On Evaluating Infinite Series -- An Example

As mentioned in the Lesson #4 Lecture Notes, the Taylor series is an essential tool for applied numerical methods and for the general field of mathematical modeling. In addition, numerically evaluating and plotting infinite series expansions are common tasks that occur frequently in engineering applications. Thus, the goals of this example are to illustrate how to develop a Taylor series expansion for a given function, $f(x)$, and then to develop an algorithm to efficiently evaluate and plot the resultant series expansion.

In this application, we will use the hyperbolic sine function to illustrate the appropriate procedures, where

$$f(x) = \sinh x = \frac{e^x - e^{-x}}{2} \tag{1}$$

If we generate a Taylor series expansion for sinh x about the point $x_o = 0$, we start with the general Taylor series,

$$f(x_o + h) = f(x_o) + f'(x_o)h + \frac{f''(x_o)h^2}{2!} + \frac{f'''(x_o)h^3}{3!} + \cdots \tag{2a}$$

and let $x = x_o + h$ or $h = (x_o + h) - x_o = x - x_o$. However, with $x_o = 0$, then $h = x$ and eqn. (2a) becomes

$$f(x) = f(x_o) + f'(x_o)x + \frac{f''(x_o)x^2}{2!} + \frac{f'''(x_o)x^3}{3!} + \cdots \tag{2b}$$

Now, the function and all its derivatives in eqn. (2b) can be easily evaluated at $x_o = 0$, as

$f(x) = \sinh x$	and	$f(x_o) = 0$
$f'(x) = \cosh x$	and	$f'(x_o) = 1$
$f''(x) = \sinh x$	and	$f''(x_o) = 0$
$f'''(x) = \cosh x$	and	$f'''(x_o) = 1$
etc.		etc.

Thus, we see that all the even-order terms vanish and the resultant Taylor series for sinh x can be written as

$$f(x) = \sinh x = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots = \sum_{n=1}^{\infty} \frac{x^{2n-1}}{(2n-1)!} \tag{3}$$

Now, the question of interest becomes "How do we evaluate eqn. (3)?". This is a very practical goal and, if done improperly, can lead to lots of problems.

Actually, there are two issues, as follows:

1. Obviously, we cannot use an infinite number of terms in the expansion. Thus, one concern is associated with determining how many terms to include. This is usually accomplished by

setting some user-specified error tolerance, **tol**, and then one simply truncates the series when the relative change associated with adding another term is less than **tol**.

2. The second issue concerns computation of the individual terms in the series. Clearly, if the series is convergent, the individual terms must eventually get smaller and smaller and approach zero as $n \rightarrow \infty$. However, as apparent in eqn. (3), the magnitude of the numerator and denominator can become quite large -- to the point where round off error can become significant.

   For example, if $x = 2$ and $n = 10$, the tenth term in eqn. (3) would be

   $$\frac{2^{19}}{19!} = \frac{524288}{1.21645 \times 10^{17}} = 4.30998 \times 10^{-12}$$

   which is definitely the kind of computation that we would like to avoid (now that we have some understanding of round off error).

One way to address these two issues, and to significantly improve the overall computational efficiency, is to implement the **infinite power series** as a recurrence relation of the form,

$$f(x) = \sum_{n=1}^{\infty} T_n(x) \tag{4}$$

with $T_{n+1} = r_n T_n$ (5)

where r_n is the ratio of the $(n+1)^{th}$ term, T_{n+1}, to the nth term, T_n. With this form, one can easily and efficiently utilize the following algorithm:

Algorithm to Evaluate Infinite Power Series

- Set maximum number of terms, **maxT**, and the user-specified tolerance, **tol**, for stopping the calculation.

- Initialize the counter variable n and set the first term in the series -- set $n = 1$ and $T = T_1$.

- Initialize the partial sum after first term -- set $f = T$.

- while $\varepsilon >$ **tol** and $n <$ **maxT**

    compute r, where $r_n = T_{n+1}/T_n$ (specific to function of interest -- see below)

    $T = r*T$ (compute next term in series)

    $f = f + T$ (update partial sum)

    $\varepsilon = max(abs(T/f))$ (compute maximum relative change due to $(n+1)^{th}$ term)

    $n = n + 1$ (increment counter)

    end

This algorithm works great, with only minor changes for a variety of functions written in the form of an infinite series involving powers of x. The only steps that are case-specific involve initializing the first term, T_1, and the computation of the ratio, $r_n = T_{n+1}/T_n$, whose formula must be determined prior to implementation. For the present case, where $f(x) = \sinh x$, r_n is given by

$$r_n = \frac{T_{n+1}}{T_n} = \frac{x^{2(n+1)-1}}{(2(n+1)-1)!} \times \frac{(2n-1)!}{x^{2n-1}}$$

$$= \frac{x^2 x^{2n-1}}{(2n+1)(2n)(2n-1)!} \times \frac{(2n-1)!}{x^{2n-1}} = \frac{x^2}{(2n+1)(2n)} \qquad (6)$$

where we have used the fact that

$$(2(n+1)-1)! = (2n+2-1)! = (2n+1)! = (2n+1)(2n)(2n-1)!$$

Thus, with the value of r_n given by eqn. (6) and the fact that $T_1 = x$, we are ready to implement the above algorithm for the current example.

This was done in function file **sinh_series.m**, which is listed in Table 1. This function file is simply called with one of the following syntaxes:

```
f = sinh_series(x)
f = sinh_series(x,maxT)
f = sinh_series(x,maxT,tol)
```

Table 1 Listing of the sinh_series.m function file.

```
%
%   SINH_SERIES.M    Evaluate sinh(x) using Taylor Series
%
%   Inputs:
%     x - vector of independent variable values
%     maxT - maximum number of terms in series (optional, default value = 10)
%     tol - error tolerance for truncating series (optional, default value = 0.0001)
%
%   Outputs:
%     f - value of sinh(x) evaluated at all x values
%
%   File prepared by J. R. White, UMass-Lowell (last update: Oct. 2017)
%

      function f = sinh_series(x,maxT,tol)
%
%  set defaults if no inputs for maxT or tol, and check on minimum value for tol
      if nargin == 1, maxT = 10; tol = 0.0001;   end
      if nargin == 2,  tol = 0.0001;   end
      if isempty(maxT) || maxT < 1, maxT = 10;   end
      if tol < 100*eps, tol = 100*eps;   end
%
%  initialize variables and perform computational loop
      T = x;  f = T;    rerr = 1.0;   n = 1;
      while rerr > tol && n < maxT
        r = x.*x/((2*n+1)*2*n);
        T = r.*T;        % next termin series
        f = f + T;       % update running sum
        i = find(f);     % finds indices of nonzero values of f
        rerr = max(abs(T(i)./f(i))); % calc max relative error
        n = n+1;
      end
%
%  display warning if hit max # of terms
      if n == maxT
        disp(' ')
        disp('  *** WARNING ***  Hit max # of terms in sinh_series.m')
        disp(' ')
      end
%
%   end of function
```

If only one input is used (i.e. **maxT** and **tol** are not specified in the input stream), then default values of **maxT = 10** and **tol = 0.0001** are specified internally within the function file. Note also that the minimum allowed value of **tol** is 100***eps**, where **eps** is the built-in value of machine epsilon. Finally, we note that, since dot arithmetic is used consistently inside **sinh_series.m**, the variable **x** can be a single value or a vector (or matrix) of values.

To illustrate the use of **sinh_series.m**, we can type the following commands at the Matlab prompt:

```
>>     format short e
>>     x = linspace(100*eps,5,11);  x = x';
>>     f = sinh_series(x);
>>     rerr = (f-sinh(x))./sinh(x);
>>     disp('       x        f = sinh(x)   rel error'),   [x  f  rerr]
```

which gives (with some slight editing)

```
       x        f = sinh(x)    rel error
  2.2204e-14   2.2204e-14             0
  5.0000e-01   5.2110e-01             0
  1.0000e+00   1.1752e+00             0
  1.5000e+00   2.1293e+00   -8.5511e-15
  2.0000e+00   3.6269e+00   -1.2000e-12
  2.5000e+00   6.0502e+00   -5.0175e-11
  3.0000e+00   1.0018e+01   -9.7455e-10
  3.5000e+00   1.6543e+01   -1.1127e-08
  4.0000e+00   2.7290e+01   -8.6059e-08
  4.5000e+00   4.5003e+01   -4.9429e-07
  5.0000e+00   7.4203e+01   -2.2453e-06
```

Clearly, this shows that our infinite series implementation agrees with Matlab's built-in *sinh* function to within the default tolerance of 0.0001.

However, if we simply type

```
>>     sinh_series(10)
  *** WARNING ***  Hit max # of terms in sinh_series.m
ans =
   1.0989e+04
```

we get a numerical value, but the default value of 10 terms was not sufficient to compute the value of sinh(10) = 1.1013e+004 (from Matlab) to the specified accuracy.

We can increase the needed tolerance

```
>>     sinh_series(10,[],0.05)
ans =
   1.0907e+04
```

or increase the maximum number of terms in the series expansion

```
>>     sinh_series(10,25)
ans =
   1.1013e+04
```

and both these options get rid of the warning that the maximum number of terms was reached -- but the second option clearly gives the better result!

Finally, we could plot f(x) = sinh(x) using our infinite series expansion with the following commands:

```
>>    clear x f
>>    x = 0:0.1:2;    f = sinh_series(x);
>>    plot(x,f,'r-','LineWidth',2),grid
>>    title('Plot of f(x) = sinh(x) using Taylor Series')
>>    xlabel('x value'), ylabel('f(x) = sinh(x)')
```

The plot given in Fig. 1 reproduces the result of the above sequence -- and it clearly behaves as expected.
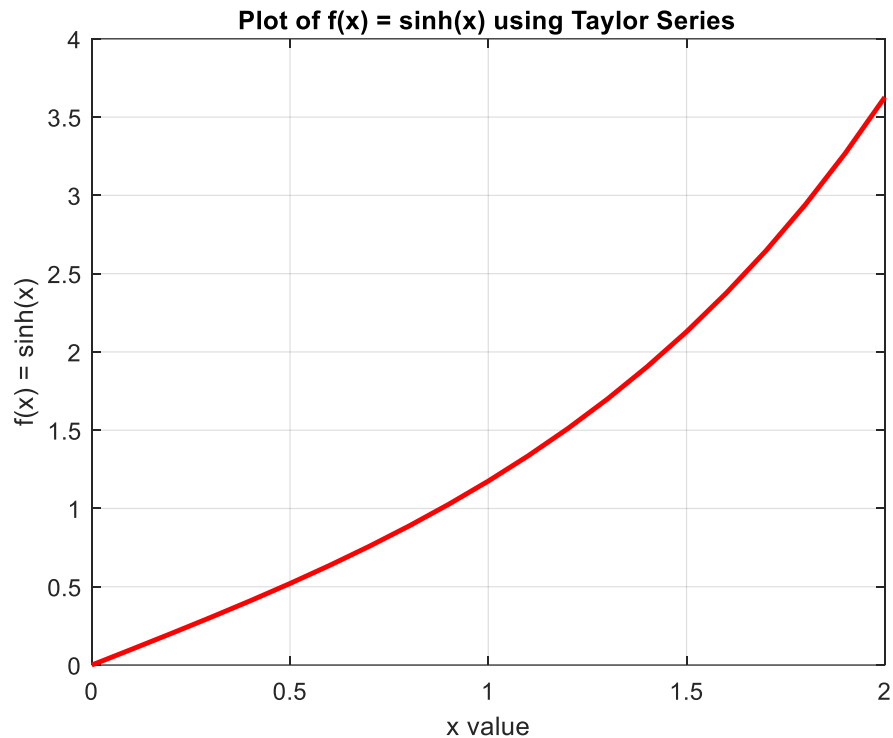

Fig. 1 Plot of series expansion for sinh(x) (from sinh_series.m).

In concluding this simple example, I should note that the recursive approach developed here to evaluate an infinite power series is a fairly common for many numerical methods. In addition, the ***while … end*** structure, with a test on the maximum number of times through the loop and a check on some error criterion, is seen routinely in most iterative schemes. Thus, this example not only represents a good illustration of working with Taylor series and numerically evaluating the resultant expansion, it also gives you a preview of some structures that are common to many algorithms. Thus, it is worth your time to really understand the current illustration…