E7: Introduction to Computer Programming for Scientists and Engineers
University of California at Berkeley, Spring 2017
Instructor: Lucas A. J. Bastien

Lab Assignment 11: Numerical Differentiation; Numerical Integration
Version: release

Due date: Friday April 14th 2017 at 12 pm (noon).

General instructions, guidelines, and comments:

- For each question, you will have to write and submit one or more Matlab functions. We provide a number of test cases that you can use to test your function. The fact that your function works for all test cases provided does not guarantee that it will work for all possible test cases relevant to the question. It is your responsibility to test your function thoroughly, to ensure that it will also work in situations not covered by the test cases provided. During the grading process, your function will be evaluated on a number of test cases, some of which are provided here, some of which are not.
- Submit on bCourses one m-file for each function that you have to write. The name of each file must be the name of the corresponding function, with the suffix `.m` appended to it. For example, if the name of the function is `my_function`, the name of the file that you have to submit is `my_function.m`. **Carefully check the name of each file that you submit.** Do not submit any zip file. If you re-submit a file that you have already submitted, bCourses may rename the file by adding a number to the file's name (*e.g.,* rename `my_function.m` into `my_function-01.m`). This behavior is okay and should be handled seamlessly by our grading system. Do not rename the file yourself as a response to this behavior.
- A number of optional Matlab toolboxes can be installed alongside Matlab to give it more functionality. All the functions that you have to write to complete this assignment can, however, be implemented without the use of any optional Matlab toolboxes. We encourage you to not use optional toolboxes to complete this assignment. All functions of the Matlab base installation will be available to our grading system, but functions from optional toolboxes may not. If one of your function uses a function that is not available to our grading system, you will loose all points allocated to the corresponding part of this assignment. To guarantee that you are not using a Matlab function from an optional toolbox that is not available to our grading system, use one or both of the following methods:
  - ◇ Only use functions from the base installation of Matlab.
  - ◇ Make sure that your function works on the computers of the 1109 Etcheverry Hall computer lab. All the functions available on these computers will be available to our grading system.

- For this assignment, the required submissions are:
  - ◇ `my_generic_derivative.m`
  - ◇ `my_generic_integral.m`
  - ◇ `my_solid_of_revolution.m`
  - ◇ `my_concentration.m`

Built-in functions that you may not use

You may not use Matlab's built-in functions `cumsum`, `cumtrapz`, `dblquad`, `diff`, `integral`, `integral2`, `integral3`, `polyint`, `quad`, `quad2d`, `quadgk`, `quadl`, `quadv`, `trapz`, and `triplequad`. This rule applies to each question of this lab assignment.

Where to find information about the methods used in this lab?

Numerical differentiation techniques were covered in lecture L28 (April 3rd 2017). Chapter 17 of the textbook also covers this topic. Numerical integration techniques were covered in lecture L29 (April 5th 2017). Chapter 18 of the textbook also covers this topic.

1. Generic numerical differentiation (30 points)

Consider a set of m data points of coordinates (x_i, y_i), $i = \{1, 2, \ldots, m\}$ such that:

- All the x_i's are different.
- The x_i's are ordered in increasing order.
- These points can be associated to a differentiable function f such that $f(x_i) = y_i$, $i = \{1, 2, \ldots, m\}$.

The following forward finite-difference formula estimates the value of the derivative f' of f at point x_i using the values of the function at points x_i and x_{i+1}:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \tag{1}$$

The following backward finite-difference formula estimates the value of the derivative f' of f at point x_i using the values of the function at points x_{i-1} and x_i:

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} \tag{2}$$

The following central finite-difference formula estimates the value of the derivative f' of f

at point x_i using the values of the function at points x_{i-1} and x_{i+1}:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_{i-1})}{x_{i+1} - x_{i-1}} \tag{3}$$

Note that the forward formula cannot be applied at $x = x_n$, the backward formula cannot be applied at $x = x_1$, and the central formula cannot be applied at either of these two points.

In this question, you will write a function that performs numerical differentiation on an arbitrary set of data points. More precisely, write a function with the following header:

```
function [forward, backward, central] = my_generic_derivative(x_data, y_data)
```

where:

- `x_data` and `y_data` are two $1 \times m$ arrays of class `double` that represent two-dimensional data. In other words, these row vectors represent a set of points of coordinates (`x_data(i)`, `y_data(i)`), $i = \{1, 2, \ldots, m\}$. You can assume that $m > 3$, that all the elements of `x_data` and `y_data` are different from `NaN`, `Inf`, and `-Inf`, and that the elements of `x_data` are different from each other and sorted in increasing order.
- `forward` is a $1 \times m$ array of class `double` whose elements are the values of the derivative of the function associated with the x- and y-data mentioned above, estimated at points `x_data(i)` with `i` $\in \{1, 2, \ldots, m\}$, in this order, using the first-order forward finite-difference formula.
- `backward` is a $1 \times m$ array of class `double` whose elements are the values of the derivative of the function associated with the x- and y-data mentioned above, estimated at points `x_data(i)` with `i` $\in \{1, 2, \ldots, m\}$, in this order, using the first-order backward finite-difference formula.
- `central` is a $1 \times m$ array of class `double` whose elements are the values of the derivative of the function associated with the x- and y-data mentioned above, estimated at points `x_data(i)` with `i` $\in \{1, 2, \ldots, m\}$, in this order, using the second-order central finite-difference formula.

In each of `forward`, `backward`, and `central`, if the corresponding finite-difference formula cannot be applied at one or more points of the data set, set the value of the derivative to `NaN` at these points.

Test cases:

```
>> x_data = 0:0.1:1;
>> [f, b, c] = my_generic_derivative(x_data, (x_data).^2)
f =
  Columns 1 through 7
    0.1000    0.3000    0.5000    0.7000    0.9000    1.1000    1.3000
  Columns 8 through 11
    1.5000    1.7000    1.9000       NaN
```

```
b =
  Columns 1 through 7
       NaN    0.1000    0.3000    0.5000    0.7000    0.9000    1.1000
  Columns 8 through 11
    1.3000    1.5000    1.7000    1.9000
c =
  Columns 1 through 7
       NaN    0.2000    0.4000    0.6000    0.8000    1.0000    1.2000
  Columns 8 through 11
    1.4000    1.6000    1.8000       NaN

>> x_data = 1:0.1:2;
>> [f, b, c] = my_generic_derivative(x_data, cos(x_data))
f =
  Columns 1 through 7
   -0.8671   -0.9124   -0.9486   -0.9753   -0.9923   -0.9994   -0.9964
  Columns 8 through 11
   -0.9836   -0.9609   -0.9286       NaN
b =
  Columns 1 through 7
       NaN   -0.8671   -0.9124   -0.9486   -0.9753   -0.9923   -0.9994
  Columns 8 through 11
   -0.9964   -0.9836   -0.9609   -0.9286
c =
  Columns 1 through 7
       NaN   -0.8897   -0.9305   -0.9620   -0.9838   -0.9958   -0.9979
  Columns 8 through 11
   -0.9900   -0.9722   -0.9447       NaN

>> x_data = [0, 0.2, 0.5, 1];
>> [f, b, c] = my_generic_derivative(x_data, exp(x_data))
f =
    1.1070    1.4244    2.1391       NaN
b =
       NaN    1.1070    1.4244    2.1391
c =
       NaN    1.2974    1.8711       NaN
```

2. Generic numerical integration (30 points)

In this question, you will write a function that performs numerical integration on an arbitrary function f to approximate the following integral:

$$\int_a^b f(x)dx \tag{4}$$

where a and b are two real numbers such that $a < b$. More precisely, write a function with the following header:

```
function [result] = my_generic_integral(f, a, b, n, method)
```

where:

- `f` is a function handle that represents a real-valued function f defined on $\mathbb{R}$. `f` takes a single input argument that is an array of class `double` of arbitrary size (the "x values"), and outputs a single output argument that is an array of class `double` of the same size (the corresponding "y values").
- `a` and `b` are two scalars of class `double` that represent a and b in Equation 4, respectively. You can assume that `a` and `b` are different from `NaN`, `Inf`, and `-Inf`, and that `a < b`.
- `n` is a scalar of class `double` that represents a non-infinite integer that is greater than zero. This number is the number of sub-intervals you should use when using numerical integration methods to approximate the value of the integral. All these sub-intervals should have the same width.
- `method` is a row vector of class `char` (*i.e.* a character string) that describes the numerical method to use to estimate the value of the integral. It can take one of the following values:
  - ◇ `'riemann'`. In this case, use the Riemann integral technique. On each sub-interval, use the value of the function at the left boundary of the sub-interval to approximate the value of the function to integrate.
  - ◇ `'midpoint'`. In this case, use the midpoint rule.
  - ◇ `'trapezoid'`. In this case, use the trapezoid rule.
  - ◇ `'simpson'`. In this case, use Simpson's rule.
- `result` is the value of

$$\int_a^b f(x)dx \tag{5}$$

  as approximated by the corresponding numerical method.

If `method` is `'simpson'` and `n` is not an even number, `result` should be set to `NaN` (hint: consult the documentation of Matlab's built-in function `mod`).

Test cases:

```
>> f = @(x) sin(x);
>> result = my_generic_integral(f, 0, pi, 20, 'riemann')
result =
    1.9959

>> f = @(x) sin(x);
>> result = my_generic_integral(f, 0, pi, 20, 'midpoint')
result =
    2.0021

>> f = @(x) exp(x);
>> result = my_generic_integral(f, 1, 2, 12, 'trapezoid')
result =
```

```
    4.6735

>> f = @(x) exp(x);
>> result = my_generic_integral(f, 1, 2, 12, 'simpson')
result =
    4.6708

>> f = @(x) exp(x);
>> result = my_generic_integral(f, 1, 2, 13, 'simpson')
result =
   NaN
```

3. Solid of revolution (20 points)

In this question, you will write a function that estimates the volume of a solid of revolution, using numerical integration. A solid of revolution is a solid obtained by rotating a profile line around an axis, as illustrated by Figure 1. In this question, the profile that is being rotated is defined by a line of equation $y = f(x)$ over an interval $[a, b]$ such that $f(x) \geqslant 0$ for all $x \in [a, b]$. This profile is rotated around the x-axis to form the solid of revolution.

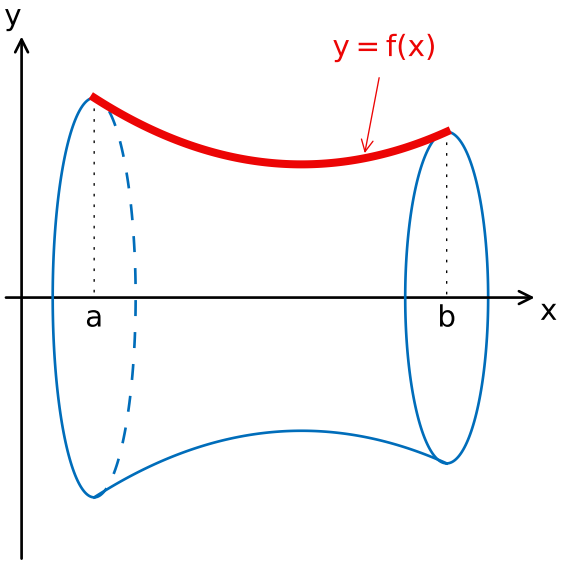

Figure 1: Illustration of a solid of revolution.

The volume V of the solid of revolution is given by:

$$V = \int_a^b \pi f(x)^2 dx \tag{6}$$

Write a function with the following header:

```
function [volume] = my_solid_of_revolution(f, a, b, n)
```

where:

- `f` is a function handle that represents the real-valued function f in Equation 6. This function is defined on $[a, b]$ and represents the profile of the solid of revolution. `f` takes a single input argument that is an array of class `double` of arbitrary size (the "x values"), and outputs a single output argument that is an array of class `double` of the same size (the corresponding "y values"). You can assume that all values returned by `f` are greater than or equal to zero.
- `a` and `b` are two scalars of class `double` that represent a and b in Equation 6, respectively. You can assume that `a` and `b` are different from `NaN`, `Inf`, and `-Inf`, and that `a < b`.
- `n` is a scalar of class `double` that represents a non-infinite integer that is greater than zero. This number is the number of sub-intervals you should use when using a numerical integration method to approximate the value of V. All these sub-intervals should have the same width.
- `volume` is a scalar of class `double` that represents the volume V of the solid as approximated by the trapezoid rule applied to Equation 6.

Test cases:

```
>> % Let us first start by calculating the volume of common solids of
>> % revolution
>> % ---> Cylinder of radius 1 and height 10
>> volume = my_solid_of_revolution(@(x) ones(size(x)), 0, 10, 20)
volume =
   31.4159

>> % ---> Sphere of radius 2
>> volume = my_solid_of_revolution(@(x) sqrt(4-x.^2), -2, 2, 50)
volume =
   33.4969

>> % Let us now calculate the volume of other solids of revolution
>> volume = my_solid_of_revolution(@(x) sin(x)+3, 5, 10, 100)
volume =
  169.2409

>> volume = my_solid_of_revolution(@(x) x.^2, 1, 2, 30)
volume =
   19.4860
```

4. Emissions from a roadway segment (20 points)

Consider a straight segment of roadway of length s in a flat region. Vehicles traveling on this roadway emit pollutants into the atmosphere. In the presence of wind, this pollution is transported to downwind areas. This polluted air is also diluted and mixed with fresh air by atmospheric motions. We make the simplifying assumption that the average wind

vector is constant, horizontal, and perpendicular to the roadway. We call u the magnitude of this wind. The concentration of pollutants downwind of the roadway depends on the distance from the road, and on atmospheric conditions that define how efficiently pollution is transported and mixed in the atmosphere. In this question, we assume that there is no source of pollutant besides the vehicles on the road segment.

We define the y-axis to be parallel to the road, and the x-axis to be horizontal and perpendicular to the road, such that the downwind direction corresponds to positive values of x. We define the axes such that the middle of the segment of road is located at the point of coordinates $(0, 0)$. Assuming that the pollutant is not reactive and does not deposit onto the ground, the surface concentration $C(x, y)$ of a pollutant of interest at a location (x, y) downwind of the road can be estimated as (adapted from Seinfeld and Pandis, 2006):

$$\int_{\lambda=-s/2}^{\lambda=s/2} \frac{q}{\pi u \sigma_y(x) \sigma_z(x)} \exp\left[-\frac{(y-\lambda)^2}{2\sigma_y^2(x)}\right] d\lambda \tag{7}$$

where:

- q is the emission rate of the pollutant of interest per unit length of the road. In this question, we assume that this quantity is constant.
- σ_y and σ_z describe the intensity of pollutant dispersion in the y- and z-directions, respectively. The z-direction is the vertical direction. These quantities depend on atmospheric conditions. In this question, we assume the atmosphere to be moderately unstable. In this case, σ_y and σ_z can be estimated as (Seinfeld and Pandis, 2006):

$$\sigma_y(x) = \exp(-1.634 + 1.0350 \ln(x) - 0.0096 \ln^2(x)) \tag{8}$$
$$\sigma_z(x) = \exp(-1.999 + 0.8752 \ln(x) + 0.0136 \ln^2(x)) \tag{9}$$

  where x, σ_y, and σ_z are in units of meters.
- λ is the integration variable. The integral in Equation 7 represents the summation (over each location λ along the road segment) of the contributions to air pollution of infinitesimally small sections of the road segment.

Figure 2 shows an example of the application of Equation 7. Surface pollutant concentration is highest near the roadway, and decreases with distance from the roadway, in both x and y-directions.

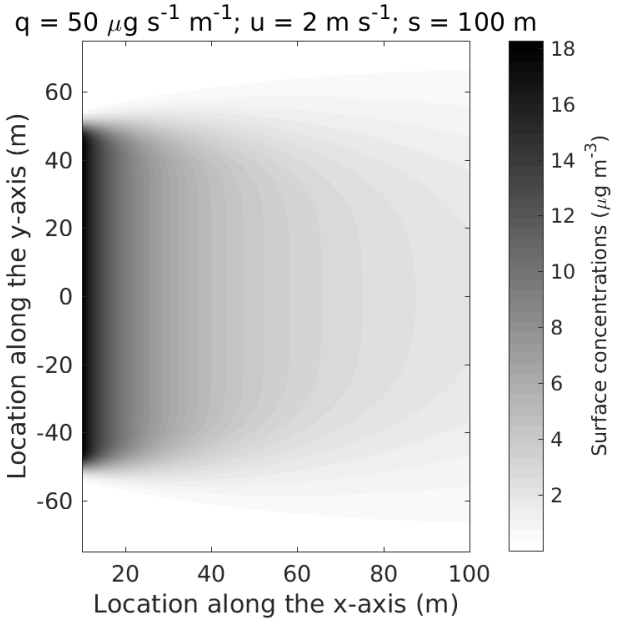

Figure 2: Surface concentration field downwind of a roadway segment, as estimated by Equation 7.

Write a function with the following header:

```
function [c] = my_concentration(q, u, s, x, y)
```

where:

- `q`, `u`, `s`, `x`, and `y` are scalars of class `double` that represent q, u, s, x, and y in Equation 7, respectively. You can assume that all these quantities are different from `NaN`, `Inf`, and `-Inf`, and that `q`, `u`, `s`, and `x` are positive. `s`, `x`, and `y` will be given in units of meters, `u` in units of meters per second, and `q` in units of micrograms of pollutant emitted per second and per meter of road.
- `c` is the concentration of pollutant (in units of micrograms per cubic meter) at location (x, y), as approximated by applying the midpoint formula to Equation 7, using 100 equally spaced sub-intervals.

You do not have to do any unit conversion in this question.

Test cases:

```
>> c = my_concentration(100, 2, 100, 50, 15)
c =
    7.7924

>> c = my_concentration(100, 2, 100, 50, 50)
c =
```

```
    3.8968

>> c = my_concentration(100, 2, 50, 40, 25)
c =
    4.8477

>> c = my_concentration(50, 4, 50, 40, 25)
c =
    1.2119
```

5. References

Seinfeld, J. H. and S. N. Pandis. 2006. *Atmospheric Chemistry and Physics. From Air Pollution to Climate Change*, John Wiley & Sons, Inc. Hoboken, New Jersey, 2nd edn.