

# Computación Aplicada - Homework 04

## Simulation – Basics & Integrals

Bruno González Soria (A01169284)  
Antonio Osamu Katagiri Tanaka (A01212611)

February 25, 2019

## Contents

<b>1</b>	<b>Problem I</b>	<b>4</b>
1.1	Function 1 . . . . .	4
1.2	Function 2 . . . . .	5
1.3	Let's plot the approximations . . . . .	5
<b>2</b>	<b>Problem II</b>	<b>8</b>
<b>3</b>	<b>Problem III</b>	<b>11</b>
<b>4</b>	<b>Problem IV</b>	<b>14</b>
<b>A</b>	<b><i>integralPlot</i> function</b>	<b>17</b>
<b>B</b>	<b>Full R Script</b>	<b>18</b>
<b>C</b>	<b>Full Output Log</b>	<b>25</b>

## List of Figures

1	Problem 1 instructions. . . . .	4
2	Taylor series until the 6th term of Function 1. . . . .	4
3	Taylor series until the 6th term of Function 2. . . . .	5
4	Listing 3 output; $f(x) = \text{Sin}(x)$ approximations, centered in 0. . . . .	7
5	Listing 3 output; $f(x) = e^{ix}$ approximations, centered in 1. . . . .	7
6	Problem 2 instructions. . . . .	8
7	Listing 5 output; $\int_{-2}^2 e^{x+x^2} dx$ . . . . .	10
8	Problem 3 instructions. . . . .	11
9	Problem 4 instructions. . . . .	14

## Listings

1	"The <i>taylorPlot</i> function" . . . . .	5
2	"Define <i>f0</i> and <i>f1</i> " . . . . .	6
3	"Implement <i>taylorPlot</i> " . . . . .	6
4	"The <i>Riemann</i> function" . . . . .	8
5	"Define the function and implement <i>riemann_sum</i> " . . . . .	9
6	"Listing 5 output" . . . . .	10
7	"The <i>Riemann 2D</i> function" . . . . .	12
8	"Define the function and implement <i>riemann_sum_2d</i> " . . . . .	12
9	"Listing 8 output" . . . . .	13
10	"The <i>derivative</i> function" . . . . .	15
11	"Define the function and implement <i>derivative</i> " . . . . .	15
12	"Listing 11 output" . . . . .	16

# 1 Problem I

HW

❖ Taylor series: (25 points)

- • Handwritten until the 6th-term to get the formula  
• Function 1

$$f(x) = \sin(x)$$

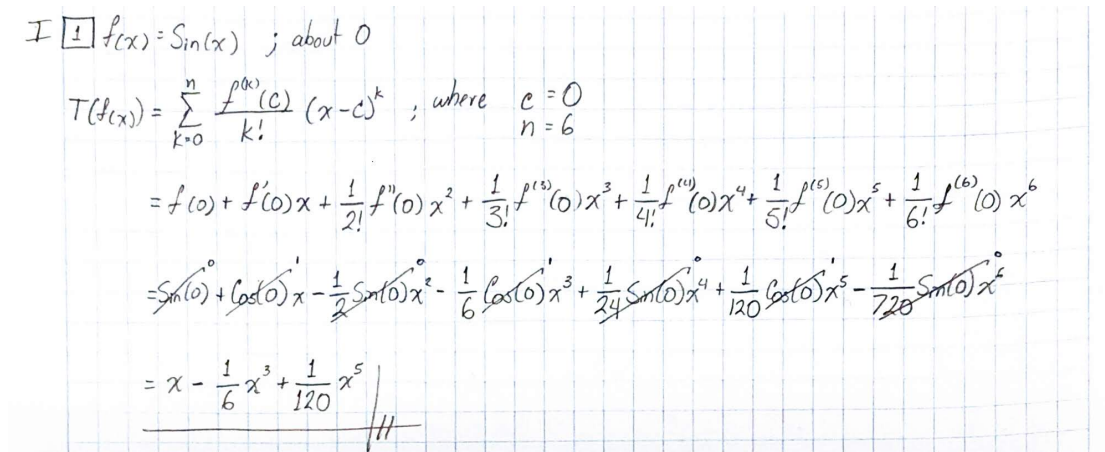
- Function 2

If  $i^2 = -1$  compute  $e^{ix}$  about 0

Figure 1: Problem 1 instructions.

## 1.1 Function 1

Figure 2 shows the “handwritten” procedure to find the first six terms of the Taylor series of  $f(x) = \sin(x)$ , centered at 0.



Handwritten work on graph paper showing the Taylor series expansion of  $f(x) = \sin(x)$  centered at 0.

1.  $f(x) = \sin(x)$  ; about 0

2.  $T(f(x)) = \sum_{k=0}^n \frac{f^{(k)}(c)}{k!} (x-c)^k$  ; where  $c=0$  and  $n=6$

3. 
$$= f(0) + f'(0)x + \frac{1}{2!}f''(0)x^2 + \frac{1}{3!}f^{(3)}(0)x^3 + \frac{1}{4!}f^{(4)}(0)x^4 + \frac{1}{5!}f^{(5)}(0)x^5 + \frac{1}{6!}f^{(6)}(0)x^6$$

4. 
$$= \sin(0) + \cos(0)x - \frac{1}{2}\sin(0)x^2 - \frac{1}{6}\cos(0)x^3 + \frac{1}{24}\sin(0)x^4 + \frac{1}{120}\cos(0)x^5 - \frac{1}{720}\sin(0)x^6$$

5. 
$$= x - \frac{1}{6}x^3 + \frac{1}{120}x^5$$

Figure 2: Taylor series until the 6th term of Function 1.

## 1.2 Function 2

Figure 3 shows the “hadwritten” procedure to find the first six terms of the Taylor series of  $f(x) = e^{ix}$ , centered at 1.

Handwritten work on graph paper showing the Taylor series expansion of  $f(x) = e^{ix}$  centered at 1. The text reads: "I [2]  $f(x) = e^{ix}$  ; about 1 (where  $i^2 = -1$ )". Below this, the expansion is written as:

$$T(f(x)) = f(1) + f'(1)(x-1) + \frac{1}{2!}f''(1)(x-1)^2 + \frac{1}{3!}f^{(3)}(1)(x-1)^3 + \frac{1}{4!}f^{(4)}(1)(x-1)^4 + \frac{1}{5!}f^{(5)}(1)(x-1)^5 + \frac{1}{6!}f^{(6)}(1)(x-1)^6$$

$$= e^i + ie^i(x-1) - \frac{1}{2}e^i(x-1)^2 - \frac{1}{6}ie^i(x-1)^3 + \frac{1}{24}e^i(x-1)^4 + \frac{1}{120}ie^i(x-1)^5 - \frac{1}{720}e^i(x-1)^6$$

The work is signed with a double hash symbol  $##$  at the bottom right.

Figure 3: Taylor series until the 6th term of Function 2.

## 1.3 Let's plot the approximations

In Listing 1, function *taylorPlot* is implemented to plot the Taylor approximations up to the 2nd, 4th, 6th and 8th terms. With the following properties:

### Parameters:

- **f** : function  
Vectorized function of one variable
- **c** : numeric  
point where the series expansion will take place
- **from, to** : numeric  
Interval of points to be plotted

### Returns:

- **void**

Listing 1: "The *taylorPlot* function"

```

1 library(pracma)
2
3 taylorPlot <- function(f, c, from, to) {
4   x <- seq(from, to, length.out = 100)
5   yf <- f(x)
6
7   yp2 <- polyval(taylor(f, c, 2), x)
8   yp4 <- polyval(taylor(f, c, 4), x)
9   yp6 <- polyval(taylor(f, c, 6), x)
10  yp8 <- polyval(taylor(f, c, 8), x)
11
12  plot(
13    x,
14    yf,
15    xlab = "x",
16    ylab = "f(x)",

```

```

17     type = "l",
18     main = 'Taylor Series Approximation of f(x)',
19     col = "black",
20     lwd = 2
21 )
22
23 lines(x, yp2, col = "#c8e6c9")
24 lines(x, yp4, col = "#81c784")
25 lines(x, yp6, col = "#4caf50")
26 lines(x, yp8, col = "#388e3c")
27
28 legend(
29     'topleft',
30     inset = .05,
31     legend = c("TS_8_terms", "TS_6_terms", "TS_4_terms", "TS_2_terms", "f
      (x)"),
32     col = c('#388e3c', '#4caf50', '#81c784', '#c8e6c9', 'black'),
33     lwd = c(1),
34     bty = 'n',
35     cex = .75
36 )
37 }

```

$f(x) = \sin(x)$  is defined as *f0* and  $f(x) = e^{ix}$  is defined as *f1* in Listing 2

Listing 2: "Define *f0* and *f1*"

```

1 f0 <- function(x) {
2   res = sin(x)
3
4   return(res)
5
6 }
7
8 f1 <- function(x) {
9   res = exp(complex(real = 0, imaginary = 1)*x)
10
11   return(res)
12
13 }

```

Listing 3 shows the use of function *taylorPlot* to plot the Taylor approximations of *f0* and *f1*. This Listing output-plots are represented within Figures 4 and 5

Listing 3: "Implement *taylorPlot*"

```

1 taylorPlot(f0, 0, -6.6, 6.6)
2 taylorPlot(f1, 1, -2*pi, 2*pi)

```

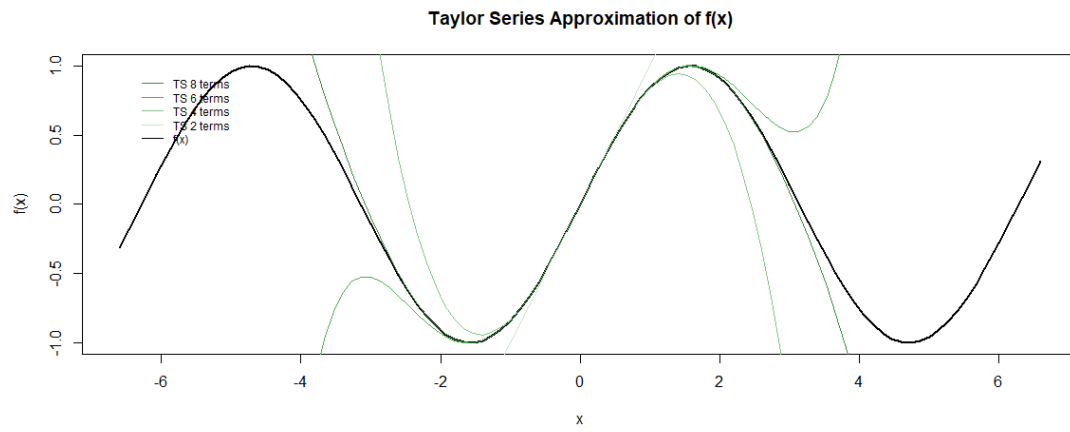


Figure 4: Listing 3 output;  $f(x) = \sin(x)$  approximations, centered in 0.

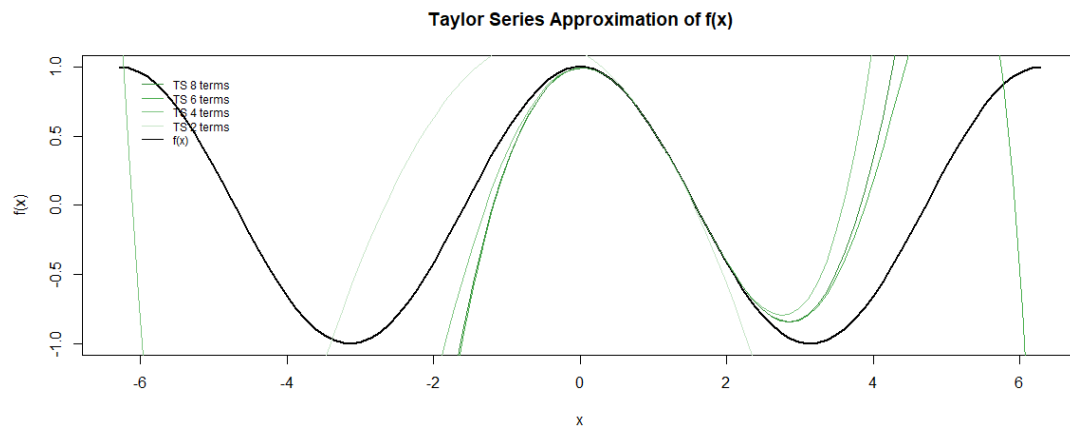


Figure 5: Listing 3 output;  $f(x) = e^{ix}$  approximations, centered in 1.

## 2 Problem II

HW

❖ Code: (25 points)

- Riemann sums function

❖ Function (25 points)

$$\int_{-2}^2 e^{x+x^2} dx$$

◦ Handwritten : Apply Taylor series

◦ R Code: Riemann sum, and Monte Carlo approach to:

error  $\Delta(RS - TS)$

6th term

Figure 6: Problem 2 instructions.

In Listing 4, function *riemann\_sum* is implemented to compute the Riemann sum of a given function  $f(x)$  over an interval  $[a, b]$ . With the following properties:

### Parameters:

- **f** : function  
Vectorized function of one variable
- **a, b** : numeric  
Endpoints of the interval  $[a, b]$
- **n** : numeric  
Number of subintervals of equal length in the partition of  $[a, b]$

### Returns:

- numeric  
Underestimate and overestimate approximations of the integral given by the Riemann sum.

Listing 4: "The *Riemann* function"

```

1 | riemann_sum <- function(f, a, b, n) {
2 |   # initialize values
3 |   lower.sum <- 0
4 |
5 |   upper.sum <- 0
6 |
7 |   h <- (b - a) / n

```



```

8
9
10 # riemann right sum
11 for (i in n:1) {
12   x <- a + i * h
13
14   lower.sum <- lower.sum + f(x)
15
16 }
17 lower.sum <- h * lower.sum
18
19
20 # riemann left sum
21 for (i in 1:n) {
22   x <- b - i * h
23
24   upper.sum <- upper.sum + f(x)
25
26 }
27 upper.sum <- h * upper.sum
28
29
30 # let's plot the curve
31 integralPlot(
32   f = f,
33   a = a,
34   b = b,
35   title = expression(f(x))
36 )
37
38 # print/get riemann sum
39 cat(sprintf(
40   "The true value is between %f and %f.\n",
41   as.double(lower.sum),
42   as.double(upper.sum)
43 ))
44
45 return(c(lower.sum, upper.sum))
46
47 }

```

Let's solve the following integral using *riemann\_sum*. Listing 5 shows the required commands.

$$\int_{-2}^2 e^{x+x^2} dx$$

Listing 5: "Define the function and implement *riemann\_sum*"

```

1 f4 <- function(x) {
2   res = exp(x + x ^ 2)
3
4   return(res)
5
6 }
7
8 # compute riemann_sum for f4
9 riemann_sum(f4, -2, 2, 100000)
10
11 # let's verify our calualtion using R's function

```

```
12 | integrate(f4, lower = -2, upper = 2)
```

Listing 6: "Listing 5 output"

```
1 > # compute riemann_sum for f4
2 > riemann_sum(f4, -2, 2, 100000)
3 The true value is between 93.170674 and 93.154833.
4 [1] 93.17067 93.15483
5
6 > # let's verify our calculation using R's function
7 > integrate(f4, lower = -2, upper = 2)
8 93.16275 with absolute error < 0.00062
```

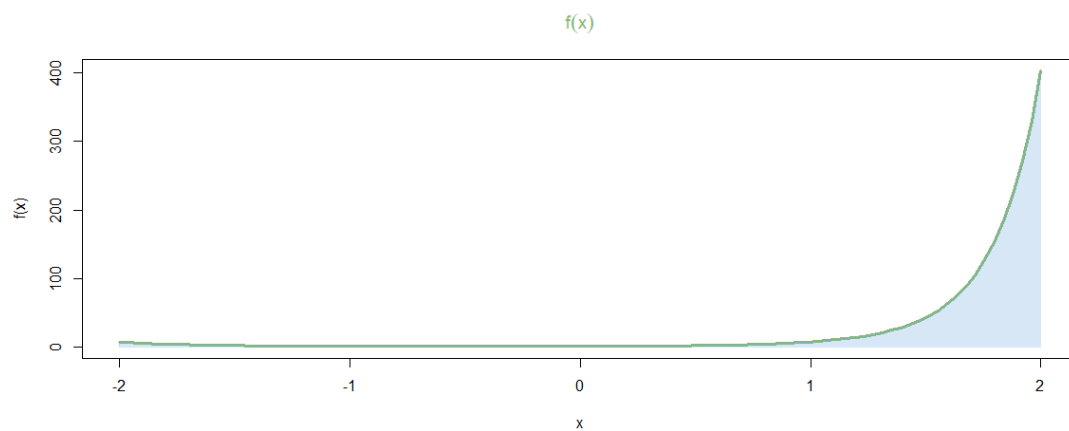


Figure 7: Listing 5 output;  $\int_{-2}^2 e^{x+x^2} dx$ .

\* Appendix A implements the R function to generate plots similar to Figure 7 (, which is used within *riemann\_sum*).

### 3 Problem III

HW

❖ Monte Carlo approach to: (25 points)

RS 2D

$$\int_0^1 \int_0^1 e^{(x+y)^2} dy dx$$

RS R-code

Figure 8: Problem 3 instructions.

In Listing 7, function *riemann\_sum\_2d* is implemented to compute the Riemann sum of a given function  $f(x, y)$  over the intervals  $[a, b]$  and  $[c, d]$ . With the following properties:

**Parameters:**

- **f** : function  
Vectorized function of one variable
- **a, b** : numeric  
Endpoints of the interval  $[a, b]$  (inner integral)
- **c, d** : numeric  
Endpoints of the interval  $[c, d]$  (outer integral)
- **nx** : numeric  
Number of subintervals of equal length in the partition of  $[a, b]$
- **ny** : numeric  
Number of subintervals of equal length in the partition of  $[c, d]$

**Returns:**

- numeric  
Approximations of the integral given by the Riemann 2D sum.

Listing 7: "The *Riemann 2D* function"

```

1 riemann_sum_2d <- function(f, a, b, c, d, nx, ny) {
2   # initialize values
3   dx = (b - a) / nx
4   s = 0.0
5   x = a
6
7   dy = (d - c) / ny
8   y = c
9
10  # riemann 2D sum
11  for (i in 1:nx) {
12    for (j in 1:ny) {
13      x = a + dx / 2 + i * dx
14      y = c + dy / 2 + j * dy
15      f_i = f(x, y)
16      s = s + f_i * dx * dy
17    }
18  }
19
20  # print/get riemann sum
21  cat(sprintf("The true value is around %f.\n",
22             as.double(s)))
23
24  return(s)
25
26 }
```

Let's solve the following integral using *riemann\_sum\_2d*. Listing 8 shows the required commands.

$$\int_0^1 \int_0^1 e^{(x+y)^2} dy dx$$

Listing 8: "Define the function and implement *riemann\_sum\_2d*"

```

1 f5 <- function(x, y) {
2   res = exp((x + y) ^ 2)
3
4   return(res)
5
6 }
7
8 # compute riemann_sum_2d for f5
9 riemann_sum_2d(f5, 0, 1, 0, 1, 1000, 1000)
10
11 # let's verify our calculation using R's function
12 integral2(f5, 0,1, 0,1)
```

Listing 9: "Listing 8 output"

```
1 > # compute riemann_sum_2d for f5
2 > riemann_sum_2d(f5, 0, 1, 0, 1, 1000, 1000)
3 The true value is around 4.926310.
4 [1] 4.92631
5
6 > # let's verify our calualtion using R's function
7 > integral2(f5, 0,1, 0,1)
8 $Q
9 [1] 4.899159
10 $error
11 [1] 9.974762e-16
```

## 4 Problem IV

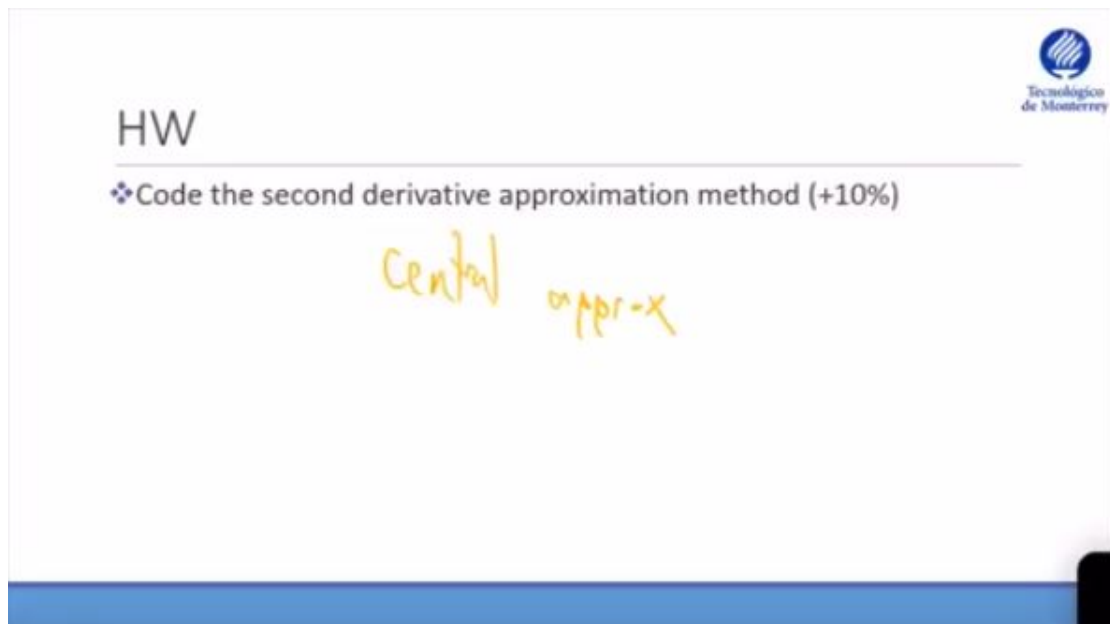


Figure 9: Problem 4 instructions.

In Listing 10, function *derivative* is implemented to compute the derivative of a function.. With the following properties:

**Parameters:**

- **f** : function  $f(x)$   
Vectorized function of one variable
- **h** : numeric  
Let  $x$  now change by an amount  $h$ .  $h$  is the variable that approaches 0

**Returns:**

- function  
Approximation of the derivative of  $f$ , given a step  $h$ .

Listing 10: "The *derivative* function"

```

1 derivative <- function(f, h) {
2   return(function(x) {
3     (f(x + h) - f(x)) / (h)
4   })
5 }

```

Let's solve the following double derivative to test our function using *derivative*. Listing 11 shows the required commands.

$$\frac{d^2}{dy^2} x^4 \sin(x)$$

Listing 11: "Define the function and implement *derivative*"

```

1 f6 <- function(x) {
2   res = x^4*sin(x)
3
4   return(res)
5 }
6
7
8 # df1 = d/dx(x^4 sin(x))
9 #      = x^3 (4 sin(x) + x cos(x))
10 df1 <- derivative(f6, 0.0001)
11
12 # df2 = d/dx(x^3 (4 sin(x) + x cos(x)))
13 #      = x^2 (8 x cos(x) - (x^2 - 12) sin(x))
14 df2 <- derivative(df1, 0.0001)
15
16 # Let's evaluate x=pi in the second derivative df2
17 df2(pi)
18
19 # Let's use the eval() function to verify our solution. The value should
20   be around df2(pi)
21 eval({ x <- pi; x^2*(8*x*cos(x) - (x^2 - 12)*sin(x))})

```

Listing 12: "Listing 11 output"

```
1 > # Let's evaluate  $x=\pi$  in the second derivative df2
2 > df2(pi)
3 [1] -248.076
4 >
5 > # Let's use the eval() function to verify our solution. The value should
   be around df2(pi)
6 > eval({ x <- pi; x^2*(8*x*cos(x) - (x^2 - 12)*sin(x))})
7 [1] -248.0502
```



## A *integralPlot* function

```
1  # Plotting the Areas under Curves #####
2  integralPlot <- function(f,
3                          a,
4                          b,
5                          from = a,
6                          to = b,
7                          title = NULL) {
8      # Plot the area under a function over the interval [a,b] between [from,
9      to].
10     # Parameters
11     # -----
12     # f : function
13     # funtion to be plotted
14     # a , b : numeric
15     # Endpoints of the integral interval [a, b]
16     # from , to : numeric (optional)
17     # Endpoints of the plot in the x-axis [x min, x max]
18     # title : expression
19     # title of the plot
20     #
21     # Returns
22     # -----
23     # void
24
25     x <- seq(from, to, length.out = 100) # input continuum
26     y <- f(x) # output
27
28     # plot the curve
29     plot(
30         x,
31         y,
32         xlim = c(from, to),
33         ylim = c(ifelse(min(y) < 0, min(y), 0), max(y)),
34         xlab = "x",
35         ylab = "f(x)",
36         main = title,
37         col.main = "#86B875",
38         type = "l",
39         lwd = 3,
40         col = "#86B875"
41     )
42
43     # area under the curve
44     x <- seq(a, b, length.out = 100)
45     y <- f(x)
46     polygon(
47         c(x, b, a, a),
48         c(y, 0, 0, f(a)),
49         border = adjustcolor("#7DB0DD", alpha.f = 0.3),
50         col = adjustcolor("#7DB0DD", alpha.f = 0.3)
51     )
52 }
```

## B Full R Script

```
1 *****
2 ** AUTHOR(S) :
3 **      Bruno Gonzalez Soria      (A01169284)
4 **      Antonio Osamu Katagiri Tanaka (A01212611)
5 **
6 ** FILENAME :
7 **      Homework4.R
8 **
9 ** DESCRIPTION :
10 **      Simulations (Ene 19 Gpo 1)
11 **      Homework 4
12 **
13 ** NOTES :
14 **      - https://www.math.ubc.ca/~pwalls/math-python/integration/riemann-sums/
15 **      - https://activecalculus.org/multi/S-11-1-Double-Integrals-Rectangles.html
16 **      - http://math.colgate.edu/faculty/valente/math113/supplements/section151handout.pdf
17 **      - http://hplgit.github.io/Programming-for-Computations/pub/p4c/p4c-sphinx-Python/\_pylight004.html
18 **      - https://rstudio-pubs-static.s3.amazonaws.com/131664\_1858eec97df54c9b8d5edcd8b22e5818.html
19 **
20 ** START DATE :
21 **      21 Feb 2019
22 *****
23
24 # Install required libraries
25 #install.packages('pracma', dependencies=TRUE);
26
27 # Plotting the Areas under Curves #####
28 integralPlot <- function(f,
29                          a,
30                          b,
31                          from = a,
32                          to = b,
33                          title = NULL) {
34   # Plot the area under a function over the interval [a,b] between [from,
35   # to].
36   #
37   # Parameters
38   # -----
39   # f : function
40   # funtion to be plotted
41   # a , b : numeric
42   # Endpoints of the integral interval [a, b]
43   # from , to : numeric (optional)
44   # Endpoints of the plot in the x-axis [x min, x max]
45   # title : expression
46   # title of the plot
47   #
48   # Returns
49   # -----
50   # void
51
52   x <- seq(from, to, length.out = 100) # input continuum
```

```

52 y <- f(x) # output
53
54 # plot the curve
55 plot(
56   x,
57   y,
58   xlim = c(from, to),
59   ylim = c(ifelse(min(y) < 0, min(y), 0), max(y)),
60   xlab = "x",
61   ylab = "f(x)",
62   main = title,
63   col.main = "#86B875",
64   type = "l",
65   lwd = 3,
66   col = "#86B875"
67 )
68
69 # area under the curve
70 x <- seq(a, b, length.out = 100)
71 y <- f(x)
72 polygon(
73   c(x, b, a, a),
74   c(y, 0, 0, f(a)),
75   border = adjustcolor("#7DB0DD", alpha.f = 0.3),
76   col = adjustcolor("#7DB0DD", alpha.f = 0.3)
77 )
78 }
79
80 #####
81 # PART 1 #####
82 # TAYLOR SERIES
83
84 library(pracma)
85
86 taylorPlot <- function(f, c, from, to) {
87   # Plot the Taylor approximations up to the 2nd, 4th, 6th and 8th terms
88   #
89   # Parameters
90   # -----
91   # f : function
92   # Vectorized function of one variable
93   # c : numeric
94   # point where the series expansion will take place
95   # from, to : numeric
96   # Interval of points to be plotted
97   #
98   # Returns
99   # -----
100  # void
101
102  x <- seq(from, to, length.out = 100)
103  yf <- f(x)
104
105  yp2 <- polyval(taylor(f, c, 2), x)
106  yp4 <- polyval(taylor(f, c, 4), x)
107  yp6 <- polyval(taylor(f, c, 6), x)
108  yp8 <- polyval(taylor(f, c, 8), x)
109
110  plot(
111    x,

```

```

112     yf,
113     xlab = "x",
114     ylab = "f(x)",
115     type = "l",
116     main = 'Taylor Series Approximation of f(x)',
117     col = "black",
118     lwd = 2
119 )
120
121 lines(x, yp2, col = "#c8e6c9")
122 lines(x, yp4, col = "#81c784")
123 lines(x, yp6, col = "#4caf50")
124 lines(x, yp8, col = "#388e3c")
125
126 legend(
127     'topleft',
128     inset = .05,
129     legend = c("TS_8_terms", "TS_6_terms", "TS_4_terms", "TS_2_terms", "f
130               (x)"),
131     col = c('#388e3c', '#4caf50', '#81c784', '#c8e6c9', 'black'),
132     lwd = c(1),
133     bty = 'n',
134     cex = .75
135 )
136
137 # -----
138
139 f0 <- function(x) {
140     res = sin(x)
141
142     return(res)
143 }
144
145
146 f1 <- function(x) {
147     res = exp(complex(real = 0, imaginary = 1)*x)
148
149     return(res)
150 }
151
152
153 # -----
154
155 taylorPlot(f0, 0, -6.6, 6.6)
156 taylorPlot(f1, 1, -2*pi, 2*pi)
157
158
159 #####
160 # PART 2 #####
161 # RIEMANN SUMS FUNCTION
162
163 riemann_sum <- function(f, a, b, n) {
164     # Compute the Riemann sum of f(x) over the interval [a,b].
165     #
166     # Parameters
167     # -----
168     # f : function
169     # Vectorized function of one variable
170     # a , b : numeric

```

```

171 # Endpoints of the interval [a,b]
172 # n : numeric
173 # Number of subintervals of equal length in the partition of [a,b]
174 #
175 # Returns
176 # -----
177 # numeric
178 # Underestimate and overestimate approximations of the integral given
    by the
179 # Riemann sum.
180
181 # initialize values
182 lower.sum <- 0
183
184 upper.sum <- 0
185
186 h <- (b - a) / n
187
188
189 # riemann right sum
190 for (i in n:1) {
191     x <- a + i * h
192
193     lower.sum <- lower.sum + f(x)
194 }
195 lower.sum <- h * lower.sum
196
197
198
199 # riemann left sum
200 for (i in 1:n) {
201     x <- b - i * h
202
203     upper.sum <- upper.sum + f(x)
204 }
205 upper.sum <- h * upper.sum
206
207
208
209 # let's plot the curve
210 integralPlot(
211     f = f,
212     a = a,
213     b = b,
214     title = expression(f(x))
215 )
216
217 # print/get riemann sum
218 cat(sprintf(
219     "The true value is between %f and %f.\n",
220     as.double(lower.sum),
221     as.double(upper.sum)
222 ))
223
224 return(c(lower.sum, upper.sum))
225
226 }
227
228 # -----
229

```

```

230 # let's generate some functions to test our algorithm
231 f2 <- function(x) {
232   res = x
233
234   return(res)
235 }
236
237
238 f3 <- function(x) {
239   res = 4 / (1 + x ^ 2)
240
241   return(res)
242 }
243
244
245 # ----
246
247 riemann_sum(f0, 0, pi / 2, 10)
248 riemann_sum(f2, 0, 1, 10000) # should be 0.5
249 riemann_sum(f3, 0, 1, 10000) # should be PI
250
251
252 #####
253 # PART 3 #####
254 # Integrate the function  $f(x)=\exp(x+x^2)$  from -2 to 2, using Riemann sums.
255 f4 <- function(x) {
256   res = exp(x + x ^ 2)
257
258   return(res)
259 }
260
261
262 # plot Taylor approximations
263 taylorPlot(f4, 0, -2.3, 1.3)
264
265 # compute riemann_sum for f4
266 riemann_sum(f4, -2, 2, 100000)
267 # let's verify our calculation using R's function
268 integrate(f4, lower = -2, upper = 2)
269
270
271 #####
272 # PART 4 #####
273 # RIEMANN SUMS 2D FUNCTION
274
275 riemann_sum_2d <- function(f, a, b, c, d, nx, ny) {
276   # Compute the Riemann sum of  $f(x,y)$  over the intervals  $[a,b]$  and  $[c,d]$ .
277   #
278   # Parameters
279   # -----
280   # f : function
281   # Vectorized function of one variable
282   # a , b : numeric
283   # Endpoints of the interval  $[a,b]$  (inner integral)
284   # c , d : numeric
285   # Endpoints of the interval  $[c,d]$  (outer integral)
286   # nx : numeric
287   # Number of subintervals of equal length in the partition of  $[a,b]$ 
288   # ny : numeric
289   # Number of subintervals of equal length in the partition of  $[c,d]$ 

```

```

290 #
291 # Returns
292 # -----
293 # numeric
294 # Approximations of the integral given by the Riemann 2D sum.
295
296 # initialize values
297 dx = (b - a) / nx
298 s = 0.0
299 x = a
300
301 dy = (d - c) / ny
302 y = c
303
304 # riemann 2D sum
305 for (i in 1:nx) {
306   for (j in 1:ny) {
307     x = a + dx / 2 + i * dx
308     y = c + dy / 2 + j * dy
309     f_i = f(x, y)
310     s = s + f_i * dx * dy
311   }
312 }
313
314 # print/get riemann sum
315 cat(sprintf("The true value is around %f.\n",
316            as.double(s)))
317
318 return(s)
319
320 }
321
322 # -----
323
324 f5 <- function(x, y) {
325   res = exp((x + y) ^ 2)
326
327   return(res)
328
329 }
330
331 # -----
332
333 # compute riemann_sum_2d for f5
334 riemann_sum_2d(f5, 0, 1, 0, 1, 1000, 1000)
335 # let's verify our calculation using R's function
336 integral2(f5, 0,1, 0,1)
337
338 #####
339 # PART 5 #####
340 # 2ND DERIVATIVE APPROXIMATION
341
342
343 derivative <- function(f, h) {
344   # Compute the Riemann sum of f(x,y) over the intervals [a,b] and [c,d].
345   #
346   # Parameters
347   # -----
348   # f : function f(x)
349   # Vectorized function of one variable

```

```

350 # h : numeric
351 # Let x now change by an amount h. h is the variable that approaches 0
352 #
353 # Returns
354 # -----
355 # function
356 # Approximations of the derivative of f, given a step h.
357
358 return(function(x) {
359   (f(x + h) - f(x)) / (h)
360 })
361 }
362
363 f6 <- function(x) {
364   res = x^4*sin(x)
365
366   return(res)
367 }
368
369
370 # df1 = d/dx(x^4 sin(x))
371 #       = x^3 (4 sin(x) + x cos(x))
372 df1 <- derivative(f6, 0.0001)
373
374 # df2 = d/dx(x^3 (4 sin(x) + x cos(x)))
375 #       = x^2 (8 x cos(x) - (x^2 - 12) sin(x))
376 df2 <- derivative(df1, 0.0001)
377
378 # Let's evaluate x=pi in the second derivative df2
379 df2(pi)
380
381 # Let's use the eval() function to verify our solution. The value should
382 #   be around df2(pi)
382 eval({ x <- pi; x^2*(8*x*cos(x) - (x^2 - 12)*sin(x))})

```



## C Full Output Log

```

1 > #####
2 > ## AUTHOR(S) :
3 > ##      Bruno Gonzalez Soria      (A01169284)
4 > ##      Antonio Osamu Katagiri Tanaka (A01212611)
5 > ##
6 > ## FILENAME :
7 > ##      Homework4.R
8 > ##
9 > ## DESCRIPTION :
10 > ##      Simulations (Ene 19 Gpo 1)
11 > ##      Homework 4
12 > ##
13 > ## NOTES :
14 > ##      - https://www.math.ubc.ca/~pwalls/math-python/integration/
      riemann-sums/
15 > ##      - https://activecalculus.org/multi/S-11-1-Double-Integrals-
      Rectangles.html
16 > ##      - http://math.colgate.edu/faculty/valente/math113/supplements/
      section151handout.pdf
17 > ##      - http://hplgit.github.io/Programming-for-Computations/pub/p4c/
      p4c-sphinx-Python/_pylight004.html
18 > ##      - https://rstudio-pubs-static.s3.amazonaws.com/131664_1858
      eec97df54c9b8d5edcd8b22e5818.html
19 > ##
20 > ## START DATE :
21 > ##      21 Feb 2019
22 > #####
23 >
24 > # Install required libraries
25 > #install.packages('pracma', dependencies=TRUE);
26 >
27 > # Plotting the Areas under Curves #####
28 > integralPlot <- function(f,
29 +                          a,
30 +                          b,
31 +                          from = a,
32 +                          to = b,
33 +                          title = NULL) {
34 +   # Plot the area under a function over the interval [a,b] between [
      from,to].
35 +   #
36 +   # Parameters
37 +   # -----
38 +   # f : function
39 +   # funtion to be plotted
40 +   # a , b : numeric
41 +   # Endpoints of the integral interval [a, b]
42 +   # from , to : numeric (optional)
43 +   # Endpoints of the plot in the x-axis [x min, x max]
44 +   # title : expression
45 +   # title of the plot
46 +   #
47 +   # Returns
48 +   # -----
49 +   # void
50 +
51 +   x <- seq(from, to, length.out = 100) # input continuum

```

```

52 + y <- f(x) # output
53 +
54 + # plot the curve
55 + plot(
56 +   x,
57 +   y,
58 +   xlim = c(from, to),
59 +   ylim = c(ifelse(min(y) < 0, min(y), 0), max(y)),
60 +   xlab = "x",
61 +   ylab = "f(x)",
62 +   main = title,
63 +   col.main = "#86B875",
64 +   type = "l",
65 +   lwd = 3,
66 +   col = "#86B875"
67 + )
68 +
69 + # area under the curve
70 + x <- seq(a, b, length.out = 100)
71 + y <- f(x)
72 + polygon(
73 +   c(x, b, a, a),
74 +   c(y, 0, 0, f(a)),
75 +   border = adjustcolor("#7DB0DD", alpha.f = 0.3),
76 +   col = adjustcolor("#7DB0DD", alpha.f = 0.3)
77 + )
78 + }
79 >
80 >
81 > #####
82 > # PART 1 #####
83 > # TAYLOR SERIES
84 >
85 > library(pracma)
86 >
87 > taylorPlot <- function(f, c, from, to) {
88 +   # Plot the Taylor approximations up to the 2nd, 4th, 6th and 8th
      terms
89 +   #
90 +   # Parameters
91 +   # -----
92 +   # f : function
93 +   # Vectorized function of one variable
94 +   # c : numeric
95 +   # point where the series expansion will take place
96 +   # from, to : numeric
97 +   # Interval of points to be plotted
98 +   #
99 +   # Returns
100 +   # -----
101 +   # void
102 +
103 +   x <- seq(from, to, length.out = 100)
104 +   yf <- f(x)
105 +
106 +   yp2 <- polyval(taylor(f, c, 2), x)
107 +   yp4 <- polyval(taylor(f, c, 4), x)
108 +   yp6 <- polyval(taylor(f, c, 6), x)
109 +   yp8 <- polyval(taylor(f, c, 8), x)
110 +

```

```

111 + plot(
112 +   x,
113 +   yf,
114 +   xlab = "x",
115 +   ylab = "f(x)",
116 +   type = "l",
117 +   main = 'Taylor Series Approximation of f(x)',
118 +   col = "black",
119 +   lwd = 2
120 + )
121 +
122 + lines(x, yp2, col = "#c8e6c9")
123 + lines(x, yp4, col = "#81c784")
124 + lines(x, yp6, col = "#4caf50")
125 + lines(x, yp8, col = "#388e3c")
126 +
127 + legend(
128 +   'topleft',
129 +   inset = .05,
130 +   legend = c("TS_8_terms", "TS_6_terms", "TS_4_terms", "TS_2_terms",
131 +     "f(x)"),
132 +   col = c('#388e3c', '#4caf50', '#81c784', '#c8e6c9', 'black'),
133 +   lwd = c(1),
134 +   bty = 'n',
135 +   cex = .75
136 + )
137 >
138 > # -----
139 >
140 > f0 <- function(x) {
141 +   res = sin(x)
142 +
143 +   return(res)
144 + }
145 >
146 >
147 > f1 <- function(x) {
148 +   res = exp(complex(real = 0, imaginary = 1)*x)
149 +
150 +   return(res)
151 + }
152 >
153 >
154 > # -----
155 >
156 > taylorPlot(f0, 0, -6.6, 6.6)
157 > taylorPlot(f1, 1, -2*pi, 2*pi)
158 Warning messages:
159 1: In xy.coords(x, y, xlabel, ylabel, log) :
160    imaginary parts discarded in coercion
161 2: In xy.coords(x, y) : imaginary parts discarded in coercion
162 3: In xy.coords(x, y) : imaginary parts discarded in coercion
163 4: In xy.coords(x, y) : imaginary parts discarded in coercion
164 5: In xy.coords(x, y) : imaginary parts discarded in coercion
165 >
166 >
167 > #####
168 > # PART 2 #####
169 > # RIEMANN SUMS FUNCTION

```

```

170 >
171 > riemann_sum <- function(f, a, b, n) {
172 +   # Compute the Riemann sum of f(x) over the interval [a,b].
173 +   #
174 +   # Parameters
175 +   # -----
176 +   # f : function
177 +   # Vectorized function of one variable
178 +   # a , b : numeric
179 +   # Endpoints of the interval [a,b]
180 +   # n : numeric
181 +   # Number of subintervals of equal length in the partition of [a,b]
182 +   #
183 +   # Returns
184 +   # -----
185 +   # numeric
186 +   # Underestimate and overestimate approximations of the integral given
187 +   # by the
188 +   # Riemann sum.
189 +   # initialize values
190 +   lower.sum <- 0
191 +
192 +   upper.sum <- 0
193 +
194 +   h <- (b - a) / n
195 +
196 +
197 +   # riemann right sum
198 +   for (i in n:1) {
199 +     x <- a + i * h
200 +
201 +     lower.sum <- lower.sum + f(x)
202 +
203 +   }
204 +   lower.sum <- h * lower.sum
205 +
206 +
207 +   # riemann left sum
208 +   for (i in 1:n) {
209 +     x <- b - i * h
210 +
211 +     upper.sum <- upper.sum + f(x)
212 +
213 +   }
214 +   upper.sum <- h * upper.sum
215 +
216 +
217 +   # let's plot the curve
218 +   integralPlot(
219 +     f = f,
220 +     a = a,
221 +     b = b,
222 +     title = expression(f(x))
223 +   )
224 +
225 +   # print/get riemann sum
226 +   cat(sprintf(
227 +     "The true value is between %f and %f.\n",
228 +     as.double(lower.sum),

```

```

229 +     as.double(upper.sum)
230 +   ))
231 +
232 +   return(c(lower.sum, upper.sum))
233 +
234 + }
235 >
236 > # -----
237 >
238 > # let's generate some functions to test our algorithm
239 > f2 <- function(x) {
240 +   res = x
241 +
242 +   return(res)
243 +
244 + }
245 >
246 > f3 <- function(x) {
247 +   res = 4 / (1 + x ^ 2)
248 +
249 +   return(res)
250 +
251 + }
252 >
253 > # -----
254 >
255 > riemann_sum(f0, 0, pi / 2, 10)
256 The true value is between 1.076483 and 0.919403.
257 [1] 1.0764828 0.9194032
258 > riemann_sum(f2, 0, 1, 10000) # should be 0.5
259 The true value is between 0.500050 and 0.499950.
260 [1] 0.50005 0.49995
261 > riemann_sum(f3, 0, 1, 10000) # should be PI
262 The true value is between 3.141493 and 3.141693.
263 [1] 3.141493 3.141693
264 >
265 >
266 > #####
267 > # PART 3 #####
268 > # Integrate the function  $f(x)=\exp(x+x^2)$  from -2 to 2, using Riemann
    sums.
269 > f4 <- function(x) {
270 +   res = exp(x + x ^ 2)
271 +
272 +   return(res)
273 +
274 + }
275 >
276 > # plot Taylor approximations
277 > taylorPlot(f4, 0, -2.3, 1.3)
278 >
279 > # compute riemann_sum for f4
280 > riemann_sum(f4, -2, 2, 100000)
281 The true value is between 93.170674 and 93.154833.
282 [1] 93.17067 93.15483
283 > # let's verify our calualtion using R's function
284 > integrate(f4, lower = -2, upper = 2)
285 93.16275 with absolute error < 0.00062
286 >
287 >

```

```

288 > #####
289 > # PART 4 #####
290 > # RIEMANN SUMS 2D FUNCTION
291 >
292 > riemann_sum_2d <- function(f, a, b, c, d, nx, ny) {
293 +   # Compute the derivative of a function.
294 +   #
295 +   # Parameters
296 +   # -----
297 +   # f : function
298 +   # Vectorized function of one variable
299 +   # a , b : numeric
300 +   # Endpoints of the interval [a,b] (inner integral)
301 +   # c , d : numeric
302 +   # Endpoints of the interval [c,d] (outer integral)
303 +   # nx : numeric
304 +   # Number of subintervals of equal length in the partition of [a,b]
305 +   # ny : numeric
306 +   # Number of subintervals of equal length in the partition of [c,d]
307 +   #
308 +   # Returns
309 +   # -----
310 +   # numeric
311 +   # Approximations of the integral given by the Riemann 2D sum.
312 +
313 +   # initialize values
314 +   dx = (b - a) / nx
315 +   s = 0.0
316 +   x = a
317 +
318 +   dy = (d - c) / ny
319 +   y = c
320 +
321 +   # riemann 2D sum
322 +   for (i in 1:nx) {
323 +     for (j in 1:ny) {
324 +       x = a + dx / 2 + i * dx
325 +       y = c + dy / 2 + j * dy
326 +       f_i = f(x, y)
327 +       s = s + f_i * dx * dy
328 +     }
329 +   }
330 +
331 +   # print/get riemann sum
332 +   cat(sprintf("The true value is around %f.\n",
333 +               as.double(s)))
334 +
335 +   return(s)
336 +
337 + }
338 >
339 > # ----
340 >
341 > f5 <- function(x, y) {
342 +   res = exp((x + y) ^ 2)
343 +
344 +   return(res)
345 +
346 + }
347 >

```

```

348 > # -----
349 >
350 > # compute riemann_sum_2d for f5
351 > riemann_sum_2d(f5, 0, 1, 0, 1, 1000, 1000)
352 The true value is around 4.926310.
353 [1] 4.92631
354 > # let's verify our calculation using R's function
355 > integral2(f5, 0,1, 0,1)
356 $Q
357 [1] 4.899159
358
359 $error
360 [1] 9.974762e-16
361
362 >
363 >
364 > #####
365 > # PART 5 #####
366 > # 2ND DERIVATIVE APPROXIMATION
367 >
368 > derivative <- function(f, h) {
369 +   # Compute the derivative of a function.
370 +   #
371 +   # Parameters
372 +   # -----
373 +   # f : function f(x)
374 +   # Vectorized function of one variable
375 +   # h : numeric
376 +   # Let x now change by an amount h. h is the variable that approaches
377 +   # 0
378 +   #
379 +   # Returns
380 +   # -----
381 +   # function
382 +   # Approximations of the derivative of f, given a step h.
383 +   #
384 +   # return(function(x) {
385 +   #   (f(x + h) - f(x - h)) / (2 * h)
386 +   # })
387 +   #
388 +   return(function(x) {
389 +     (f(x + h) - f(x)) / (h)
390 +   })
391 + }
392 >
393 > f6 <- function(x) {
394 +   res = x^4*sin(x)
395 +
396 +   return(res)
397 +
398 + }
399 >
400 > # df1 = d/dx(x^4 sin(x))
401 > #       = x^3 (4 sin(x) + x cos(x))
402 > df1 <- derivative(f6, 0.0001)
403 >
404 > # df2 = d/dx(x^3 (4 sin(x) + x cos(x)))
405 > #       = x^2 (8 x cos(x) - (x^2 - 12) sin(x))
406 > df2 <- derivative(df1, 0.0001)

```

```
407 >
408 > # Let's evaluate  $x=\pi$  in the second derivative  $df2$ 
409 > df2(pi)
410 [1] -248.076
411 >
412 > # Let's use the eval() function to verify our solution. The value should
      be around  $df2(\pi)$ 
413 > eval({ x <- pi; x^2*(8*x*cos(x) - (x^2 - 12)*sin(x))})
414 [1] -248.0502
415 >
```