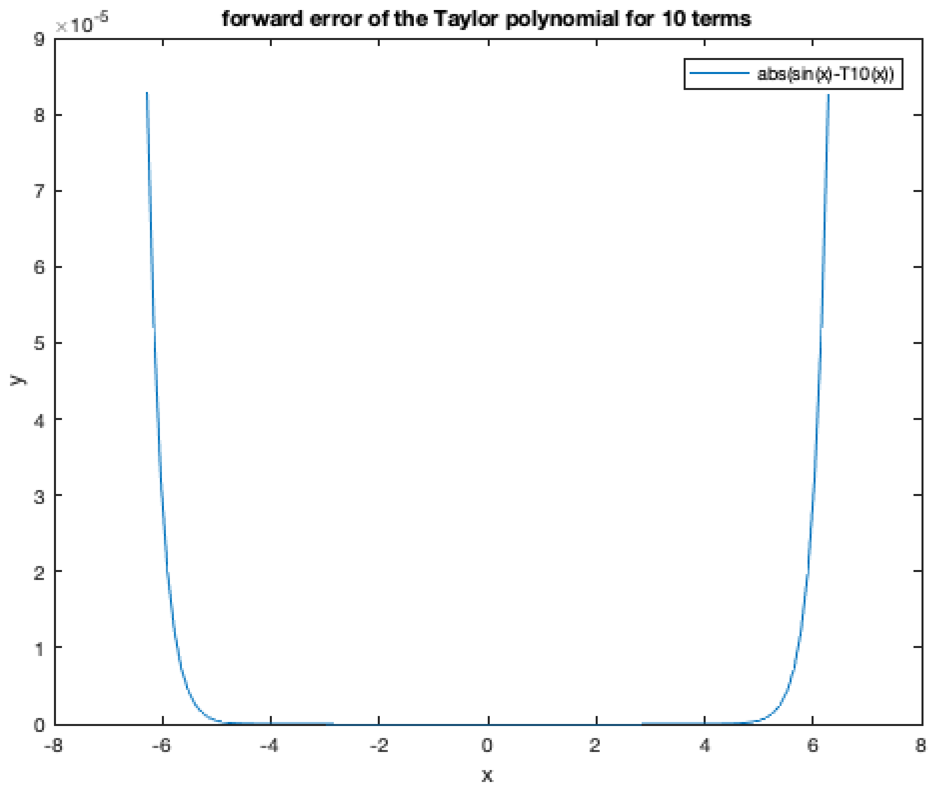
```
%
% OUTPUT
% T : Evaluated Taylor series at points given by x to n terms
%
 =========================================================================
function H = sinHorner(x, n)
%initialize the H as (-1)^n
H = (-1)^n;
% Loop over terms in series
for k = n:-1:1
    H=(-1)^(k-1)+x.^2 .* H / (2*k*(2*k+1));
end
H=H.*x;
end
```

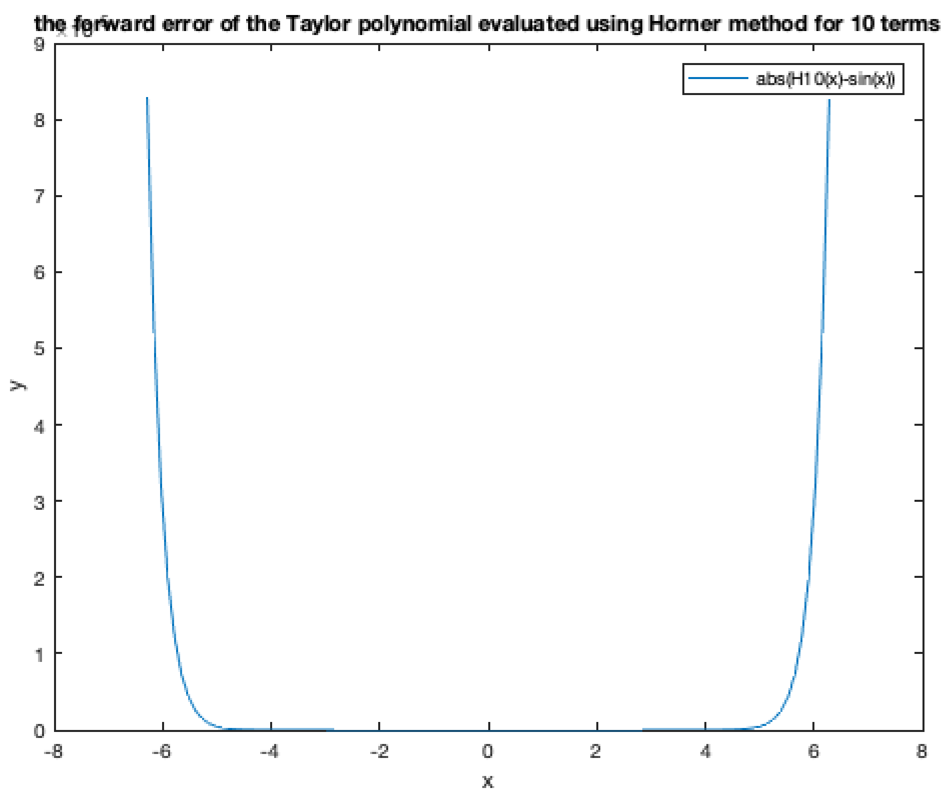
Script

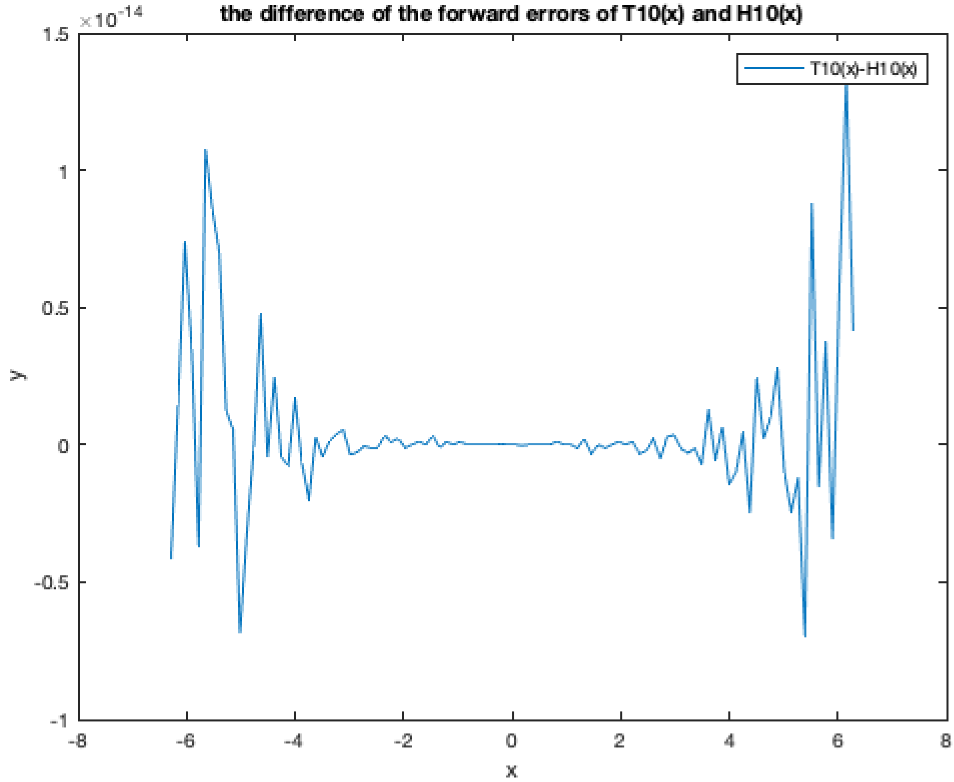
```
%The following is a code that Plotted the approximates of sin(x) on
%different methonds and comparing their differences and errors
%
clf;
close all;
clear all;
% Create a vector of 100 equally-spaced x values between -2*pi and
 2*pi
y=linspace(-2*pi,2*pi,100);
%Evaluate sin(x)
sin=sin(y);
% Evaluate the sin(x) Taylor Polynomial at n = 10
T10=sinTayorPoly(y,10);
% Evaluate the exp(x) Taylor Polynomial on Horner methond at n = 10
H10=sinHorner(y,10);
% figure 1
figure
%plot sin as  green dashed line, T10(x) as blue circle marker H10(x)as
 red
%sign marker
plot(y,sin,'--g',y,T10,'ob',y,H10,'+r')
legend('sin','T10', 'H10')
xlabel('x')
ylabel('y')
title('Plot result of sin x  directly , using Taylor series  and using
  Taylor polynomial by Horner method.')

% figure 2
figure(2)
%plot forward error of the Taylor polynomial for 10 terms
plot(y,abs(sin-T10))
legend('abs(sin(x)-T10(x))')
title('forward error of the Taylor polynomial for 10 terms')
xlabel('x')
ylabel('y')
```

```
% figure 3
figure(3)
%plot of the forward error of the Taylor polynomial evaluated using
 Horner?s method for 10 terms
plot(y,abs(H10-sin))
legend('abs(H10(x)-sin(x))')
title('the forward error of the Taylor polynomial evaluated using
 Horner method for 10 terms')
xlabel('x')
ylabel('y')

% figure 4
figure(4)
%plot will be the difference of the forward errors of T10(x) and
 H10(x).
plot(y,T10-H10)
legend('T10(x)-H10(x)')
title('the difference of the forward errors of T10(x) and H10(x)')
xlabel('x')
ylabel('y')
```




forward error of the Taylor polynomial for 10 terms
abs(sin(x)-T10(x))
y
x


the forward error of the Taylor polynomial evaluated using Horner method for 10 terms
abs(H10(x)-sin(x))
y
x




```
% *Text Answers*
% # This is the answer to the first question
% The forward error satisfies |sin x - T10| <= (2*pi)*23/23! , but it
 does not satisfy |sin x ? T10(x)| <= 1/23 ! of x on [-1 1], which is
 caused by roundoff errors that may increase or decrease the result
 of T10(x). Since MATLAB only can express fraction at most 52 bits
 in binary, when the actual result is longer than the limit bits then
 roundoff error occurs, it will increase the absolute value of the
 difference between sin x  and T10(x) in the last digit of the 52 bits
 and this increase is much larger than 1/23!, so it can not satisfy
 the second claim. The graph is not smooth because x is in [-1 1],
 comparing to the difference between each x, the absolute values of
 differences between roundoff error results and no roundoff error
 results are too large when the y-axis is amplified.

%# This is the answer to the second question
%No, they produce different results, because the result of each
 calculation in MATLAB will be roundoff if the result has a decimal
 part  longer than 52 bits in binary, which will cause the result
 either larger or smaller, so when these methods have different
 roundoff actions in their partial calculation since their calculation
 procedures are different, thus they will produce different results.
```

Published with MATLAB® R2017b