CS100M Spring 2007
Project 3
Due Thursday, March 8, at 6pm

Submit your files on-line in CMS before the project deadline. Both correctness and good programming style contribute to your project score.

You must work either on your own or with one partner. You may discuss background issues and general solution strategies with others, but the project you submit must be the work of just you (and your partner). If you work with a partner, you and your partner must register as a group in CMS and submit your work as a group.

Objectives

Completing this project will help you learn about vectors (1D arrays), matrices (2D arrays), and vectorized code.

Part 1: Numerical Integration

Sometimes it is necessary to calculate the value of integral using numerical methods.
Taking the integral of function f(x) from a to b means calculating the area between the following curves: y=0; y=f(x); x=a, x=b.

We can use different methods to approximate this area. In this task you have to implement different kinds of approximation.

1) right-rectangular fitting
2) left-rectangular fitting
3) trapezoidal rule

The basic idea of all these approximations is that we are trying to calculate the area under the function through the sum of the areas of "thin" rectangles or trapezoids.

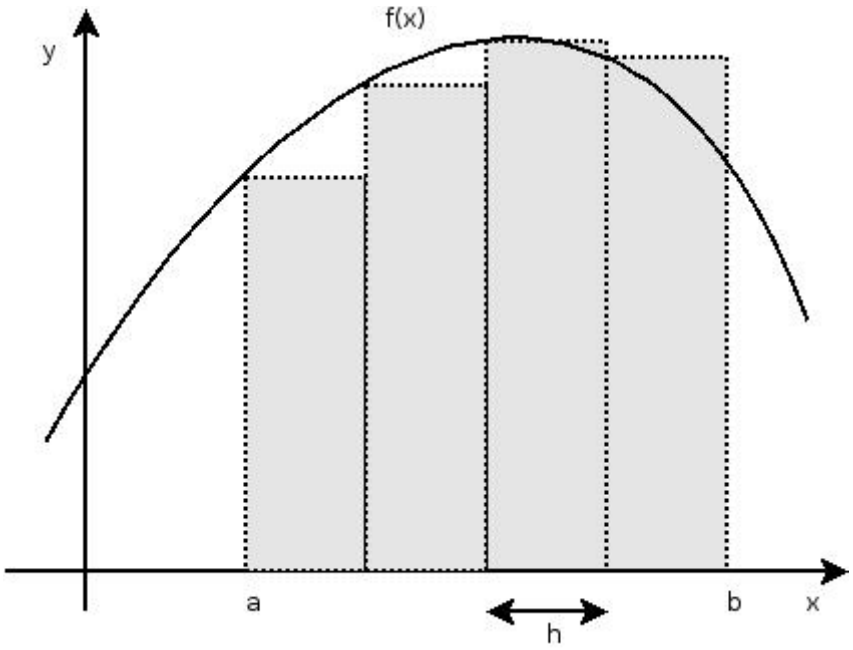
Suppose we have the following grid of values of x: a= $x_0, x_1, ... x_n$ =b, where two adjacent points are in a constant distance h = (b-a)/n from each other. As you know, if we increase the number n, the more accurate approximation we get.

1. **Left-rectangular fitting formula.** We can think of fitting the first rectangular with height = f(x_0), width = h;

If we take a sum then we will get the following formula:

$$IntegralAppr1 = \int_a^b f(x)dx = \sum_{k=0}^{n-1} \int_{x_k}^{x_{k+1}} f(x)dx \approx \sum_{k=0}^{n-1} h \times (f(x_k)) = h \times [f(x_0) + f(x_1) + ... + f(x_{n-1})],$$

where $h = \frac{(b-a)}{n}$

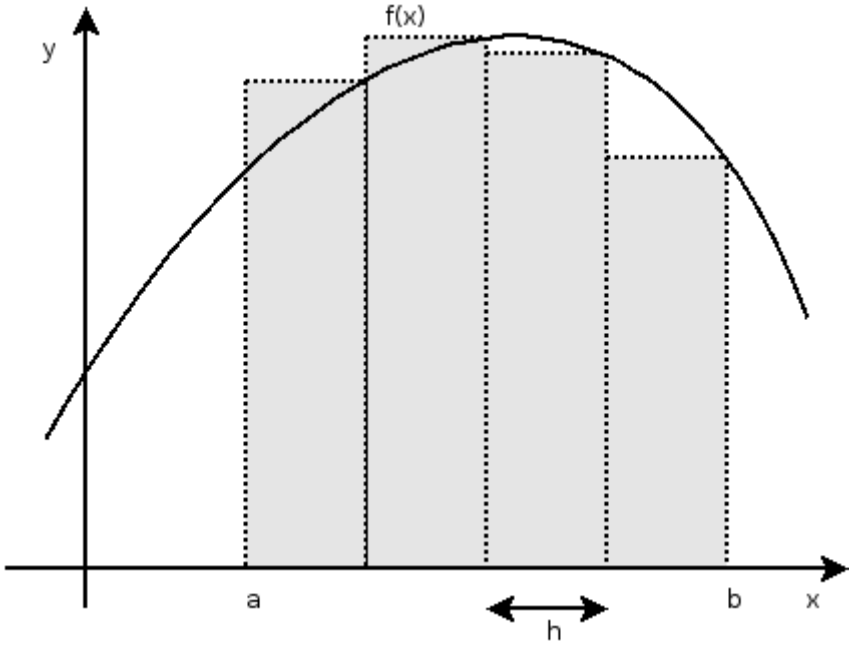


2. **Right-rectangular fitting formula.** We can think of fitting the first rectangular with height = f(x_1), width = h;

If we take a sum then we will get the following formula:

$$IntegralAppr2 = \int_a^b f(x)dx = \sum_{k=0}^{n-1} \int_{x_k}^{x_{k+1}} f(x)dx \approx \sum_{k=1}^{n} h \times (f(x_k)) = h \times (f(x_1) + f(x_2) + ... + f(x_n)),$$

where $h = \frac{(b-a)}{n}$

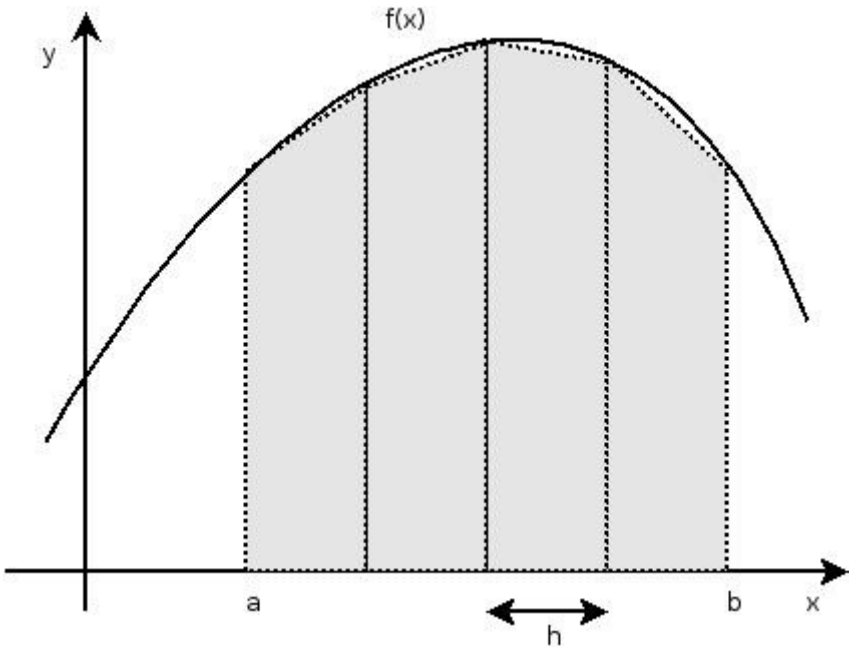


If the function is monotonically increasing then right-rectangular fitting formula will give the result greater than the Left-rectangular fitting formula.

3. **Trapezoid fitting formula.**

$$IntegralAppr3 = \int_a^b f(x)dx = \sum_{k=0}^{n-1} \int_{x_k}^{x_{k+1}} f(x)dx \approx \sum_{k=0}^{n-1} \frac{h}{2} \times [f(x_k) + f(x_{k+1})]$$

$$= \frac{h}{2} \times [f(x_0) + 2f(x_1) + \ldots + 2f(x_{n-1}) + f(x_n)].$$



Implementation issues:

Input:

You have to compute different approximations for function $f(x) = e^x$

a- the lower bound of integration

b- the upper bound of integration

n- defines the number of pieces in which want to divide *[a,b]*

Output:

1. The real value of integral $\int_a^b e^x dx = e^x \,|_{x=a}^{x=b} = e(b) - e(a)$
2. *IntegralAppr*1
3. *IntegralAppr*2
4. *IntegralAppr*3

Implementation issues:

We recommend using the Matlab function *cumsum,* although you may use loops instead if you wish. See Appendix for more information.

Here is a short description for the file provided: integral.m

This file contains the main function, which has 4 subfunctions. You have to write the subfunctions.

How to call the main function:

example: integral(1,2,100)

Some Useful Matlab Functions

`y = linspace(a,b,n)` generates a row vector `y` of `n` points linearly spaced between and including `a` and `b`.

Example: linspace(1,2,5)

ans =

1.0000 1.2500 1.5000 1.7500 2.0000

`A = arrayfun(fun, S)` applies the function specified by `fun` to each element of array `S`, and returns the results in array `A`. The value `A` returned by `arrayfun` is the same size as `S`, and the `(I,J,...)`th element of `A` is equal to `fun(S(I,J,...))`. The first input argument `fun` is a function handle to a function that takes one input argument and returns a scalar value. `fun` must return values of the same class each time it is called.

```
Example:  f = arrayfun (@(x)exp(x),x );
```

Cumsum : Cumulative sum

```
cumsum(1:5)
ans =
       [1  3  6  10  15]
We got the cumulative sum for [1 2 3 4 5] in following way
[1  1+2  1+2+3 1+2+3+4 1+2+3+4+5 ]
```

Part 2: Connect Four

Connect Four is a two-player game in which players take turns dropping discs into a vertical grid with the objective of getting four of one's own discs in a line. The game proceeds until a player wins or until there is no more space left on the grid.

Your goal for this problem is to implement Connect Four using Matlab. Your code should allow two human players to pit their skills against each other. In particular, you need to write code for ***connect4.m***, ***findbottom.m*** and ***checkvictory.m***.

Here are short descriptions for the files provided:

connect4.m

This is the main file for executing the game. You have to write code to determine which cell has been clicked and to place a disc into the appropriate cell. The "discs" are represented as X's in this version of the game. A two dimensional array, ***grid***, is used to keep track of the current state of the game. For integers x and y, grid(x,y) represents the contents of the grid-square with lower-left corner (x, y). A grid square can hold 0, 1, or 2 representing empty, player one's disc, or player two's disc, respectively.

You can use the Matlab function *ginput* to get the user's mouse click. Here's an example of its use. The values returned are floating point numbers.

```
[clickx, clicky] = ginput(1);   % Get location of one click
```

Be careful! Note that the indices to ***grid*** are based on x- and y- coordinates on the game-grid rather than row and column indices on the game-grid. Thus, for grid(x,y), the legal values for x are 1:maxCol and the legal values for y are 1:maxRow.

If someone clicks somewhere off the game board, your program should do something reasonable. All of these are reasonable: (1) ending the game, (2) using the nearest column in place of an "illegal" one, or (3) ignoring the bad click and informing/chastising the user (via the window title, for instance). An unreasonable response is for your program to halt with an error message about index out of bounds.

drawX.m

You do not have to change this file. For integers x and y, drawX(x,y,'r') draws a red X in the grid square with lower-left corner (x, y). Similarly, drawX(x, y, 'b') draws a blue X in the grid square with lower-left corner (x, y).

findbottom.m

This function finds the lowest available space in a particular column; the column is represented by an integer x. Note that, if a column is full, the lowest available space is one more than the number of rows.

checkvictory.m

This function checks if either player has won the game by having 4 of the same "discs" in a row – horizontally, vertically or diagonally. This function returns 0, 1, or 2 indicating that no player, player one, or player two, respectively, has won the game.