Assignment 1 — 2.086/2.090 Spring 2013

Released: *Thursday, 7 February, at 5 PM.*

Due: *Friday, 22 February, at 5 PM.*

Upload your solution as a zip file "`YOURNAME_ASSIGNMENT_1`" which includes the script for each question in the proper format as well as any MATLAB functions (of your own creation) your scripts may call. You should also include in your folder the contents of the `grade_o_matic` folder for Assignment 1.

Instructions

Before embarking on this assignment you should

(1) Complete the Textbook reading for Unit I

(2) Execute ("cell-by-cell") two MATLAB Tutorials

a tutorial on MATLAB Basics (environment, data types, elementary operations, scripts);

a tutorial on Single-Index Arrays.

(3) Download the `Templates_Assignment_1` folder. This folder contains a script template for each of the questions; you should use these templates as the point of departure for your own codes. Note these script templates, `AxQy_Template.m` for Question `y` of Assignment `x`, will both help you follow the required format but also frequently provide some guidance and hints (with "?" indicating the additional pieces that you must provide). In later assignments we will provide function templates as well.

We indicate here several general format and performance requirements:

(a.) Your script for Question `y` of Assignment `x` *must* be a proper MATLAB ".m" script file and *must* be named `AxQy.m`. (Hence to start you should "save as" `AxQy_Template.m` to `AxQy.m`.)

(b.) For each question and hence each script we will identify *input parameters*; we will also identify corresponding *allowable instances* for the parameters — the parameter values or "parameter domains" for which the code must work. More strictly speaking, inputs are typically defined for a function, which we will learn about shortly, rather than a script; however, less strictly speaking, we can also define inputs for scripts — variables which must be assigned values (in the workspace) before the script is executed. Your script should perform correctly for any allowable instances.[1] We may similarly and informally associate to each script an *output* or *outputs*.

[1] Note that most properly, and certainly if you write code for third parties, you should always confirm that any inputs constitute an allowable instance and if not the code should set an appropriate "error flag." We will not require you to include such error flags in your codes: we will only grade your code based on allowable instances as prescribed in the problem statement.

(c.) We shall define the inputs and outputs and associated specific MATLAB variable names for each question as we proceed. The input variables must be assigned *outside* your script (of course before the script is executed) — *not* inside your script — in the workspace; all other variables required by the script must be defined *inside* the script. Hence you should test your scripts in the following fashion: `clear` the workspace; assign the input variables in the workspace; run your script.

(d.) Finally, we ask that in the submitted version of your script you suppress all display by placing a "`;`" at the end of each line of your script. (Of course during debugging you will often choose to display many intermediate and final results.)

(4) Download the `grade_o_matic_Assignment_1` folder and add this folder to your path. The `grade_o_matic_A1` software (and more generally `grade_o_matic_Ax` for Assignment `x`) will be used by both you and by the graders: in the former case, `grade_o_matic_A1` runs your code for a (relatively small) set of "student instances" of the input parameters; in the latter case, `grade_o_matic_A1` runs your code for a (larger and more diabolical) set of "grader instances" of the input parameters. In student (respectively, grader) mode, `grade_o_matic_A1` assigns and displays credit for each auto-gradable question[2] if and only if your codes work correctly for all the "student instances" (respectively, "grader instances") of the parameter; in questions with several outputs, points are assigned *pro rata* for each correct output.[3]

In "student" mode, `grade_o_matic_A1` is intended to verify that your inputs and outputs are in the proper format and furthermore to confirm necessary *but not sufficient* conditions for correct performance. You can run `grade_o_matic_A1` for any single (auto-gradable) question, say Question 2, as `grade_o_matic_A1(2)`; if you run `grade_o_matic_A1` with no arguments then all auto-gradable questions will be assessed. We strongly suggest that when you have completed the assignment and placed all your scripts and functions in `YOURNAME_ASSIGNMENT_1` and **before** you upload your solution you should run `grade_o_matic_A1` (from your `YOURNAME_ASSIGNMENT_1` folder) for final confirmation that all is in order.

In "grader" mode, `grade_o_matic_A1` will determine your grade for each question: we reiterate that you will earn credit for each question (*pro rata* for multiple outputs) if and only if your code works correctly for all the "grader instances" of the input parameters.

It should be clear that you should not rely exclusively on the student mode ("student instances") `grade_o_matic_A1` assessment to determine if your code is working properly. You should confirm independently the "logic" of your codes and also conduct your own tests for carefully chosen instances — designed to flush out flaws either for special cases or more generic cases. Note the parameter domains — allowable instances — will most often be intervals or regions and hence it will typically not be possible to conduct exhaustive brute force testing. In 2.086 the consequences of a faulty code or poorly prepared input parameters, and in particular insufficient testing, are relatively benign: you might receive a low grade on a question due to a bug not detected by the "student instances" but subsequently caught by the "grader instances"; more optimistically, you might receive unearned credit for a bug caught by neither the "student instances" or the "grader instances." In contrast, in real life, the failure of a code or a computational method can lead to engineering disasters.

[2] Some questions, such as questions with multiple choice answers (in student mode) or questions which involve graphics (in student or grader mode), are not auto-gradable.

[3] Offered by Mathworks, `http://www.mathworks.com/matlabcentral/about/cody/` is a multiplayer MATLAB programming "game" — a more amusing example of automated "grading."

Cartoon by Sidney Harris removed due to copyright restrictions.

Questions

Questions 1–5 are based on the Fibonacci sequence. We recall that the Fibonacci sequence is defined by

$$F_n = \begin{cases} 1, & n = 1; \\ 1, & n = 2; \\ F_{n-1} + F_{n-2}, & n \geq 3, \end{cases}$$

which is strictly increasing for $n > 2$. In actual practice we will consider only a finite number of terms in this sequence. **Note**: in Question 1, Question 2, and Question 3, you should not use any single-index or double-index arrays as these questions are intended to exercise your "scalar" skills.

1. (8 points) Write a script which, given a positive integer N, calculates F_N. Our input parameter is N and must correspond in your script to MATLAB variable `Nfib`; the allowable instances, or input parameter domain, is given by $1 \leq N \leq 25$. Our output is F_N and must correspond in your script to MATLAB (scalar) variable `F_Nfib`: your script must assign the result F_N — the N^{th} term in the Fibonacci sequence as defined in the problem statement above — to the MATLAB variable `F_Nfib`.

   We ask that you use a `for` loop. Note that at any given time you need only store the three active members of the sequence, which you will re-assign — shift, or "shuffle" — appropriately. Figure 2 may help you visualize this shuffle.

2. (8 points) Write a script which, given a positive integer $F_{\text{upper limit}}$, finds N^* such that $F_{N^*} \leq F_{\text{upper limit}}$ and $F_{N^*+1} > F_{\text{upper limit}}$. Note that from the monotonicity properties of the Fibonacci sequence we may interpret our task more intuitively: N^* is the number of terms in the sequence which are less than or equal to $F_{\text{upper limit}}$. Our input is $F_{\text{upper limit}}$ and must

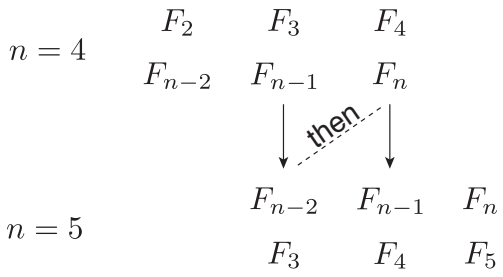

Figure 2: Visualization of the "shuffle" sequence.

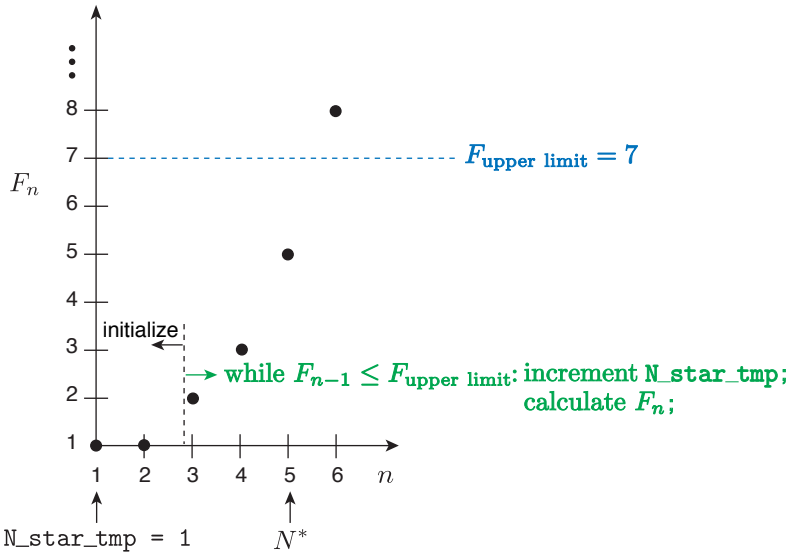

Figure 3: Visualization of a `while` loop and a "counter" variable.

correspond in your script to MATLAB variable `F_upper_limit`; the allowable instances, or input parameter domain, is given by $2 \leq F_{\text{upper limit}} \leq 70,000$. Our output is N^* and must correspond in your script to MATLAB (scalar) variable `N_star`: your script should assign the result N^* — as defined in the problem statement above — to the MATLAB variable `N_star`.

We ask that you use a `while` loop and a "counter" variable. Figure 3 may help you visualize the process.

3. (8 points) Write a script which, given any positive integer N, finds the sum S_N of those F_n, $1 \leq n \leq N$, which are divisible by either 2 or 5:

$$S_N = \sum_{n=1}^{N} \begin{cases} F_n \text{ if } F_n \text{ is divisible by 2 or 5} \\ 0 \text{ otherwise .} \end{cases}$$

For example, for say $N = 3$, the only term in the Fibonacci sequence to be included in the sum will be $F_3 = 2$ and hence S_N shall equal 2. Our input is N and must correspond in your script to MATLAB variable `Nfib`; the allowable instances, or input parameter domain, is given by $1 \leq N \leq 25$. Our output is S_N and must correspond in your script to MATLAB (scalar) variable `fibsum`: your script should assign the result S_N — as defined in the problem statement above — to the MATLAB variable `fibsum`.

We ask that you use a `for` statement and a summation (or "accumulation") variable. You should find the MATLAB built-in function `mod` helpful to determine divisibility.

4. (8 points) Write a script which, given a positive integer N, constructs a single-index (row) array `Ffib` which contains the first N terms of the Fibonacci sequence — $[F_1, F_2, \ldots, F_N]$. For example, for $N = 3$, `Ffib` will be given by $[1, 1, 2]$. Our input is N and must correspond in your script to MATLAB variable `Nfib`; the allowable instances, or input parameter domain, is given by $1 \leq N \leq 25$. Our output is $[F_1, F_2, \ldots, F_N]$ and must correspond in your script to MATLAB single-index row array `Ffib`: your script should assign the result $[F_1, F_2, \ldots, F_N]$ — the first N terms in the Fibonacci sequence as defined in the problem statement above — to the MATLAB variable `Ffib`; note that `length(Ffib)` will be `Nfib`.

   We ask that you use a `for` loop and a single-index array `Ffib` initialized to all zeros.

5. (8 points) Write a script which, given a positive integer $F_{\text{upper limit}}$, constructs a single-index (row) array `Ffib_star` which contains the first N^* terms of the Fibonacci sequence — $[F_1, F_2, \ldots, F_{N^*}]$ — for N^* such that $F_{N^*} \leq F_{\text{upper limit}}$ and $F_{N^*+1} > F_{\text{upper limit}}$. Our input is $F_{\text{upper limit}}$ and must correspond in your script to MATLAB variable `F_upper_limit`; the allowable instances, or input parameter domain, is given by $2 \leq F_{\text{upper limit}} \leq 70,000$. Our output is $[F_1, F_2, \ldots, F_{N^*}]$ and must correspond in your script to MATLAB single-index (row) array `Ffib_star`: your script should assign the result $[F_1, F_2, \ldots, F_{N^*}]$ — as defined in the problem statement above — to `Ffib_star`; note that `length(Ffib_star)` will be N^* and is not known *a priori*.

   Use a `while` loop and a single-index array which is initialized and subsequently "grown" by horizontal concatenation.[4]

6. (8 points) Write a script which, given a real number r and an integer N, calculates the sum of a geometric series with $N + 1$ terms,

$$G_N = \sum_{i=0}^{N} r^i = 1 + r + r^2 + r^3 + \cdots + r^N \ .$$

   Note that we may also write the first term in the sum as $1 = r^0$, which shall prove useful subsequently. Our inputs are N and r which must correspond in your script to MATLAB variables `Ngeo` and `rgeo`, respectively; the corresponding allowable instances, or input parameter domains, are $1 \leq N \leq 20$ and $0 < r < 1$, respectively. Our output is G_N and must correspond in your script to MATLAB variable `G_Ngeo`: your script should assign the result G_N — as defined in the problem statement above — to the MATLAB variable `G_Ngeo`.

   Do *not* use a `for` loop or a `while` loop. Instead, we ask that you use MATLAB built-in function `ones`, the `colon` operator, array "dotted" operators, and the MATLAB built-in function `sum`. In fact, a single line of MATLAB code should suffice, though we suggest you break this into several lines for improved readability.

7. (10 points) Write a script which, given a vector of points `x_vec` $= [x_1, x_2, \ldots, x_N]$ and a point x , finds the index i^* such that $x_{i^*} \leq x$ and $x_{i^*+1} > x$. Our inputs are `x_vec` and

[4] When we get to double-index arrays we will emphasize the importance of initializing arrays (with `zeros` and later `spalloc`). Initialization ensures that you control the size/shape of the array and also is the most efficient way to allocate memory. On the other hand, concatenation can be very useful in dynamic contexts (in which array size may not be known *a priori*) or in situations in which a large array is most easily expressed in terms of several smaller arrays. But use concatenation sparingly in particular for very large arrays.

x. The input `x_vec` must correspond in your script to a MATLAB single-index row array `x_vec` of `length` N; as regards allowable instances, you may assume that the points in `xvec` are distinct and ordered, $x_i < x_{i+1}$, $1 \leq i \leq N-1$, but *you should not assume in this or subsequent questions that the points are equi-distantly spaced*. The input x must correspond in your script to MATLAB (scalar) variable `x`; the allowable instances, or input parameter domain, is $x_1 \leq x < x_N$. (Note that N is not an input.) Our output is i^* and must correspond in your script to MATLAB (scalar) variable `i_star`: your script should assign the result i^* — as defined in the problem statement above — to the MATLAB variable `i_star`.

Do *not* use a `for` loop or a `while` loop. Instead, we ask that you use array relational operators, the MATLAB built-in `find`, and then the MATLAB built-in function `max` (or `length`).[5] You may test your code for various choices of `x_vec` and `x`: possible choices for `x_vec` include (for some given `N`) `(1/(N-1))*[0:N-1]`, `linspace(-3.,1.,N)`, and `sort(rand(1,N))` (this last option will create points which are not equidistantly spaced).

8. (10 points) In this question we are interested in the interpolant of some univariate function $f(x)$. Write a script which, given a vector of distinct ordered points `x_vec` $= [x_1, x_2, \ldots, x_N]$, a vector of associated function values `f_vec` $= [f(x_1), f(x_2), \ldots, f(x_N)]$, and a point x, finds the piecewise-linear interpolant of f at x, $(\mathcal{I}f)(x)$ as defined in Example 2.1.4 of the textbook. (Note for our linear interpolant the evaluation points coincide with the segment (end)points, $\tilde{x}_i = x_i$, $1 \leq i \leq N$.) Our inputs are `x_vec`, `f_vec`, and x. The inputs `x_vec` and x must correspond in your script to MATLAB variables `x_vec`, `x`; these inputs are already described in Question 7. The input `f_vec` must correspond in your script to a MATLAB single-index row array `f_vec` of `length` N; we do not specify a corresponding input parameter domain but if the interpolant is to be convergent for appropriate choices of `x_vec` we must confirm that the function f which engenders `f_vec` is suitably smooth.[6] Our output is $(\mathcal{I}f)(x)$ and must correspond in your script to MATLAB variable `Interp_f_h`: your script should assign the result $(\mathcal{I}f)(x)$ — as defined in the problem statement above — to the MATLAB variable `Interp_f_h`.

We ask that you build your interpolation script on your "find segment" script of Question 7. You may test your code for various choices of `x_vec` (and `x`) as described in Question 7. To form `f_vec`, choose some appropriate function $f(x)$ and then form `f_vec(i)` $= f(x_i)$, $1 \leq i \leq N$. Note the latter can be facilitated by dotted operators and MATLAB built-in functions: for example, for $f(x) = x^p$ for some given p, you may write `f_vec = x_vec.^p`; for $f(x) = e^x$, you may write `f_vec = exp(x_vec)`.

A general comment on testing: In problems in which you implement a numerical method which itself represents an approximation it can be difficult to confirm that the implementation is correct — since it is possible to conflate bugs with legitimate numerical errors. Hence it is always good to start with a very small case which you can check by hand. Then next proceed to an instance in which the numerical errors are zero (or round-off) — this might require some art. Then finally proceed to a case in which you can anticipate the convergence rate —

[5] Note the algorithm suggested here requires $O(N)$ FLOPs. A better (but more complicated) approach can be developed which requires only $O(\log N)$ FLOPs.

[6] In this case we clearly see why exhaustive testing is impossible: the dimensionality of the input parameter domain can be very high. More importantly, this example illustrates the distinction between a code which executes the intended steps correctly and a code which executes the intended steps correctly *and also yields a suitably accurate answer to the question posed*; the former relates to implementation while the latter relates to the computational method implemented and the allowed instances (as well as the implementation in some cases).

this will catch more subtle errors. You can apply this approach to Question 8, Question 9, and Question 10.

9. (10 points total) In this question we are interested in the first derivative of some univariate function $f(x)$. Write a script which, given a vector of distinct ordered points `x_vec` $= [x_1, x_2, \ldots, x_N]$ and a vector of associated function values `f_vec` $= [f(x_1), f(x_2), \ldots, f(x_N)]$, finds

   (a.) (5 points) `fprime_h_vec`, the forward difference approximation to $f'(x)$ at the set of points x_i, $1 \leq i \leq N_1$ — $[f'_h(x_1), f'_h(x_2), \ldots, f'_h(x_{N-1}]$;

   (b.) (5 points) `xpos_fprime_h_vec`, the set of points x_i at which $f'_h(x_i)$ is positive.

   The forward difference approximation $f'_h(x)$ is defined in Example 3.1.1 of the textbook; in our case here, the evaluation points coincide with the segment (end)points, $\tilde{x}_i = x_i$, $1 \leq i \leq N$. Note that if $f'_h(x_i) > 0$ for all x_i, $1 \leq i \leq N-1$, then `xpos_fprime_h_vec` will be identical to `x_vec`; conversely, if $f'_h(x_i) \leq 0$ for all x_i, $1 \leq i \leq N-1$, then `xpos_fprime_h_vec` will be the "empty matrix." Our inputs are `x_vec` and `f_vec` which must correspond in your script to MATLAB variables `xvec` and `fvec` (already described in Question 8), respectively. Our outputs are `fprime_h_vec` and `xpos_fprime_h_vec` as defined above and must correspond in your script to MATLAB single-index row arrays `fprime_h_vec` and `xpos_fprime_h_vec`, respectively; note that `length(fprime_h_vec)` is $N-1$ but `length(xpos_fprime_h_vec))` is not known *a priori* and will be determined by the particular instance studied.

   We ask that you use dotted arithmetic operators and array relational operators. You may test your code for various choices of `x_vec` and `f_vec` as described in Question 7 and Question 8.

10. (10 points) In this question we are interested in the integral of some univariate function $f(x)$. Write a script which, given a vector of distinct ordered points `x_vec` $= [x_1, x_2, \ldots, x_N]$ and a vector of associated function values `f_vec` $= [f(x_1), f(x_2), \ldots, f(x_N)]$, finds the trapezoidal-rule approximation to the integral of f over the interval $[x_1, x_N]$, I_h as defined in Example 7.1.4 of the textbook. Our inputs are `x_vec` and `f_vec` which must correspond in your script to MATLAB variables `x_vec` and `f_vec` (already described in Question 8), respectively. Our output is I_h and must correspond in your script to MATLAB (scalar) variable `Integral_f_h`: your script should assign the result I_h — as defined in the problem statement above — to the MATLAB variable `Integral_f_h`.

    We ask that you use dotted arithmetic operators. You may test your code for various choices of `x_vec` and `f_vec` as described in Question 7 and Question 8.

    CHALLENGE 1: (optional: *no credit*) Expand your scripts of Questions 8, 9, and 10 to provide an estimate for the error in the interpolant, derivative, and integral, respectively, for the case in which $f(x)$ is very smooth and the points in `x_vec` are equi-distant, $x_{i+1} - x_i = h$, $1 \leq i \leq N-1$.

11. (6 points) (Driscoll 5.1[7]) Write a script to plot, on a single figure, the functions $\sinh x$, $\cosh x$, $\tanh x$, and e^x over the interval $-1 \leq x \leq 1$. Plot each function at 100 equi-spaced points in x (over the interval $-1 \leq x \leq 1$); choose for sinh a red line and no mark at each point, cosh

[7] Derived from *Learning* MATLAB by Tobin Driscoll.

a black dotted line with no mark at each point, tanh a blue line with a circle at each point, and e^x no line with a green $\times$ at each point; create a legend for the four curves; label your axes as appropriate; and provide the figure with a title. There are no inputs or outputs for this question.

Create the set of points in x and the corresponding vectors of function values as described in Question 8. See Section 5.4 of the textbook for plotting examples.

12. (6 points total) A surface of revolution (see Figure 4) is described by a function $f(x)$ for $0 \leq x \leq 1$: here x represents the axial coordinate and $f(x)$ represents the radius. We shall suppose that $f(x)$ is strictly greater than zero for all x in our interval $0 \leq x \leq 1$, that $f(x)$ is very smooth, and in particular that $f'(x)$ — the derivative of the radius function with respect to axial coordinate — is always finite. It follows that the surface area of the surface can then be expressed as

$$A = \int_0^1 2\pi \big(1 + (f'(x))^2 \big)^{1/2} f(x)\, dx \ . \tag{1}$$

(Note that we consider only the lateral area: we do not include the surface area of the end plates at $x = 0$ and $x = 1$, respectively.)

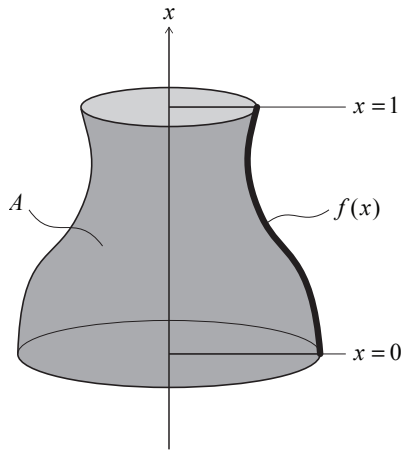

Figure 4: A surface of revolution described by the function $f(x)$ for $0 \leq x \leq 1$.

We now divide the interval $0 \leq x \leq 1$ into $N - 1$ segments, S_i, $1 \leq i \leq N - 1$, defined by the N points x_i, $1 \leq i \leq N$: $S_i \equiv [x_i, x_{i+1}]$. Note the points are ordered, $x_i < x_{i+1}$, $1 \leq i \leq N_1$, and also equi-spaced, $x_{i+1} - x_i \equiv h$, $1 \leq i \leq N - 1$. It follows that all the segments S_i, $1 \leq i \leq N - 1$, are of the same length, h.

We now write

$$A = \sum_{i=1}^{N-1} A^i \ ,$$

where

$$A^i \equiv \int_{x_i}^{x_{i+1}} 2\pi \big(1 + (f'(x))^2 \big)^{1/2} f(x)\, dx$$

is the contribution to the surface area from segment S_i for $1 \leq i \leq N-1$. Next, we introduce A_h given by

$$A_h = \sum_{i=1}^{N-1} A_h^i \quad , \tag{2}$$

where

$$A_h^i = 2\pi \left(1 + \left(\frac{f(x_{i+1}) + c_1 f(x_i)}{h} \right)^2 \right)^{1/2} \left(\frac{1}{2} f(x_i) + c_2 f(x_{i+1}) \right) h \ . \tag{3}$$

Here c_1 and c_2 are to be chosen to ensure that the approximation given by equations (2) and (3) converges to A of (1) as N tends to infinity and h tends to zero (for *any* f which satisfies our assumptions above).

(i) (2 points) The constant c_1 should be chosen as

(a) 0

(b) $-\frac{1}{2}$

(c) $\frac{1}{2}$

(d) -1

(e) 1

(ii) (2 points) The constant c_2 should be chosen as

(a) 0

(b) -$\frac{1}{2}$

(c) $\frac{1}{2}$

(d) -1

(e) 1

Now assume that the constants c_1 and c_2 have been chosen correctly.

(iii) (2 points) The statement " A_h given by equations (2) and (3) will *exactly* equal A given by equation (1) for $f(x) = 1 + 2x$ " is

(a) TRUE

(b) FALSE

You may assume here that all floating point calculations are performed exactly without any round-off or other finite-precision effects.

There are no inputs or outputs for this question. Your answer should be a one-line script as indicated in `A1Q12_Template.m`.

CHALLENGE 2: (optional: *no credit*) Our robot scans a room at angles (single-index row array) `angle_scan` and records associated voltages (single-index row array) `voltage_scan`. Write a script which estimates the area of the room. You will also need calibration data for our IR Rangefinder: at the set of distances (single-index row array) `distance_calib` the associated voltages are (single-index row array) `voltage_calib`. All of these arrays can be found in `room_data.mat` available on the Assignment 1 page.

2.086 Numerical Computation for Mechanical Engineers
Spring 2013