Lab Assignment #8
Topics: Numerical differentiation/integration
Due 4/10/2019 at 11:59pm on bCourses

The assignment will be graded by an autograder which will check the outputs of your functions. For your functions to be scored properly, it is important that the names of your functions **exactly match** the names specified in the problem statements, and input and output variables to each function are in the **correct order** (i.e. use the given function header exactly). Instructions for submitting your assignment are included at the end of this document.
If any problem specifies a certain method and/or algorithm to be used, you must implement that method and/or algorithm or you will receive a zero (0) on that problem.
On this assignment and future assignments, you are required to check that the input follows the format we provide in the table. For example, if we tell you that x is a positive integer, your code must evaluate if the x provided is a positive integer. If it is not valid, your program should terminate and you should display (disp) a message to the user why it has terminated.

1 Finite difference basics

1.1 Finite Difference Methods

`function [derivative] = myDerivative(y,dx,n,method)`		
Input	Type	Description
`y`	`1xN double`	An array of data: $y = [y_1, y_2, ..., y_N]$
`dx`	`1x1 double`	The spacing Δx
`n`	`1x1 double`	The order of the derivative, either first(1) or second(2)
`method`	`1xQ char`	The finite difference method you should use to find the derivative, either `'forward'`, `'backward'`, or `'central'`
Output		
`derivative`	`1x(N-2)double`	The value of the derivative given by the chosen finite difference method

Details

For this problem, you will code your first set of finite difference methods for finding the first or second derivative of an array of data: $y = [y_1, y_2, ..., y_N]$. This data is sampled from some unknown function $y = f(x)$ for an array of data: $x = [x_1, x_2, ..., x_N]$ at a constant x-interval Δx (i.e. $x_{i+1} - x_i = \Delta x$ for $i = \{2, 3, ..., N-1\}$). For this data, you will approximate the first or second derivative of $f(x)$, depending on whether `n` is `1` or `2`, for all x_i values except the endpoints (i.e. for $i = \{2, 3, ..., N-1\}$). Your code should make use of the following finite difference formulas, depending on the value given by

`method`:

First derivative forward difference method:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{\Delta x} \qquad i = 2, 3...N-1$$

First derivative backward difference method:

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{\Delta x} \qquad i = 2, 3...N-1$$

First derivative central difference method:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_{i-1})}{2\Delta x} \qquad i = 2, 3...N-1$$

Second derivative central difference method:

$$f''(x_i) \approx \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{\Delta x^2} \qquad i = 2, 3...N-1$$

Tips

- Note that the size of the output array is smaller than the input array, as you cannot calculate the finite difference approximations at both endpoints.

1.2 When are finite difference methods exact?

```
function [isExact,exact_val,fd_approx] = isFDMethodExact(p,x,method)
```

Input	Type	Description
`p`	`1xM double`	A set of polynomial coefficients (from highest to lowest order)
`x`	`1xN double`	A set of x-values you should use to evaluate the given method
`method`	`1xQ char`	The finite difference method that you should check
Output		
`isExact`	`1x1 logical`	Whether the given method is exact for the polynomial
`exact_val`	`1x(N-2)double`	The exact value of the derivative, evaluated from the polynomial derivative
`fd_approx`	`1x(N-2)double`	The approximated value of the derivative, evaluated from the chosen method

Details

Finite difference methods might have errors which depend on Δx. If it is used on polynomials of sufficiently low degree, finite difference methods are exact and have an error of 0 for any Δx. For this problem, you will write a function that takes one of the finite difference methods from the previous problem and checks if it is exact for the given polynomial. The `method` can be any from the following list:

- `'1stD-forward'`, for the first derivative forward difference method.
- `'1stD-backward'`, for the first derivative backward difference method.
- `'1stD-central'`, for the first derivative central difference method.
- `'2ndD-central'`, for the second derivative central difference method.

Your function should output the exact value of the derivative, the approximated value of the derivative, and whether the given method is exact.

Tips

- You can get the exact polynomial derivative from MALAB built-in function `polyder`.
- You may assume that the interval spacing Δx is uniform across all values in `x`.
- You may assume that the given list of polynomial coefficients `p` has no leading 0s.
- You may assume that `method` will not have a character array other than those listed.

2 Numerical integration basics

2.1 Numerical integration via Riemann sums

`function [integral] = myRiemann(f,interval,N,method)`		
Input	Type	Description
`f`	`function_handle`	A function handle which represents the function you are integrating
`interval`	`1x2 double`	The interval over which you are integrating your function
`N`	`1x1 double`	The number of sub-intervals to use in your integration
`method`	`1xQ char`	The type of Riemann sum you should use, either `'left'`, `'right'`, or `'midpoint'`
Output		
`integral`	`1x1 double`	The area under the curve (integral) given by the chosen Riemann sum

Details

For this problem, you will write a code that performs numerical integration of a function provided by the function handle `f` over the interval provided by `interval`, using `N` subintervals.

Assume we want to approximate the integral of $f(x)$ over the total interval $[a, b]$. To accomplish this goal, we discretize the interval into a numeral grid, x, consisting of $N+1$ points with spacing, $h = \frac{b-a}{N}$. We denote each point in x by x_i, where $x_0 = a$ and $x_N = b$. The interval $[x_i, x_{i+1}]$ is refferred to as a subinterval.

One simple method for approximating integrals is summing the area of rectangles for each subintergal. The width of the rectangle is $x_{i+1} - x_i = h$ and the height is defined by a function value $f(x)$ for some x in the subinterval. An obvious choice for the height is the function value at the left endpoint, x_i , or the right endpoint, x_{i+1}, or the midpoint $\frac{x_i+x_{i+1}}{2}$, while $y_i = \frac{x_i+x_{i+1}}{2}$. This method gives the Riemann Integral approximation, which is
Riemman left:

$$\int_a^b f(x)dx \approx \sum_{i=0}^{N-1} hf(x_i)$$

Riemann right:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{N} hf(x_{i+1})$$

Riemann midpoint:

$$\int_a^b f(x)dx \approx \sum_{i=0}^{N-1} hf(y_i)$$

Tips

- You may assume that the given function handle `f` can take arrays of x values as inputs and correctly produce outputs for them on a element-by-element basis.
- Note that N is the number of sub-intervals now, not the number of grid points.
- You may assume that `method` will only consist of `'left'`, `'right'`, or `'midpoint'`.

2.2 Numerical integration via Simpson's rule

`function [integral] = mySimpson(f,interval,N)`		
Input	Type	Description
`f`	`function_handle`	A function handle which represents the function you are integrating
`interval`	`1x2 double`	The x interval over which you are integrating your function
`N`	`1x1 double`	The number of sub-intervals to use in your integration
Output		
`integral`	`1x1 double`	The area under the curve (integral) given by Simpson's rule

Tips

For this problem, you will write a code that performs numerical integration of a function provided by the function handle `f` over the interval provided by `interval`, using `N` sub-intervals. You function should do this through the use of Simpson's rule.

Recall Simpson's rule from the lecture:

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(x_0) + 4\left(\sum_{i=1,i\ odd}^{N-1} f(x_i)\right) + 2\left(\sum_{i=2,i\ even}^{N-2} f(x_i)\right) + f(x_N)\right]$$

Details

- You may assume that the given function handle `f` can take arrays of x values as inputs and correctly produce outputs for them on a element-by-element basis.
- N is the number of intervals between grid points, thus your Simpson's rule should be based off of $\frac{N}{2}$ quadratics. You may also assume that N is even.

2.3 Comparing the error of numerical integration methods

`function [errors,orders,fig] = compareIntError(f,interval,Ns,int_methods)`		
Input	Type	Description
`f`	`function handle`	A function handle which represents the function you are evaluating your numerical integration methods on
`interval`	`1x2 double`	The x interval over which you are integrating your function
`Ns`	`1xQ double`	A set of sub-interval numbers that you should use
`int_methods`	`1xM cell`	A list of numerical integration methods to compare
Output		
`errors`	`MxQ double`	A matrix containing the error in the numerical integration methods
`orders`	`1xM double`	The estimated order of accuracy for each numerical integration method
`fig`	`1x1 matlab.ui.Figure`	A plot of the error values for each method in log-log space

Details

For this problem, you will write a function that compares a set of numerical integration methods provided to you via the `int_methods` input and plots the error of each method as Δx is varied across the given set of sub-intervals in `Ns`. The result should be a figure, plotted in log-log space, such as the one shown below:

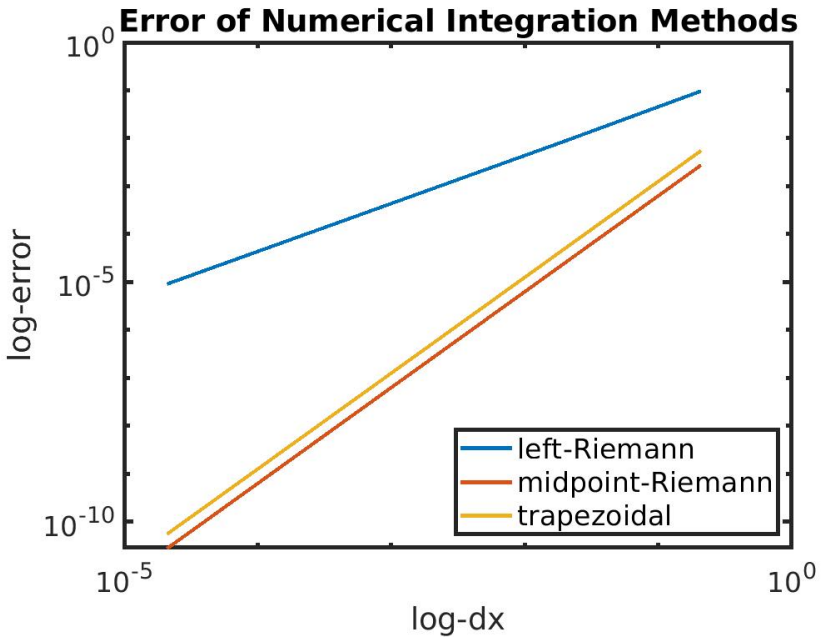

Figure 1: Example from tester

The `int_methods` input will be a cell array containing a set of the following `char` arrays:

- `'left-Riemann'`. In this case, you should use the left Riemann sum.
- `'right-Riemann'`. In this case, you should use the right Riemann sum.
- `'midpoint-Riemann'`. In this case, you should use the midpoint Riemann sum.
- `'trapezoidal'`. In this case, you should use the trapezoidal method. (You may make use of Matlab's `trapz` function to accomplish this).
- `'Simpson'`. In this case, you should use Simpson's method.

For each method, you should determine the error of each approximation by assuming that the value obtained by Matlab's built-in `integral` function is the true value of the integral. You should then compile these errors into the matrix `errors` and calculate the slope of the best fit line for each method in log-log space and store that value in `orders`.

Tips

- You may assume that the given function handle `f` can take arrays of x values as inputs and correctly produce outputs for them on a element-by-element basis.
- Your regression should allow for a non-zero y-intercept.
- Again, use `fig=figure;` **before** you do any plotting in order to create the figure handle. All plotting commands will then be applied to the most recent figure by default.

- Your figure should contain a **legend** that labels each line plotted with its corresponding finite difference method, as well as the **title** and **axis labels** shown in the sample figure. Otherwise, you may format the figure however you choose as long as the data plotted is correct.
- You may assume that `int_methods` will not have any character arrays in it other than those listed above and will not have any duplicates.
- You may assume that none of the methods will be exact for the given function.

3 Curve a class

`function [edges]=myCurve(mu,sigma,rate)`		
Input	Type	Description
`mu`	`1x1 double`	The location parameter
`sigma`	`1x1 double`	The scale parameter
`rate`	`1x4 double`	The ideal letter distribution from the professor
Output		
`edges`	`1x3 double`	The the grade score cutoffs for curving

Details

A probability density function (pdf) is a function that describes the probability P(x) that a random variable X will take on some value x. The integral of the pdf then gives the cumulative distribution function (cdf), whose function values represent the probability that a random variable X will take on any value less than or equal to some value x, i.e., $D(x) = P(X \leq x)$. The value given by $D(b) - D(a) = \int_a^b f(x)dx$ would then represent the probability that the random variable X takes on some value between a and b. The pdf is also constrained such that $\int_{-\infty}^{\infty} f(x)dx = 1$.

A professor uses a distribution $f(x)$ as the pdf for a group of students' midterm scores. The pdf is as follows:
Given $q(x) = \sigma^{-1} e^{\frac{x-\mu}{\sigma}} e^{-e^{\frac{x-\mu}{\sigma}}}$,

$$f(x) = \begin{cases} \frac{q(x)}{\int_0^{100} q(x)dx} & 0 \leq x \leq 100 \\ 0 & otherwise \end{cases}$$

For example, the figure below shows the extreme distribution pdf with $\mu = 75$ and $\sigma = 7$.

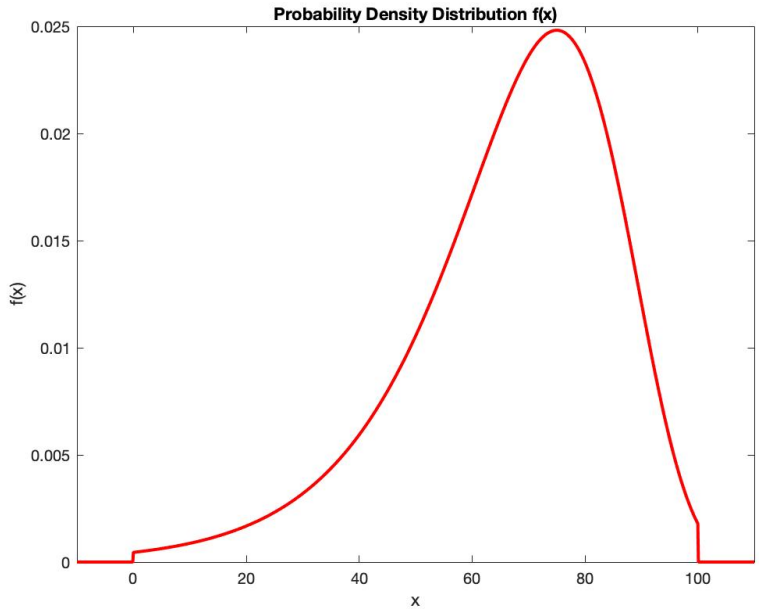

Figure 2: $f(x)$ with $\mu = 75$ and $\sigma = 15$

The students' scores are in the range of $[0, 100]$. Based on the existing pdf (given parameters μ and σ), the professor wants to curve the students' grade by making sure that some percentage of students receive an A, B, C, or D.

For this problem, you will write a function that takes the mean μ, standard deviation σ, and the desired letter grade distribution `rate` and uses the pdf $f(x)$ to compute the grade cutoffs required for the desired distribution.

For example, say the professor would like to curve the class such that 15% of students get an A, 30% of students get a B, 40% of students get a C, and 15% of students get a D. You would then write a function that takes `rate` = [0.15 0.3 0.4 0.15], `mu`, and `sigma` as inputs, and then outputs a 1-by-3 array `edges` that contains the grade cutoffs for the class.

In this case, if `edges` = [50,65,85], this means scores below 50 will receive a D, scores between 50 and 65 will receive a C, scores between 65 and 85 will receive a B, and scores above 85 will receive a A.

The accuracy of the `edge` scores should be within 0.1.

Submission Guidelines

When you are ready to test one or more of your function(s), download the `Lab08Tester.m` script from bCourses and run it in the same directory as your function(s). This will run a series of test cases for your function(s) and display the outputs. You should check these results with the example outputs displayed. Note that the series of tests that the script will run is not exhaustive and that it is up to you to make sure that your function fully

satisfies the requirements of the problem. If you want to run all the test cases for one function, select that part of the tester script and push the `Run Section` button. If you wish to only check a single test case, you may do so by highlighting the appropriate lines in Matlab and then pressing F9 (on Windows and linux) and `shift + fn + F7` (on a Mac) to run the selected text in the Command Window.

The script will also pack your function into a zip file for final submission. Your function file names and headers should match the examples given in the problem statement. If they do not match, the tester script will not be able to find them and will issue a warning. If you do not correct this problem, your missing function(s) will not be able to be graded and **you will receive 0 points**, so be sure to check carefully! Make sure to upload the `Lab08.zip` file to bcourses before the deadline. You may resubmit your zip file by re-uploading to bcourses before the deadline, but please be aware that only the most recent zip submission is considered for grading. That said, it is critical that you include your most recent version of **all** of your functions in each new zip submission.

Your final zip file for this assignment should contain the following:

- myDerivative.m
- isFDMethodExact.m
- myRiemann.m
- mySimpson.m
- compareIntError.m
- myCurve.m