Introduction to M-files

In this tutorial we learn the basics of working with ***M-files*** in MATLAB, so called because they must use ".m" for their filename extension. There are two types of M-files:

- A ***script*** is a collection of MATLAB commands which are executed just as if they had been typed in at the command line. When the script is finished running, any variables defined in the script are still available in the current MATLAB command-line environment.

  Executing MATLAB commands from within a script, even for relatively simple tasks, makes it much easier to reproduce results later, or to rerun the same calculations with different parameter settings. At the same time the script serves as a record of your activity and can serve as a useful starting point for future projects.

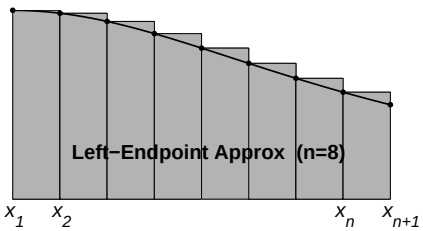
- A ***function*** accepts input arguments and returns output arguments, similar to functions in structured programming languages such as JAVA, C, and FORTRAN. Variables defined in a function are *local* to the function. The variables created in the function are not directly accessible from the command line once the function completes execution.

The Windows version of MATLAB provides a text editor for creating and modifying M-files. Throughout this document it is assumed that the reader is running the student version of MATLAB6 on the Windows platform. In this case the default working directory is typically `C:\matlab_sv12\work` and the M-file editor will save the M-files to this directory unless the user specifies otherwise.[1] The MATLAB principles will be introduced in the context of learning several methods of *numerical integration*. At the end of the tutorial we will have produced MATLAB code that can be used to approximate definite integrals.

Numerical integration

To approximate the definite integral $\int_a^b f(x)\,dx$ we first divide the interval $[a, b]$ into n subintervals, each of length $h = (b-a)/n$. The endpoints of the subintervals are denoted by the points $x_1, x_2, \ldots, x_{n+1}$, where $x_1 = a$ and $x_{n+1} = b$. We write y_i to denote the evaluation of $f(x)$ at node x_i, $y_i = f(x_i)$. The i^{th} subinterval is given by $[x_i, x_{i+1}]$, with $x_{i+1} - x_i = h$, for $i = 1, \ldots, n$. [*The endpoints are indexed as* $1, 2, \ldots, n+1$ *to coincide with the indexing of array elements in Matlab. In the textbook the grid points are indexed as* $0, 1, \ldots, n$.]

The ***left-endpoint*** approximation uses n rectangles of width h to approximate the integral. The height of the rectangle for interval $[x_i, x_{i+1}]$ is given by $f(x_i)$, the value of the function at the left endpoint. The area of the i^{th} rectangle is $A_i = hf(x_i) = hy_i$. Summing over the n rectangles yields the approximation:

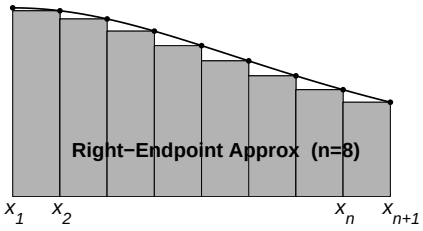


$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} hy_i \;=\; h\big(y_1 + y_2 + \cdots + y_n\big) \doteq L_n$$

[1] Use the `pwd` command to confirm the current working directory. To change the current directory, enter `cd` *dir_name* from the command line, or choose **MATLAB → Current Directory** from the MATLAB *Launch Pad*.

For the ***right-endpoint*** approximation the height of the rectangle for the interval $[x_i, x_{i+1}]$ is given by $f(x_{i+1})$, the value of the function at the right endpoint.

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} h y_{i+1} \;=\; h\big(y_2 + y_3 + \cdots + y_{n+1}\big) \doteq R_n$$



The ***trapezoidal rule*** uses n trapezoids of width h to approximate the definite integral. The top edge of the trapezoid for the subinterval $[x_i, x_{i+1}]$ is the line segment connecting the two points (x_i, y_i) and (x_{i+1}, y_{i+1}). The area of the i^{th} trapezoid is given by $A_i = h(y_i + y_{i+1})/2$. Summing over the n trapezoids,

$$\int_a^b f(x)\,dx \;\approx\; \sum_{i=1}^{n} \frac{h}{2}\big(y_i + y_{i+1}\big) \;=\; \frac{h}{2}\big(y_1 + 2y_2 + 2y_3 + \cdots + 2y_n + y_{n+1}\big) \;\doteq\; T_n \tag{1}$$

The trapezoidal rule is equivalent to averaging the left-endpoint and right-endpoint approximations,

$$T_n \;=\; \big(L_n + R_n\big)/2\,. \tag{2}$$

Creating a MATLAB script

We first write a MATLAB script that calculates the left-endpoint, right-endpoint, and trapezoidal approximations for a particular definite integral. After the script is verified to be working, we will convert it into a MATLAB function and later incorporate two additional approximations, the midpoint rule and Simpson's rule.

As a test case for our numerical integration, we will approximate the definite integral of $f(x) = 4/(1+x^2)$ on the interval $[0, 1]$. In section 3.6 of the Stewart text we used implicit differentiation to show that the derivative of $\arctan x$ is the function $1/(1+x^2)$. In the language of section 4.9 we say $\arctan x$ is an *antiderivative* of the function $1/(1+x^2)$. Applying the *evaluation theorem* in section 5.3,

$$\int_0^1 \frac{4}{1+x^2}\,dx \;=\; 4\arctan x\Big|_0^1 \;=\; 4\left(\frac{\pi}{4} - 0\right) \;=\; \pi \tag{3}$$

After starting up MATLAB enter the command **pwd** to confirm the name of the current working directory. This current directory is where your M-files will be stored.

```
>> pwd                                  % print name of working directory
ans =
C:\matlab_sv12\work                     % M-files will be saved here
```

Note: When copying commands from this document into your own M-files or to the command line, the percent sign (%) marks the rest of the line as a comment. It is not necessary to copy the comments.

To create a new M-file, select **File** → **New** → **M-file** from the menu bar. The M-file editor starts up with an empty window. Copy the following MATLAB commands into the new M-file.

```
a = 0.0;  b = 1.0;                      % integrate f(x) from a=0 to b=1
n = 8;                                  % number of subintervals; n EVEN
h = (b - a) / n;                        % width of subintervals

clear x y;                              % remove current values of x and y
x(1) = a;                               % first node is the left endpoint
y(1) = 4 / (1 + x(1)^2);
```

```
for i = 1:n                          % i = 1, 2, ..., n-1, n
  x(i+1) = x(i) + h;                 % x(2)=x(1)+h; x(3)=x(2)+h; ...
  y(i+1) = 4 / (1 + x(i+1)^2);       % y(i) = 4 / (1 + x(i)*x(i))
end

Ln = 0;  Rn = 0;                     % initialize the sums to 0
for i = 1:n
  Ln = Ln  + y(i);                   % left-endpoint
  Rn = Rn + y(i+1);                  % right-endpoint
end
Ln = h*Ln;                           % left-endpoint approximation
Rn = h*Rn;                           % right-endpoint approximation
Tn = (Ln + Rn) / 2;                  % trapezoidal approx; equation (2)

fprintf(1, 'n = %d\n', n);           % display some results
fprintf(1, 'Ln = %.12f  error= %.2e\n', Ln, Ln - pi);
fprintf(1, 'Rn = %.12f  error= %.2e\n', Rn, Rn - pi);
fprintf(1, 'Tn = %.12f  error= %.2e\n', Tn, Tn - pi);
```

The function `fprintf` is for writing ***formatted data*** to a file.[2] To save the new M-file to the disk, choose **File** → **Save** from the editor's pull-down menu and follow the dialog box. To test the new script, enter the name of the script from the *Command Window*. This writeup assumes the script has been named `m2s.m`. If MATLAB reports an error, use the editor to make the corrections, save the revised file, and rerun `m2s`. Once the program is working correctly you should see the following output:

```
>> m2s
n = 8
Ln = 3.263988494491  error= 1.22e-01
Rn = 3.013988494491  error= -1.28e-01
Tn = 3.138988494491  error= -2.60e-03
```

Converting the script to a function

Increasing the value of `n` in the script `m2s.m` will improve the accuracy of the numerical integration. However, each new value of `n` requires editing the M-file, changing the value of `n`, and saving the revised file. Here it will be helpful to first convert the script into a function so that the number of subintervals is controlled through the value of an input argument. The function will require a single input argument, the number of subintervals `n`, and return a single output argument, the trapezoidal approximation for the definite integral. Note that `n` must be a positive *even* integer.

Rather than overwrite the working script, first create a new file `m2f.m` that is just a copy of the file `m2s.m`. In this new file, delete the line `n = 8` and, *at the very top* of the file, add the following line of Matlab code:

```
function Tn = m2f(n)
```

The first line of the file identifies `m2f` as a function that can be called with a single input argument. The input value will be copied to the variable `n` inside the function. The contents of the variable `Tn` are returned to the

[2] With the first argument of `fprintf` set to `1` the output goes to the *Command Window*. The second argument is a string for specifying the output format. For example, `%d` specifies decimal integer output, `%.12f` specifies real numbers with twelve digits to the right of the decimal point, `%.2e` specifies exponential notation with two digits to the right of the decimal point (three significant digits), and `\n` produces a linefeed.

program that called `m2f` (just the *Command Window* in this case). After saving these changes, run the new function for different values of `n`. For example, pass the number 8 to duplicate the earlier results.

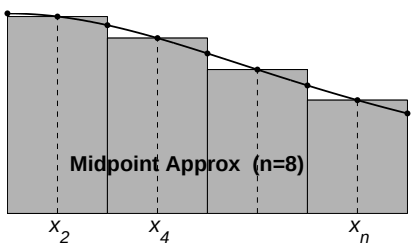
```
>> m2f(8)
n = 8
Ln = 3.263988494491  error= 1.22e-01
Rn = 3.013988494491  error= -1.28e-01
Tn = 3.138988494491  error= -2.60e-03
ans =
    3.1390
```

Calling the function with the command **`S = m2f(8)`** will save the output in the variable `S`.

***Exercise 1*:** Run your function `m2f.m` using increasing values of the input argument `p`. What is the smallest value of `n` that makes the absolute value of the error in `Tn` less than 10^{-6}? Less than 10^{-8}?

The midpoint rule

For the same collection of n subintervals with endpoints $x_1, x_2, \ldots, x_n, x_{n+1}$, the ***midpoint*** approximation uses $n/2$ rectangles of width $2h$ to approximate the integral. For $k = 1, \ldots, n/2$, the k^{th} subinterval is $[x_{2k-1}, x_{2k+1}]$ with midpoint x_{2k}. The height of the rectangle for this interval is taken to be $f(x_{2k})$, the value of the function at the midpoint of the interval, and the area of the k^{th} rectangle is $A_k = 2hy_{2k}$.



Summing the areas from the $n/2$ rectangles yields the midpoint approximation:

$$\int_a^b f(x)\,dx \;\approx\; \sum_{k=1}^{n/2} 2hy_{2k} \;=\; 2h\big(y_2 + y_4 + \cdots + y_{n-2} + y_n\big) \;\doteq\; M_n \tag{4}$$

To include the midpoint approximation in your MATLAB function, add the following code[3] to the function `m2f.m`. This new `for` loop can be added just after the calculation of `Tn`.

```
Mn = 0;                          % initialize the sum to 0
for k = 1:n/2                    % k = 1, 2, ..., n/2
  Mn = Mn + y(2*k);              % see equation (4)
end
Mn = 2*h*Mn;                     % midpoint approximation
```

Include another `printf` statement at the end of the M-file.

```
fprintf(1, 'Mn = %.12f  error= %.2e\n', Mn, Mn - pi);
```

***Exercise 2*:** Run `m2f.m` and compare the accuracy of the midpoint and trapezoidal approximations for several values of the input argument `n`. Comment on your observations.

Simpson's rule

In each of these integration methods the approximation formula is just the definite integral for a simple function that approximates $f(x)$. For example, the trapezoidal rule uses a piecewise linear function. On

[3] It is ALWAYS a good idea to first make a copy of the existing program before adding new features. If you are unable to get the revisions to work properly, you can still return to a working version of the program.

each subinterval $[x_i, x_{i+1}]$ we integrate a linear function in place of $f(x)$. The approximating function is *piecewise* because the equation of the linear function changes from one subinterval to the next. Each of the other three methods use piecewise constant functions to approximate $f(x)$.

With this point of view, we look to improve the accuracy of the numerical integration by approximating $f(x)$ with a higher order polynomial. On each subinterval $[x_{2k-1}, x_{2k+1}]$ of length $2h$, there is exactly one second-order polynomial that passes through the three points (x_{2k-1}, y_{2k-1}), (x_{2k}, y_{2k}), and (x_{2k+1}, y_{2k+1}). The area under this polynomial is given by $A_k = 2h(y_{2k-1} + 4y_{2k} + y_{2k+1})/6$, for $k = 1, 2, \ldots, n/2$. Summing these areas (there are $n/2$ subintervals of length $2h$) results in ***Simpson's Rule***:

$$\int_a^b f(x)\,dx \;\approx\; \sum_{k=1}^{n/2} \frac{h}{3}\big(y_{2k-1} + 4y_{2k} + y_{2k+1}\big) \;=\; \frac{h}{3}\big(y_1 + 4y_2 + 2y_3 + 4y_4 + \cdots + 4y_n + y_{n+1}\big) \;\doteq\; S_n \quad (5)$$

Simpson's rule can also be expressed as a weighted average of the trapezoidal and midpoint rules.

$$S_n \;=\; \big(2T_n + M_n\big)/3 \quad (6)$$

Exercise 3 **:** Add the calculation for Simpson's rule from equation (6) to the function `m2f.m`, including an additional `fprintf` statement and error calculation. Change the function so it returns the value of the Simpson approximation rather than the trapezoidal approximation. Compare the accuracy of the different integration methods. What is the smallest value of n that makes the absolute value of the error in S_n less than 10^{-6}? Less than 10^{-8}?

Exercise 4 **:** Modify the function `m2f.m` to estimate the definite integral $\int_0^4 e^{-x^2} dx$ using Simpson's approximation. Calculate the Simpson approximation for increasing values of $n = \{2, 4, 8, 16, \ldots\}$ until the difference $|S_n - S_{n/2}| < 10^{-10}$. *In this exercise you must change in the M-file both the value of b and the expressions for calculating* $y = f(x)$. The error calculations (`Tn - pi`,...) are no longer valid for this exercise. The following code shows how one might proceed to call `m2f` once the function has been revised.

```
>> format long;                 % display more significant digits
>> s8 = m2f(8)                  % display Simpson approx with n = 8
ans = ...
>> s16 = m2f(16)                % display Simpson approx with n = 16
ans = ...
>> abs(s16 - s8)                % display the difference |S_16 - S_8|
ans = ...
>> ...                          % continue until difference < 1.0e-10
```

Record the final value of n, the estimate S_n with at least 12 significant digits, and the difference $|S_n - S_{n/2}|$.

Exercise 5 **:** If $f(x)$ is any third-order polynomial, show that Simpson's rule with $n = 2$ is exact for the definite integral $\int_{-a}^{a} f(x)\,dx$. In other words, verify the following equality:

$$\int_{-a}^{a} (Ax^3 + Bx^2 + Cx + D)\,dx \;=\; \frac{h}{3}\big(y_1 + 4y_2 + y_3\big)$$

First calculate the integral on the left-hand side in terms of the parameters A, B, C, D, and a. Then calculate the right-hand side where $y_i = f(x_i) = Ax_i^3 + Bx_i^2 + Cx_i + D$. In evaluating the right-hand side, you must first identify the value of h, and the nodes x_1, x_2, x_3, in terms of the parameter a. You should be able to show that the two sides are equal for any choice of the constants A, B, C, D, and a.

[*This is one reason for the popularity of Simpson's rule. We started by approximating the integrand with a piecewise second-order polynomial and the method turns out to be exact for all third-order polynomials.*]