

Computación Aplicada - Homework 04

Simulation – Basics & Integrals

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

February 27, 2019

Contents

| | | |
|----------|---|-----------|
| 1 | Problem I | 4 |
| 1.1 | Function 1 | 4 |
| 1.2 | Function 2 | 5 |
| 1.3 | Let's plot the approximations | 5 |
| 2 | Problem II | 8 |
| 3 | Problem III | 11 |
| 4 | Problem IV | 14 |
| A | <i>integralPlot</i> function | 17 |
| B | <i>derivativePlot</i> function | 18 |
| C | Full R Script | 20 |
| D | Full Output Log | 28 |

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 |
| 10 | Listing 11 output. | 16 |

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

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

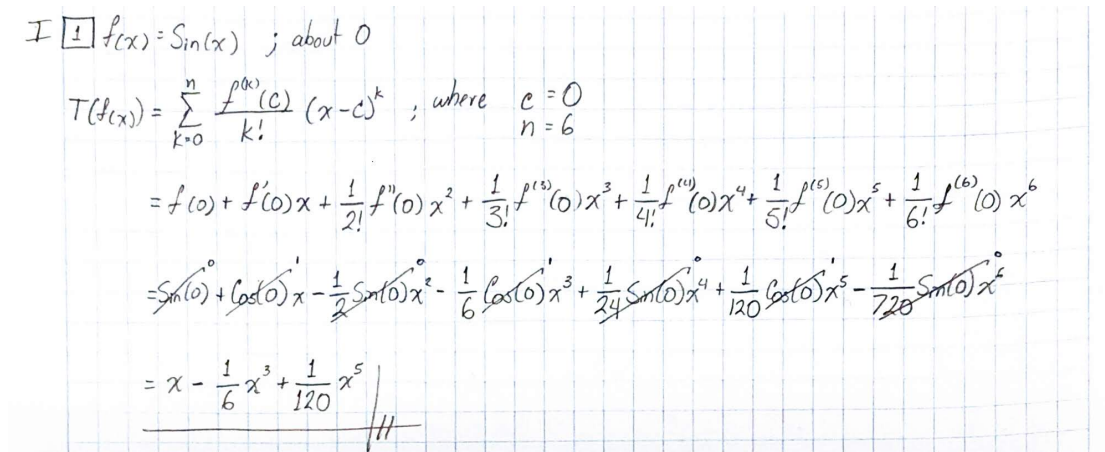
- Function 2

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:

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

$$T(f(x)) = \sum_{k=0}^n \frac{f^{(k)}(c)}{k!} (x-c)^k ; \text{ where } c=0, n=6$$
$$= f(0) + f'(0)x + \frac{1}{2!}f''(0)x^2 + \frac{1}{3!}f^{(3)}(0)x^3 + \frac{1}{4!}f^{(4)}(0)x^4 + \frac{1}{5!}f^{(5)}(0)x^5 + \frac{1}{6!}f^{(6)}(0)x^6$$
$$= \sin(0) + \cos(0)x - \frac{1}{2}\sin(0)x^2 - \frac{1}{6}\cos(0)x^3 + \frac{1}{24}\sin(0)x^4 + \frac{1}{120}\cos(0)x^5 - \frac{1}{720}\sin(0)x^6$$
$$= x - \frac{1}{6}x^3 + \frac{1}{120}x^5$$

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)

```

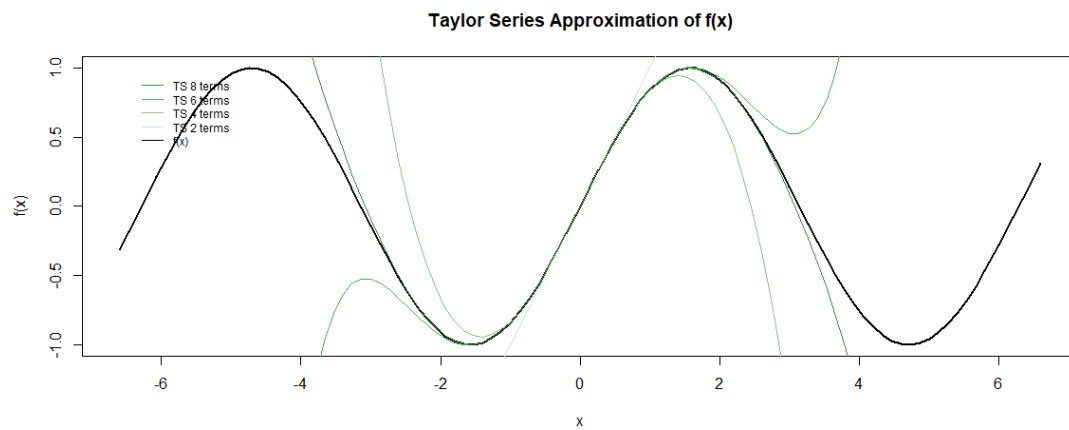


Figure 4: Listing 3 output; $f(x) = \sin(x)$ approximations, centered in 0.

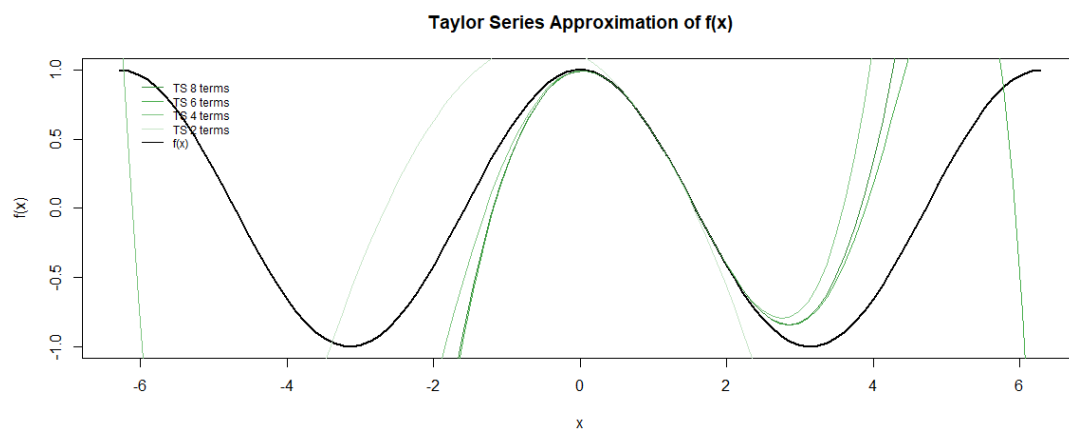


Figure 5: Listing 3 output; $f(x) = e^{ix}$ approximations, centered in 1.

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
```

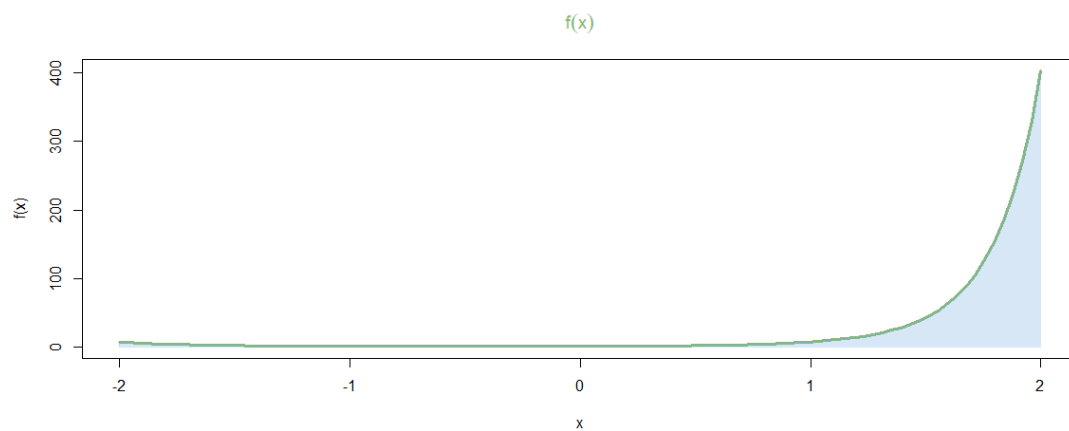


Figure 7: Listing 5 output; $\int_{-2}^2 e^{x+x^2} dx$.

* 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

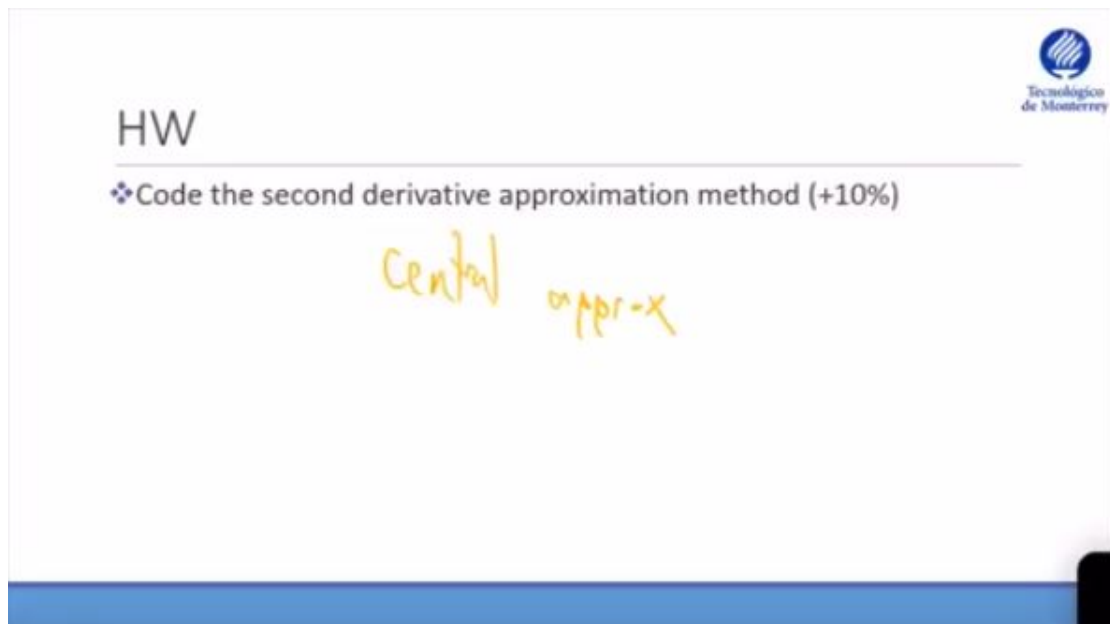


Figure 9: Problem 4 instructions.

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.

$$\frac{d^2}{dy^2} \frac{1}{25} x^3$$

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

```

1 f6 <- function(x) {
2   res = (x^3)/25
3
4   return(res)
5 }
6
7
8 true_d1f6 <- function(x) {
9   res = (3*x^2)/25
10
11   return(res)
12 }
13
14
15 true_d2f6 <- function(x) {
16   res = (6*x)/25
17
18   return(res)
19 }
20
21
22 # df1 = d/dx(x^4 sin(x))
23 #      = x^3 (4 sin(x) + x cos(x))
24 aprox_df1 <- derivative(f6, 0.01)
25
26 # df2 = d/dx(x^3 (4 sin(x) + x cos(x)))
27 #      = x^2 (8 x cos(x) - (x^2 - 12) sin(x))
28 aprox_df2 <- derivative(aprox_df1, 0.01)
29
30 # Let's evaluate x=pi in the second derivative df2
31 aprox_df2(pi)
32
33 # Let's use the eval() function to verify our solution. The value should
34   be around df2(pi)
35 true_d2f6(pi)
36
37 # Let's plot the real and the approximated derivatives (just to compare)
38 derivativePlot(f6, aprox_df1, aprox_df2, true_d1f6, true_d2f6, -0.75, 0.75)

```

Listing 12: "Listing 11 output"

```

1 > # Let's evaluate x=pi in the second derivative df2
2 > aprox_df2(pi)
3 [1] 0.7563822
4 >
5 > # Let's use the eval() function to verify our solution. The value should
   be around df2(pi)
6 > true_d2f6(pi)
7 [1] 0.7539822

```

Figure 10 shows the error between the true and approximate functions of the first and second derivatives of $f(x)$.

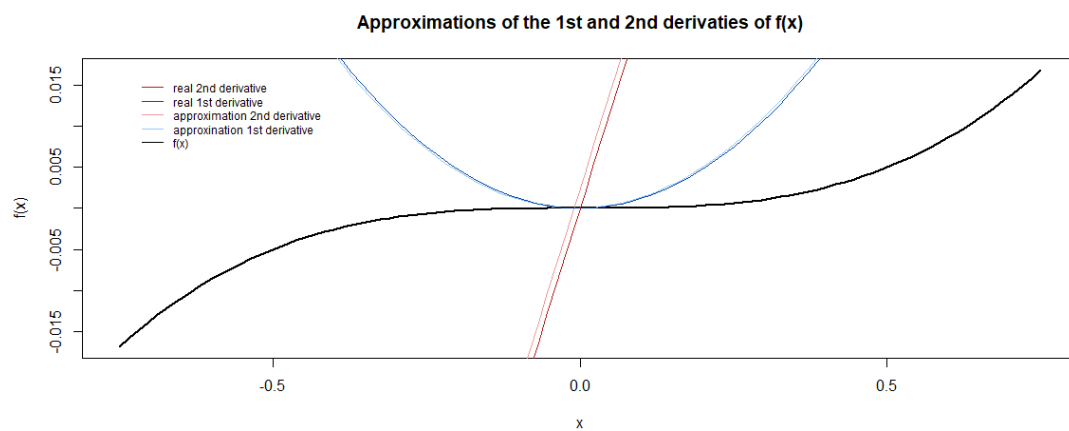


Figure 10: Listing 11 output.

* Appendix B implements the R function to generate plots similar to Figure 10.

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     #
11     # Parameters
12     # -----
13     # f : function
14     # funtion to be plotted
15     # a , b : numeric
16     # Endpoints of the integral interval [a, b]
17     # from , to : numeric (optional)
18     # Endpoints of the plot in the x-axis [x min, x max]
19     # title : expression
20     # title of the plot
21     #
22     # Returns
23     # -----
24     # void
25
26     x <- seq(from, to, length.out = 100) # input continuum
27     y <- f(x) # output
28
29     # plot the curve
30     plot(
31         x,
32         y,
33         xlim = c(from, to),
34         ylim = c(ifelse(min(y) < 0, min(y), 0), max(y)),
35         xlab = "x",
36         ylab = "f(x)",
37         main = title,
38         col.main = "#86B875",
39         type = "l",
40         lwd = 3,
41         col = "#86B875"
42     )
43
44     # area under the curve
45     x <- seq(a, b, length.out = 100)
46     y <- f(x)
47     polygon(
48         c(x, b, a, a),
49         c(y, 0, 0, f(a)),
50         border = adjustcolor("#7DB0DD", alpha.f = 0.3),
51         col = adjustcolor("#7DB0DD", alpha.f = 0.3)
52     )
53 }
```

B *derivativePlot* function

```

1  # Plotting the 1ST and 2ND DERIVATIVES #####
2  library(pracma)
3
4  derivativePlot <- function(f,
5                             ap_df1, ap_df2,
6                             tr_df1, tr_df2,
7                             from, to) {
8      # Plot the f and its 1st and 2nd derivatives between [from,to].
9      #
10     # Parameters
11     # -----
12     # f : function
13     # funtion to be plotted
14     # ap_df1 , ap_df2 : function
15     # aproxiomation of the 1st and 2nd derivatives of f (Normally, these are
16     # generated by the derivative function)
17     # tr_df1 , tr_df2 : function
18     # functions that represent the true 1st and 2nd derivatives of f
19     # from , to : numeric
20     # Endpoints of the plot in the x-axis [x min, x max]
21     #
22     # Returns
23     # -----
24     # void
25
26     x <- seq(from, to, length.out = 100)
27
28     yf <- f(x)
29     ap_yp2 <- ap_df1(x)
30     ap_yp4 <- ap_df2(x)
31     tr_yp6 <- tr_df1(x)
32     tr_yp8 <- tr_df2(x)
33
34     plot(
35         x, yf,
36         xlab = "x",
37         ylab = "f(x)",
38         type = "l",
39         main = 'Approximations of the 1st and 2nd derivatives of f(x)',
40         col = "black",
41         lwd = 2
42     )
43
44     lines(x, ap_yp2, col = "#90caf9") # blue lighten-3
45     lines(x, ap_yp4, col = "#ef9a9a") # red lighten-3
46     lines(x, tr_yp6, col = "#0d47a1") # blue darken-4
47     lines(x, tr_yp8, col = "#b71c1c") # red darken-4
48
49     legend(
50         'topleft', inset = .05,
51         legend = c("real 2nd derivative", "real 1st derivative", "
                    approximation 2nd derivative", "approximation 1st derivative", "f
                    (x)"),
52         col = c('#b71c1c', '#0d47a1', '#ef9a9a', '#90caf9', 'black'),
53         lwd = c(1), bty = 'n', cex = .75
54     )
55 }

```

C 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-
15 **        sums/
16 **      - https://activecalculus.org/multi/S-11-1-Double-Integrals-
17 **        Rectangles.html
18 **      - http://math.colgate.edu/faculty/valente/math113/supplements/
19 **        section151handout.pdf
20 **      - http://hplgit.github.io/Programming-for-Computations/pub/p4c/p4c
21 **        -sphinx-Python/.pylight004.html
22 **      - https://rstudio-pubs-static.s3.amazonaws.com/131664_1858
23 **        eec97df54c9b8d5edcd8b22e5818.html
24 **
25 ** START DATE :
26 **      21 Feb 2019
27 *****
28
29 # Install required libraries
30 #install.packages('pracma', dependencies=TRUE);
31
32 # Plotting the Areas under Curves #####
33 integralPlot <- function(f,
34                          a,
35                          b,
36                          from = a,
37                          to = b,
38                          title = NULL) {
39   # Plot the area under a function over the interval [a,b] between [from,
40   #   to].
41   #
42   # Parameters
43   # -----
44   # f : function
45   # funtion to be plotted
46   # a , b : numeric
47   # Endpoints of the integral interval [a, b]
48   # from , to : numeric (optional)
49   # Endpoints of the plot in the x-axis [x min, x max]
50   # title : expression
51   # title of the plot
52   #
53   # Returns
54   # -----
55   # void
56
57   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 # Plotting the 1ST and 2ND DERIVATIVES #####
81 library(pracma)
82
83 derivativePlot <- function(f,
84                             ap_df1,
85                             ap_df2,
86                             tr_df1,
87                             tr_df2,
88                             from,
89                             to) {
90   # Plot the f and its 1st and 2nd derivatives between [from,to].
91   #
92   # Parameters
93   # -----
94   # f : function
95   # funtion to be plotted
96   # ap_df1 , ap_df2 : function
97   # aproximation of the 1st and 2nd derivatives of f (Normally, these
98   # are generated by the derivative function)
99   # tr_df1 , tr_df2 : function
100  # functions that represent the true 1st and 2nd derivatives of f
101  # from , to : numeric
102  # Endpoints of the plot in the x-axis [x min, x max]
103  #
104  # Returns
105  # -----
106  # void
107
108  x <- seq(from, to, length.out = 100)
109
110  yf <- f(x)
111  ap_yp2 <- ap_df1(x)

```

```

112 ap_yp4 <- ap_df2(x)
113 tr_yp6 <- tr_df1(x)
114 tr_yp8 <- tr_df2(x)
115
116 plot(
117   x,
118   yf,
119   xlab = "x",
120   ylab = "f(x)",
121   type = "l",
122   main = 'Approximations of the 1st and 2nd derivatives of f(x)',
123   col = "black",
124   lwd = 2
125 )
126
127 lines(x, ap_yp2, col = "#90caf9") # blue lighten-3
128 lines(x, ap_yp4, col = "#ef9a9a") # red lighten-3
129 lines(x, tr_yp6, col = "#0d47a1") # blue darken-4
130 lines(x, tr_yp8, col = "#b71c1c") # red darken-4
131
132 legend(
133   'topleft',
134   inset = .05,
135   legend = c("real_2nd_derivative", "real_1st_derivative", "
               approximation_2nd_derivative", "approximation_1st_derivative", "f
               (x)"),
136   col = c('#b71c1c', '#0d47a1', '#ef9a9a', '#90caf9', 'black'),
137   lwd = c(1),
138   bty = 'n',
139   cex = .75
140 )
141 }
142
143 #####
144 # PART 1 #####
145 # TAYLOR SERIES
146
147 library(pracma)
148
149 taylorPlot <- function(f, c, from, to) {
150   # Plot the Taylor approximations up to the 2nd, 4th, 6th and 8th terms
151   #
152   # Parameters
153   # -----
154   # f : function
155   # Vectorized function of one variable
156   # c : numeric
157   # point where the series expansion will take place
158   # from, to : numeric
159   # Interval of points to be plotted
160   #
161   # Returns
162   # -----
163   # void
164
165   x <- seq(from, to, length.out = 100)
166   yf <- f(x)
167
168   yp2 <- polyval(taylor(f, c, 2), x)
169   yp4 <- polyval(taylor(f, c, 4), x)

```

```

170 yp6 <- polyval(taylor(f, c, 6), x)
171 yp8 <- polyval(taylor(f, c, 8), x)
172
173 plot(
174   x,
175   yf,
176   xlab = "x",
177   ylab = "f(x)",
178   type = "l",
179   main = 'Taylor Series Approximation of f(x)',
180   col = "black",
181   lwd = 2
182 )
183
184 lines(x, yp2, col = "#c8e6c9")
185 lines(x, yp4, col = "#81c784")
186 lines(x, yp6, col = "#4caf50")
187 lines(x, yp8, col = "#388e3c")
188
189 legend(
190   'topleft',
191   inset = .05,
192   legend = c("TS_8_terms", "TS_6_terms", "TS_4_terms", "TS_2_terms", "f
193             (x)"),
194   col = c('#388e3c', '#4caf50', '#81c784', '#c8e6c9', 'black'),
195   lwd = c(1),
196   bty = 'n',
197   cex = .75
198 )
199
200 # -----
201
202 f0 <- function(x) {
203   res = sin(x)
204
205   return(res)
206 }
207
208
209 f1 <- function(x) {
210   res = exp(complex(real = 0, imaginary = 1)*x)
211
212   return(res)
213 }
214
215
216 # -----
217
218 taylorPlot(f0, 0, -6.6, 6.6)
219 taylorPlot(f1, 1, -2*pi, 2*pi)
220
221
222 #####
223 # PART 2 #####
224 # RIEMANN SUMS FUNCTION
225
226 riemann_sum <- function(f, a, b, n) {
227   # Compute the Riemann sum of f(x) over the interval [a,b].
228   #

```

```

229 # Parameters
230 # -----
231 # f : function
232 # Vectorized function of one variable
233 # a , b : numeric
234 # Endpoints of the interval [a,b]
235 # n : numeric
236 # Number of subintervals of equal length in the partition of [a,b]
237 #
238 # Returns
239 # -----
240 # numeric
241 # Underestimate and overestimate approximations of the integral given
    by the
242 # Riemann sum.
243
244 # initialize values
245 lower.sum <- 0
246
247 upper.sum <- 0
248
249 h <- (b - a) / n
250
251
252 # riemann right sum
253 for (i in n:1) {
254     x <- a + i * h
255
256     lower.sum <- lower.sum + f(x)
257
258 }
259 lower.sum <- h * lower.sum
260
261
262 # riemann left sum
263 for (i in 1:n) {
264     x <- b - i * h
265
266     upper.sum <- upper.sum + f(x)
267
268 }
269 upper.sum <- h * upper.sum
270
271
272 # let's plot the curve
273 integralPlot(
274     f = f,
275     a = a,
276     b = b,
277     title = expression(f(x))
278 )
279
280 # print/get riemann sum
281 cat(sprintf(
282     "The true value is between %f and %f.\n",
283     as.double(lower.sum),
284     as.double(upper.sum)
285 ))
286
287 return(c(lower.sum, upper.sum))

```

```

288 }
289
290
291 # -----
292
293 # let's generate some functions to test our algorithm
294 f2 <- function(x) {
295   res = x
296
297   return(res)
298 }
299
300
301 f3 <- function(x) {
302   res = 4 / (1 + x ^ 2)
303
304   return(res)
305 }
306
307
308 # -----
309
310 riemann_sum(f0, 0, pi / 2, 10)
311 riemann_sum(f2, 0, 1, 10000) # should be 0.5
312 riemann_sum(f3, 0, 1, 10000) # should be PI
313
314
315 #####
316 # PART 3 #####
317 # Integrate the function  $f(x)=\exp(x+x^2)$  from -2 to 2, using Rieman sums.
318 f4 <- function(x) {
319   res = exp(x + x ^ 2)
320
321   return(res)
322 }
323
324
325 # plot Taylor approximations
326 taylorPlot(f4, 0, -2.3, 1.3)
327
328 # compute riemann_sum for f4
329 riemann_sum(f4, -2, 2, 100000)
330 # let's verify our calualtion using R's function
331 integrate(f4, lower = -2, upper = 2)
332
333
334 #####
335 # PART 4 #####
336 # RIEMANN SUMS 2D FUNCTION
337
338 riemann_sum_2d <- function(f, a, b, c, d, nx, ny) {
339   # Compute the Riemann sum of  $f(x,y)$  over the intervals  $[a,b]$  and  $[c,d]$ .
340   #
341   # Parameters
342   # -----
343   # f : function
344   # Vectorized function of one variable
345   # a , b : numeric
346   # Endpoints of the interval  $[a,b]$  (inner integral)
347   # c , d : numeric

```



```

348 # Endpoints of the interval [c,d] (outer integral)
349 # nx : numeric
350 # Number of subintervals of equal length in the partition of [a,b]
351 # ny : numeric
352 # Number of subintervals of equal length in the partition of [c,d]
353 #
354 # Returns
355 # -----
356 # numeric
357 # Approximations of the integral given by the Riemann 2D sum.
358
359 # initialize values
360 dx = (b - a) / nx
361 s = 0.0
362 x = a
363
364 dy = (d - c) / ny
365 y = c
366
367 # riemann 2D sum
368 for (i in 1:nx) {
369   for (j in 1:ny) {
370     x = a + dx / 2 + i * dx
371     y = c + dy / 2 + j * dy
372     f_i = f(x, y)
373     s = s + f_i * dx * dy
374   }
375 }
376
377 # print/get riemann sum
378 cat(sprintf("The true value is around %f.\n",
379            as.double(s)))
380
381 return(s)
382
383 }
384
385 # -----
386
387 f5 <- function(x, y) {
388   res = exp((x + y) ^ 2)
389
390   return(res)
391 }
392
393 # -----
394
395 # compute riemann_sum_2d for f5
396 riemann_sum_2d(f5, 0, 1, 0, 1, 1000, 1000)
397 # let's verify our calualtion using R's function
398 integral2(f5, 0,1, 0,1)
399
400
401
402 #####
403 # PART 5 #####
404 # 2ND DERIVATIVE APPROXIMATION
405
406 derivative <- function(f, h) {
407   # Compute the derivative of a function.

```

```

408 #
409 # Parameters
410 # -----
411 # f : function f(x)
412 # Vectorized function of one variable
413 # h : numeric
414 # Let x now change by an amount h. h is the variable that approaches 0
415 #
416 # Returns
417 # -----
418 # function
419 # Approximations of the derivative of f, given a step h.
420
421 return(function(x) {
422     (f(x + h) - f(x)) / (h)
423 })
424 }
425
426 f6 <- function(x) {
427     res = (x^3)/25
428
429     return(res)
430 }
431
432 true_d1f6 <- function(x) {
433     res = (3*x^2)/25
434
435     return(res)
436 }
437
438 true_d2f6 <- function(x) {
439     res = (6*x)/25
440
441     return(res)
442 }
443
444 }
445
446 # df1 = d/dx(x^4 sin(x))
447 #      = x^3 (4 sin(x) + x cos(x))
448 aprox_df1 <- derivative(f6, 0.01)
449
450 # df2 = d/dx(x^3 (4 sin(x) + x cos(x)))
451 #      = x^2 (8 x cos(x) - (x^2 - 12) sin(x))
452 aprox_df2 <- derivative(aprox_df1, 0.01)
453
454 # Let's evaluate x=pi in the second derivative df2
455 aprox_df2(pi)
456
457 # Let's use the eval() function to verify our solution. The value should
458 # be around df2(pi)
459 true_d2f6(pi)
460
461 # Let's plot the real and the approximated derivatives (just to compare)
462 derivativePlot(f6, aprox_df1, aprox_df2, true_d1f6, true_d2f6, -0.75, 0.75)

```

D 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 > # Plotting the 1ST and 2ND DERIVATIVES #####
81 > library(pracma)
82 >
83 > derivativePlot <- function(f,
84 +   ap_df1,
85 +   ap_df2,
86 +   tr_df1,
87 +   tr_df2,
88 +   from,
89 +   to) {
90 +   # Plot the f and its 1st and 2nd derivatives between [from,to].
91 +   #
92 +   # Parameters
93 +   # -----
94 +   # f : function
95 +   # funtion to be plotted
96 +   # ap_df1 , ap_df2 : function
97 +   # aproximation of the 1st and 2nd derivatives of f (Normally, these
98 +   # are generated by the derivative function)
99 +   # tr_df1 , tr_df2 : function
100 +   # functions that represent the true 1st and 2nd derivatives of f
101 +   # from , to : numeric
102 +   # Endpoints of the plot in the x-axis [x min, x max]
103 +   #
104 +   # Returns
105 +   # -----
106 +   # void
107 +
108 +   x <- seq(from, to, length.out = 100)
109 +
110 +   yf <- f(x)
111 +   ap_yp2 <- ap_df1(x)

```

```

112 + ap_yp4 <- ap_df2(x)
113 + tr_yp6 <- tr_df1(x)
114 + tr_yp8 <- tr_df2(x)
115 +
116 + plot(
117 +   x,
118 +   yf,
119 +   xlab = "x",
120 +   ylab = "f(x)",
121 +   type = "l",
122 +   main = 'Approximations of the 1st and 2nd derivatives of f(x)',
123 +   col = "black",
124 +   lwd = 2
125 + )
126 +
127 + lines(x, ap_yp2, col = "#90caf9") # blue lighten-3
128 + lines(x, ap_yp4, col = "#ef9a9a") # red lighten-3
129 + lines(x, tr_yp6, col = "#0d47a1") # blue darken-4
130 + lines(x, tr_yp8, col = "#b71c1c") # red darken-4
131 +
132 + legend(
133 +   'topleft',
134 +   inset = .05,
135 +   legend = c("real_2nd_derivative", "real_1st_derivative", "
approximation_2nd_derivative", "approximation_1st_derivative", "f(x)"
),
136 +   col = c('#b71c1c', '#0d47a1', '#ef9a9a', '#90caf9', 'black'),
137 +   lwd = c(1),
138 +   bty = 'n',
139 +   cex = .75
140 + )
141 + }
142 >
143 > #####
144 > # PART 1 #####
145 > # TAYLOR SERIES
146 >
147 > library(pracma)
148 >
149 > taylorPlot <- function(f, c, from, to) {
150 +   # Plot the Taylor approximations up to the 2nd, 4th, 6th and 8th
terms
151 +   #
152 +   # Parameters
153 +   # -----
154 +   # f : function
155 +   # Vectorized function of one variable
156 +   # c : numeric
157 +   # point where the series expansion will take place
158 +   # from, to : numeric
159 +   # Interval of points to be plotted
160 +   #
161 +   # Returns
162 +   # -----
163 +   # void
164 +
165 +   x <- seq(from, to, length.out = 100)
166 +   yf <- f(x)
167 +
168 +   yp2 <- polyval(taylor(f, c, 2), x)

```

```

169 +   yp4 <- polyval(taylor(f, c, 4), x)
170 +   yp6 <- polyval(taylor(f, c, 6), x)
171 +   yp8 <- polyval(taylor(f, c, 8), x)
172 +
173 +   plot(
174 +     x,
175 +     yf,
176 +     xlab = "x",
177 +     ylab = "f(x)",
178 +     type = "l",
179 +     main = 'Taylor Series Approximation of f(x)',
180 +     col = "black",
181 +     lwd = 2
182 +   )
183 +
184 +   lines(x, yp2, col = "#c8e6c9")
185 +   lines(x, yp4, col = "#81c784")
186 +   lines(x, yp6, col = "#4caf50")
187 +   lines(x, yp8, col = "#388e3c")
188 +
189 +   legend(
190 +     'topleft',
191 +     inset = .05,
192 +     legend = c("TS_8_terms", "TS_6_terms", "TS_4_terms", "TS_2_terms",
193 +       "f(x)"),
194 +     col = c('#388e3c', '#4caf50', '#81c784', '#c8e6c9', 'black'),
195 +     lwd = c(1),
196 +     bty = 'n',
197 +     cex = .75
198 +   )
199 + }
200 >
201 > # -----
202 > f0 <- function(x) {
203 +   res = sin(x)
204 +
205 +   return(res)
206 + }
207 >
208 >
209 > f1 <- function(x) {
210 +   res = exp(complex(real = 0, imaginary = 1)*x)
211 +
212 +   return(res)
213 + }
214 >
215 >
216 > # -----
217 >
218 > taylorPlot(f0, 0, -6.6, 6.6)
219 > taylorPlot(f1, 1, -2*pi, 2*pi)
220 Warning messages:
221 1: In xy.coords(x, y, xlabel, ylabel, log) :
222    imaginary parts discarded in coercion
223 2: In xy.coords(x, y) : imaginary parts discarded in coercion
224 3: In xy.coords(x, y) : imaginary parts discarded in coercion
225 4: In xy.coords(x, y) : imaginary parts discarded in coercion
226 5: In xy.coords(x, y) : imaginary parts discarded in coercion
227 >

```

```

228 >
229 > #####
230 > # PART 2 #####
231 > # RIEMANN SUMS FUNCTION
232 >
233 > riemann_sum <- function(f, a, b, n) {
234 +   # Compute the Riemann sum of  $f(x)$  over the interval  $[a,b]$ .
235 +   #
236 +   # Parameters
237 +   # -----
238 +   # f : function
239 +   # Vectorized function of one variable
240 +   # a , b : numeric
241 +   # Endpoints of the interval  $[a,b]$ 
242 +   # n : numeric
243 +   # Number of subintervals of equal length in the partition of  $[a,b]$ 
244 +   #
245 +   # Returns
246 +   # -----
247 +   # numeric
248 +   # Underestimate and overestimate approximations of the integral given
    by the
249 +   # Riemann sum.
250 +
251 +   # initialize values
252 +   lower.sum <- 0
253 +
254 +   upper.sum <- 0
255 +
256 +   h <- (b - a) / n
257 +
258 +
259 +   # riemann right sum
260 +   for (i in n:1) {
261 +     x <- a + i * h
262 +
263 +     lower.sum <- lower.sum + f(x)
264 +
265 +   }
266 +   lower.sum <- h * lower.sum
267 +
268 +
269 +   # riemann left sum
270 +   for (i in 1:n) {
271 +     x <- b - i * h
272 +
273 +     upper.sum <- upper.sum + f(x)
274 +
275 +   }
276 +   upper.sum <- h * upper.sum
277 +
278 +
279 +   # let's plot the curve
280 +   integralPlot(
281 +     f = f,
282 +     a = a,
283 +     b = b,
284 +     title = expression(f(x))
285 +   )
286 +

```

```

287 + # print/get riemann sum
288 + cat(sprintf(
289 +   "The true value is between %f and %f.\n",
290 +   as.double(lower.sum),
291 +   as.double(upper.sum)
292 + ))
293 +
294 + return(c(lower.sum, upper.sum))
295 +
296 + }
297 >
298 > # -----
299 >
300 > # let's generate some functions to test our algorithm
301 > f2 <- function(x) {
302 +   res = x
303 +
304 +   return(res)
305 +
306 + }
307 >
308 > f3 <- function(x) {
309 +   res = 4 / (1 + x ^ 2)
310 +
311 +   return(res)
312 +
313 + }
314 >
315 > # -----
316 >
317 > riemann_sum(f0, 0, pi / 2, 10)
318 The true value is between 1.076483 and 0.919403.
319 [1] 1.0764828 0.9194032
320 > riemann_sum(f2, 0, 1, 10000) # should be 0.5
321 The true value is between 0.500050 and 0.499950.
322 [1] 0.50005 0.49995
323 > riemann_sum(f3, 0, 1, 10000) # should be PI
324 The true value is between 3.141493 and 3.141693.
325 [1] 3.141493 3.141693
326 >
327 >
328 > #####
329 > # PART 3 #####
330 > # Integrate the function  $f(x)=\exp(x+x^2)$  from -2 to 2, using Riemann
    sums.
331 > f4 <- function(x) {
332 +   res = exp(x + x ^ 2)
333 +
334 +   return(res)
335 +
336 + }
337 >
338 > # plot Taylor approximations
339 > taylorPlot(f4, 0, -2.3, 1.3)
340 >
341 > # compute riemann_sum for f4
342 > riemann_sum(f4, -2, 2, 100000)
343 The true value is between 93.170674 and 93.154833.
344 [1] 93.17067 93.15483
345 > # let's verify our calualtion using R's function

```



```

346 > integrate(f4, lower = -2, upper = 2)
347 93.16275 with absolute error < 0.00062
348 >
349 >
350 > #####
351 > # PART 4 #####
352 > # RIEMANN SUMS 2D FUNCTION
353 >
354 > riemann_sum_2d <- function(f, a, b, c, d, nx, ny) {
355 +   # Compute the Riemann sum of f(x,y) over the intervals [a,b] and [c,d]
356 +   #
357 +   # Parameters
358 +   # -----
359 +   # f : function
360 +   # Vectorized function of one variable
361 +   # a , b : numeric
362 +   # Endpoints of the interval [a,b] (inner integral)
363 +   # c , d : numeric
364 +   # Endpoints of the interval [c,d] (outer integral)
365 +   # nx : numeric
366 +   # Number of subintervals of equal length in the partition of [a,b]
367 +   # ny : numeric
368 +   # Number of subintervals of equal length in the partition of [c,d]
369 +   #
370 +   # Returns
371 +   # -----
372 +   # numeric
373 +   # Approximations of the integral given by the Riemann 2D sum.
374 +
375 +   # initialize values
376 +   dx = (b - a) / nx
377 +   s = 0.0
378 +   x = a
379 +
380 +   dy = (d - c) / ny
381 +   y = c
382 +
383 +   # riemann 2D sum
384 +   for (i in 1:nx) {
385 +     for (j in 1:ny) {
386 +       x = a + dx / 2 + i * dx
387 +       y = c + dy / 2 + j * dy
388 +       f_i = f(x, y)
389 +       s = s + f_i * dx * dy
390 +     }
391 +   }
392 +
393 +   # print/get riemann sum
394 +   cat(sprintf("The true value is around %f.\n",
395 +     as.double(s)))
396 +
397 +   return(s)
398 + }
399 >
400 >
401 > # -----
402 >
403 > f5 <- function(x, y) {
404 +   res = exp((x + y) ^ 2)

```

```

405 +
406 +   return(res)
407 +
408 + }
409 >
410 > # -----
411 >
412 > # compute riemann_sum_2d for f5
413 > riemann_sum_2d(f5, 0, 1, 0, 1, 1000, 1000)
414 The true value is around 4.926310.
415 [1] 4.92631
416 > # let's verify our calualtion using R's function
417 > integral2(f5, 0,1, 0,1)
418 $Q
419 [1] 4.899159
420
421 $error
422 [1] 9.974762e-16
423
424 >
425 >
426 > #####
427 > # PART 5 #####
428 > # 2ND DERIVATIVE APPROXIMATION
429 >
430 > derivative <- function(f, h) {
431 +   # Compute the derivative of a function.
432 +   #
433 +   # Parameters
434 +   # -----
435 +   # f : function f(x)
436 +   # Vectorized function of one variable
437 +   # h : numeric
438 +   # Let x now change by an amount h. h is the variable that approaches
439 +   #
440 +   # Returns
441 +   # -----
442 +   # function
443 +   # Approximations of the derivative of f, given a step h.
444 +
445 +   return(function(x) {
446 +     (f(x + h) - f(x)) / (h)
447 +   })
448 + }
449 >
450 > f6 <- function(x) {
451 +   res = (x^3)/25
452 +
453 +   return(res)
454 +
455 + }
456 >
457 > true_d1f6 <- function(x) {
458 +   res = (3*x^2)/25
459 +
460 +   return(res)
461 +
462 + }
463 >

```

```

464 > true_d2f6 <- function(x) {
465 +   res = (6*x)/25
466 +
467 +   return(res)
468 +
469 + }
470 >
471 > # df1 = d/dx(x^4 sin(x))
472 > #       = x^3 (4 sin(x) + x cos(x))
473 > aprox_df1 <- derivative(f6, 0.01)
474 >
475 > # df2 = d/dx(x^3 (4 sin(x) + x cos(x)))
476 > #       = x^2 (8 x cos(x) - (x^2 - 12) sin(x))
477 > aprox_df2 <- derivative(aprox_df1, 0.01)
478 >
479 > # Let's evaluate x=pi in the second derivative df2
480 > aprox_df2(pi)
481 [1] 0.7563822
482 >
483 > # Let's use the eval() funtion to verify our solution. The value should
484   be around df2(pi)
485 > true_d2f6(pi)
486 [1] 0.7539822
487 >
488 > # Let's plot the real and the aproximated derivatives (just to compare)
489 > derivativePlot(f6, aprox_df1, aprox_df2, true_d1f6, true_d2f6, -0.75, 0.75)

```