Chapter 3
Taylor Approximation and M-files

3.1 Introduction

Approximation of a known function by a simpler expression is an important task in many areas of engineering, mathematics and science. Perhaps you have used the approximations $1 + x$ for e^x or x for sin(x). Both of these simple results are satisfactory as long as x is small with accuracy improving as x approaches zero. Function approximation is introduced early in calculus with the concept of the tangent line approximation – an application of the derivative. The familiar result, at $x = a$, is

$$f(x) \approx f(a) + f'(a)(x - a) \text{ for } x \text{ close to } a. \tag{3.1}$$

The approximations of e^x and sin(x) noted above are tangent line approximations with $a = 0$. Substitute $x = a$, into (3.1) to see that the approximation and function value have the same value at $x = a$. In addition, the first derivative of the approximation $\frac{d}{dx}[f(a) + f'(a)(x - a)]$ is also the same as the first derivative of the function at $x = a$, $f'(a)$. In other words, the tangent line approximation passes through the point $(\, a,\ f(a)\,)$ with slope $f'(a)$.

The tangent line approximation may also be called a Taylor polynomial approximation of degree one since $f(a) + f'(a)(x - a)$ is a first degree polynomial. A more complicated approximation is a quadratic Taylor polynomial in the form

$$T_2(x) = c_0 + c_1(x - a) + c_2(x - a)^2 \tag{3.2}$$

with three requirements

$$\begin{aligned} T_2(a) &= f(a), \\ T_2'(a) &= f'(a), \\ T_2''(a) &= f''(a). \end{aligned}$$

The third requirements asks that the second derivative of the quadratic Taylor polynomial be the same as the second derivative of the function at $x = a$. Using (3.2) to compute T_2, T_2', and T_2'' at $x = a$ shows that $c_0 = f(a)$, $c_1 = f'(a)$, and $2c_2 = f''(a)$. In other words

$$T_2(x) = f(a) + f'(a)(x - a) + \frac{1}{2}f''(a)(x - a)^2. \tag{3.3}$$

We may continue this scheme to find the general expression for a Taylor polynomial of degree n centered at a. The following Taylor formula is found in most calculus textbooks.

$$T_n(x) = \sum_{k=0}^{n} \frac{1}{k!} f^{(k)}(a)(x - a)^k \tag{3.4}$$

Taylor polynomials, centered at $a = 0$, for the two functions mentioned above, e^x and $\sin(x)$, are simple to derive. They are

$$e^x \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} \tag{3.5}$$

$$\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + (-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!} \tag{3.6}$$

As another example, compute the third degree Taylor polynomial for $f(x) = \sqrt{x+1}$ centered at $a = 1$. In order to evaluate,

$$T_3(x) = \sum_{k=0}^{3} \frac{1}{k!} f^{(k)}(1)(x-1)^k$$

we must compute the function and derivative values at $a = 1$. The following table provides the necessary values.

$$\begin{aligned}
f(x) &= (x+1)^{\frac{1}{2}}, & f(1) &= \sqrt{2} \\
f'(x) &= \frac{1}{2}(x+1)^{-\frac{1}{2}}, & f'(1) &= \frac{1}{2}2^{-\frac{1}{2}} = \frac{\sqrt{2}}{4} \\
f''(x) &= \frac{1}{2}(-\frac{1}{2})(x+1)^{-\frac{3}{2}}, & f''(1) &= \frac{1}{2}(-\frac{1}{2})2^{-\frac{3}{2}} = -\frac{\sqrt{2}}{16} \\
f'''(x) &= \frac{1}{2}(-\frac{1}{2})(-\frac{3}{2})(x+1)^{-\frac{5}{2}}, & f'''(1) &= \frac{1}{2}(-\frac{1}{2})(-\frac{3}{2})2^{-\frac{5}{2}} = \frac{3\sqrt{2}}{64}
\end{aligned}$$

Although $f(-1) = 0$, none of the derivatives exist for $x \leq -1$. Substituting into $T_3(x)$ gives the third degree Taylor polynomial for $\sqrt{x+1}$.

$$T_3(x) = \sqrt{2} + \frac{\sqrt{2}}{4}(x-1) - \frac{\sqrt{2}}{16 \cdot 2!}(x-1)^2 + \frac{3\sqrt{2}}{64 \cdot 3!}(x-1)^3, \; x > -1$$

As the previous example shows, finding a Taylor polynomial requires the systematic computation and evaluation of derivatives. MATLAB provides an interactive calculator which will compute and graph the Taylor polynomial of a given function. See **>> help taylortool**. You should be able to verify the expression for $T_3(x)$. To make use of the calculator enter the command **taylortool**.

In order to study the accuracy of Taylor polynomials, it is necessary to make use of Taylor's theorem found in most calculus textbooks. Brook Taylor (1685-1731) published his work in 1715. In simple terms, the theorem shows how differentiable functions may be approximated by polynomials.

Taylor's Theorem: Assume that $f(x)$ has $n+1$ continuous derivatives on a closed interval I. Then for any a and x in I,

$$f(x) = T_n(x) + R_n(x). \tag{3.7}$$

or $$f(x) = \sum_{k=0}^{n} \frac{1}{k!} f^{(k)}(a)(x-a)^k + R_n(x).$$

The term $R_n(x)$ is called the *remainder* and is given by

$$R_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(c)(x-a)^{n+1} \tag{3.8}$$

where c is a number (usually unknown) between x and a.

Since $R_n(x) = f(x) - T_n(x)$, the remainder is an expression which will allow us to analyze the error in a Taylor polynomial. Typically, absolute values are used. Unfortunately, the presence of the unknown value c will only allow us to compute a bound for the error.

$$| f(x) - T_n(x) | = | R_n(x) | \leq \frac{1}{(n+1)!} \cdot \max_I |f^{(n+1)}(x)| \cdot |x-a|^{n+1}. \tag{3.9}$$

Note the use of $\max_I |f^{(n+1)}(x)|$ as an upper bound for $|f^{(n+1)}(c)|$.

A standard problem is to find the degree of a Taylor polynomial approximating $f(x)$ on the interval I so that the error, $|f(x) - T_n(x)|$, is less than or equal to a prescribed tolerance, say TOL. The rightmost term in (3.9) is used to determine n by solving the inequality,

$$\frac{1}{(n+1)!} \cdot \max_I |f^{(n+1)}(x)| \cdot |x-a|^{n+1} \leq \text{TOL} \tag{3.10}$$

With the interval I specified and a given, we can easily determine an upper bound for the term $|x-a|$; however, finding the maximum of $|f^{(n+1)}(x)|$ is a difficult task unless it is possible to express the derivatives of $f(x)$ with a simple formula. For example, if $f(x) = e^{\alpha x}$, we may show that $f^{(n)}(x) = \alpha^n e^{\alpha x}$ and $f^{(n+1)}(x) = \alpha^{n+1} e^{\alpha x}$. In general, finding a simple relation for the n^{th} derivative of a function is not an easy task.

To illustrate the use of (3.10), let's find the degree of a Taylor polynomial which will approximate $f(x) = e^x$ on the interval $[-2, 2]$ with an error tolerance of 0.00001. In many cases a is chosen as the midpoint of the interval, 0 in this example, so that the interval $[-2, 2]$ may be written as $|x-0| \leq 2$. Substituting this bound into (3.10) gives

$$\frac{1}{(n+1)!} \cdot \max_{[-2,2]} |\frac{d^{n+1}}{dx^{n+1}}(e^x)| \cdot |2|^{n+1} \leq 0.00001.$$

Recall that all derivatives of e^x are identical to e^x itself; thus,

$$\frac{1}{(n+1)!} \cdot \max_{[-2,2]} |e^x| \cdot |2|^{n+1} \leq 0.00001.$$

Since e^x is an increasing function, the maximum value of $|e^x|$ on the interval $[-2, 2]$ is e^2 leading to the inequality

$$\frac{1}{(n+1)!} \cdot e^2 \cdot 2^{n+1} \leq 0.00001. \tag{3.11}$$

Inequalities involving factorials are usually difficult to solve analytically; however, using MATLAB's **factorial**, computations leading to a solution are easy. Beginning with a guess of $n = 11$, the left hand side of (3.11) is

```
>> exp(2)*2^(11+1)/factorial(11+1)
ans =
  6.3185e-005
```

The result is not less than 0.00001. A second try with $n = 12$ proves satisfactory.

```
>> exp(2)*2^(12+1)/factorial(12+1)
ans =
  9.7207e-006
```

$= 0.97207e{-}005 \leq 1.0000e{-}005 = 0.00001$

In other words, it will be necessary to use a Taylor polynomial of degree 12, $T_{12}(x)$, to reach our objective. As long as x is in the interval $[-2, 2\,]$ we can be assured that $|\, e^x - T_{12}(x)\,| \leq 10^{-5}$. In the next section, numerical computations will verify that $T_{12}(x)$ satisfies the error tolerance.

3.2 Function M-files

To verify the results at the end of Section 3.1 we must evaluate the Taylor polynomial. In other words, compute $T_{12}(x)$ at values of x in the interval $[-2, 2\,]$, and then confirm that our error tolerance is indeed met by computing the difference $|f(x) - T_{12}(x)|$. Most high-level programming languages allow for user defined subroutines, procedures or functions not part of the main language. MATLAB is a collection of functions, called M-files, which may be combined and executed by a user to accomplish a particular task. In fact, MATLAB allows users to write their own M-files which may incorporate other M-files, either MATLAB provided or user written. Multiple inputs and multiple outputs are permitted. An M-file is a text file of the form filename.m that contains MATLAB commands. The file extension, .m, denotes an M-file.

As an example, consider the Taylor expansion of e^x found in (3.5)

$$e^x \approx T_n(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}.$$

Our task is to write an M-file which will compute the value of $T_n(x)$ for a specified degree, n, and a particular value of x. In this situation there are two input values, n and x, and one output value, $T_n(x)$.

Function M-files have one major requirement. The function definition line must appear before executable commands. As a general rule, make it the first line of the file. See below. To create an M-file it is necessary to call up MATLAB's Editor/Debugger. This may be accomplished by entering the command >> **edit** or by using Window's features. Enter the following example M-file and save it as tay.m.

M-file tay.m

```
function z = tay(n,x)
%TAY Taylor expansion of exp(x)
z = 0;
for k = 0:n
        z = z +x^k/factorial(k);
end
exact = exp(x);
err = abs(exact-z);
disp([n,x,z,exact,err])
```

Line-by-line comments:

1. The function definition line. The word function must appear as shown. *tay* is the name of the function and must match the name of the file, *tay*. This is a MATLAB convention.
2. A comment line. Anything after % is not executed. Include something here to describe the M-file. For example, %TAY serves as easy reference to the file. >> **help tay** will give the text of this line. In addition, >> **lookfor tay** or >> **lookfor expansion** have the same result. Insert additional %-lines as needed.
3. Initializes z. (Old programming habits are difficult to forget.)
4. The start of a **for** loop. See >> **help for**. The variable k will take on integer values from 0 to n.
5. The Taylor polynomial constructed in recursive form.
6. The **end** of the for loop. See >> **help end**.
7. Computation of the exact value of e^x using MATLAB's exponential function **exp**.
8. Computation of error using absolute value, **abs**. (It is best not to use the variable error at this point. MATLAB uses the command **error** for special purposes.)
9. Prints the results of computations using MATLAB's **disp**. See >> **help disp**.

With the M-file saved as tay.m it is a simple task to evaluate the Taylor polynomial for e^x. Line 9 indicates how the output is displayed with $z = T_n(x)$ equal to the third term in the row. For example

```
>> tay(5,0.5);
  5.0000e+000  5.0000e-001  1.6487e+000  1.6487e+000  2.3354e-005
```

We are now in a position to evaluate the efforts of our analysis that predicted a Taylor polynomial of degree 12 was needed to meet the error tolerance. The following MATLAB command will evaluate Taylor polynomials for e^x of degree 10, 11 and 12 at $x = 2$.

```
>> for k = 10:12,tay(k,2);end
  1.0000e+001  2.0000e+000  7.3890e+000  7.3891e+000  6.1390e-005
  1.1000e+001  2.0000e+000  7.3890e+000  7.3891e+000  1.0083e-005
  1.2000e+001  2.0000e+000  7.3891e+000  7.3891e+000  1.5321e-006
```

As the degree of the polynomial (first column) increases the error at $x = 2$ (last column) decreases. Our selection of $T_{12}(x)$ is indeed verified; however, $T_{11}(x)$ comes very close to meeting the error tolerance requirement. Expect similar results at other values in the interval.

This is a good opportunity to experiment with MATLAB. The up arrow key will recall the previous MATLAB command. Edit the command and investigate what happens if x is replaced by 0, 1, and 3.

3.3 Graphing and Symbolic Taylor Polynomials

Graphs of functions are an integral part of engineering, science and mathematics. MATLAB provides various graphical features in both two and three dimensions. Perhaps the simplest way to graph or plot a function is aptly named **ezplot**. For example,

>> ezplot('sin(sqrt(x))',[0,2])

will draw a graph of sin(sqrt(x)) on the interval $[\ 0, 2\ ]$. See Figure 3.1 Your plot will show a blue graph in the figure window. Blue is the default color.

Many users will prefer more control over the visual image. This is available using MATLAB's **plot** command. The basic form **>> plot(x,y)** produces a piece wise linear plot of the data (x,y). If the data points are spaced closely enough together the result appears as a smooth image as seen in Figure 3.1.

To use the **plot** command, construct data vectors **x** and **y** as follows:

```
>> x = (0:.2:2)';
>> y = sin(sqrt(x));
>> plot(x,y)
```

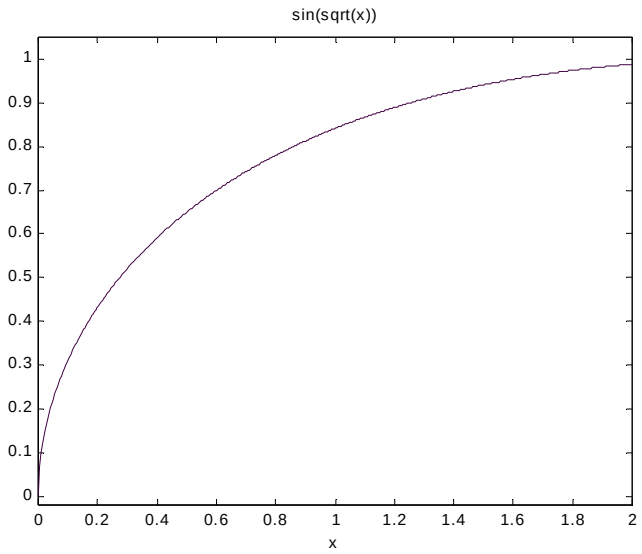

Figure 3.1 Ezplot Example

It may be necessary to experiment with the spacing between the x_i's. See also >> **help linspace**. The basic **plot** command has numerous options. Various aspects of these options will be considered later.

For repeated use of a function or functions it is convenient to have one or more M-files which contain various functions. In general, select names that are easy to recall or have a special meaning for a particular problem. For example, the M-file fct1.m may contain functions of one variable, fct2.m may be used for functions of two variables, and force.m may represent force functions in a mechanics problem. The following M-file contains three functions $x\sin(x)$, sin(sqrt(x)) and $\frac{1}{\sqrt{2\pi}}e^{-x^2/2}$. Since the first and third functions are preceded by the % symbol MATLAB ignores these lines. This means that the value of sin(sqrt(x)) will be returned when **fct1** is used.

M-file fct1.m

```
function y = fct1(x)
%FCT1 Generic function y = fct1(x)
%       y = x*sin(x);
        y = sin(sqrt(x));
%       y = exp(-x.^2/2)/sqrt(2*pi);
```

It is a good practice to end each function definition line with a semicolon to prevent unnecessary display of function values when **fct1** is called by another M-file. A second good practice is to include array operations so that **fct1** may be evaluated when x is a vector. For example, if **x*sin(x)** is made the active function in **fct1**, an error message will be generated when x is a vector. For example

```
>> fct1((0:.2:2)')
??? Error using ==> *
Inner matrix dimensions must agree.
Error in ==> C:\matlabR12\work\fct1.m
On line 3  ==>    y = x*sin(x);
```

Since x is a column vector, the standard multiplication on line 3 aborts the computation. Replacing ***** with **.*** makes the multiplication an array operation. Line 3 should be **y = x.*sin(x);**.

The command >> **plot(x,fct1(x))** will also draw the graph of sin(sqrt(x)) on the interval of x values. If the piece wise linear appearance of the graph is not satisfactory, replace the vector x with values spaced closer together, perhaps (0:.05:2)' . The **plot** command has the capability to draw more that one function in the same figure window. For example,

```
>> plot(x,fct1a(x),x,fct1b(x))
```

will plot both **fct1a** and **fct1b**. See also >> **help fplot**.

Returning to the major topic of this chapter, Taylor expansions, MATLAB has the capability to compute Taylor polynomials using the symbolic command, **taylor**. See >> **help**

taylor for information about the list of variables in **taylor**. For example, the following commands will compute a linear Taylor approximation for $\sqrt{x}$ centered at a.

```
>> syms a x
>> taylor(sqrt(x),2,a)
ans =
a^(1/2)+1/2/a^(1/2)*(x-a)
```

With care, it is possible to combine the symbolic and numerical features of MATLAB. Suppose we wish to find a symbolic quadratic Taylor polynomial centered at $a = 1$ for $f(x) = \sin(\text{sqrt}(x))$ and then plot both the polynomial and $f(x)$. You should be sure that sin(sqrt(x)) is still the active function in the appropriate M-file. Clearing all former variables and the command window itself is sometimes a needed housekeeping task. See **>> help clear** and **>> help clc**. In the following MATLAB text, the first command defines x as a symbolic variable. The second command constructs a symbolic quadratic polynomial, centered at 1, using **taylor**.

```
>> syms x
>> T2 = taylor(fct1(x),3,1)
T2 =
sin(1)+1/2*cos(1)*(x-1)+(-1/8*sin(1)-1/8*cos(1))*(x-1)^2
```

To evaluate the polynomial it is necessary to substitute into the symbolic expression. MATLAB's **subs** command us designed to replace symbolic variables with other symbols or numerical values. To replace the symbolic x in T2 with zero and compute the value we use the command

```
>> subs(T2,x,0)
ans =
  3.9860e-001
```

A graphical display will reveal the qualitative features of the Taylor approximation. The following four commands will generate a vector of values for the independent variable, xi, centered at 1, evaluate sin(sqrt(xi)), substitute into the symbolic T2, and plot both the function (solid line) and the Taylor approximation (dotted). For reference, the point (1, sin(sqrt(1))) is displayed in the same figure window. Within the **plot** command the terms **'k'**, **'k:'**, and **'ko'**, are used to plot a solid black line, a dotted black line and a black circle in the figure window.

```
>> xi = (0:.01:2)';
>> yi = fct1(xi);
>> ti = subs(T2,x,xi);
>> plot(xi,yi,'k',xi,ti,'k:',1,sin(sqrt(1)),'ko')
```

Only positive values are chosed for xi as a result of the square root within the sin function. The plot of T2 begins at the point $(0,\ 0.3986)$ as computed above.

The graphs in Figure 3.2 show that the Taylor approximation is too high for $x < 1$ and too low for $x > 1$. In either case, the approximation improves as x approaches one.

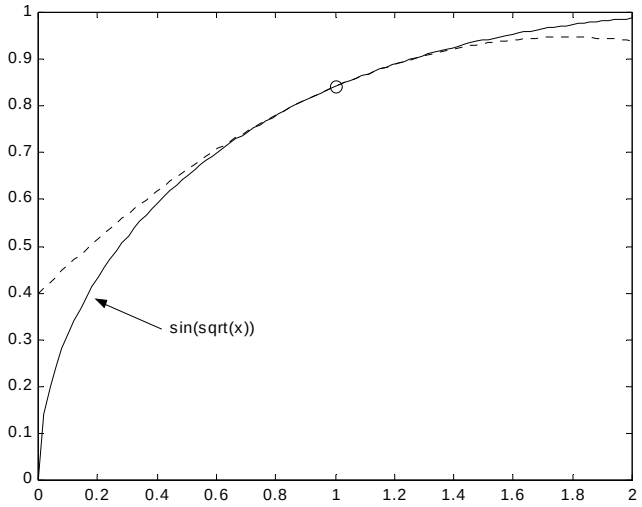

Figure 3.2 Taylor Expansion of sin(sqrt(x))

.

3.4 Problems

3-1. a. Find the Taylor polynomial of degree 4 for $f(x) = \sqrt{x}$ centered at $a = 1$.
b. Find the remainder term for part a.
c. Estimate the size of the remainder at $x = 1.5$.

3-2. Show that $1 + \frac{1}{2}x^2$ is a Taylor polynomial approximation centered at $a = 0$ for $\sqrt{1 + x^2}$.

3-3. a. Find the Taylor polynomial of degree 3 for $f(x) = \tan^{-1}(x)$ centered at $a = 0$.
b. Find the remainder term for part a.
c. Compare $T_3(0.3)$ with $\tan^{-1}(0.3)$.

3-4. Determine a Taylor polynomial of degree 4 for $f(x) = \ln(x)$ centered at $a = 2$. Check your result with MATLAB's **taylor** or **taylortool**. MATLAB uses **log** for the natural logarithm function, ln.

3-5. Determine a Taylor polynomial of degree 4 for $f(x) = \cos(x)$ centered at $a = \pi$. Check your result with MATLAB's **taylor** or **taylortool**.

3-6. The Taylor polynomial for $f(x) = e^{-x^2}$ centered at $a = 0$ may be obtained from (3.5) by replacing x with $-(x^2)$.
a. Write out the first six terms of the Taylor polynomial for $f(x)$.
b. The results of part (a) reveal an alternating series. It is possible to show that

$| f(\alpha) - T_8(\alpha) | < \alpha^{10}/5!$. Compute $| f(1) - T_8(1) |$ and verify that the error bound is met.

3-7. The remainder term for the Taylor representation of $\sin(x)$, (3.6), is given by $R_{2n+1}(x) = (-1)^n x^{2n+1} \cos(c)/(2n+1)!$.
a. Find the degree of a Taylor polynomial on the interval $[-\pi/2, \pi/2]$ which will insure that the error is less than or equal to $1.0000e{-}10$.
b. Using tay.m as a guide, verify the results of part (a).

3-8. Consider a Taylor polynomial centered at $a = 3$ for some $f(x)$. If we know $f^{(n+1)}(x) = x^{-2n-2}$, determine the smallest n so that $| R_n(x) | \leq 1.0000e{-}5$ on the interval $| x - 3 | \leq 2$.

3-9. Let $f(x) = \tan^{-1}(x)$, the arc tangent function (**atan** within MATLAB).
Use **taylor** to determine a symbolic Taylor polynomial of third degree centered at zero. Plot both the polynomial and $f(x)$ in the same figure on $[-2, 2]$. Comment on the graphical result.