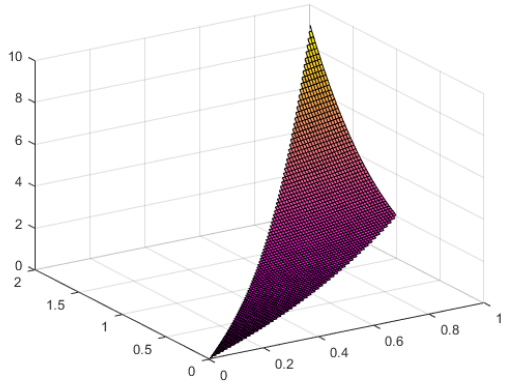
Numerical Analysis

Calculating a Double Integral Using Different Methods

Aristotle University of Thessaloniki

Computational Physics

Professor of Subject: Dr. Kosmas Kosmidis



Papadopoulou Martha

AEM 4418

15/1/2024

Contents

1 Introduction

This assignment explores the calculation of the following integral:

$$\int_0^1 \int_x^{2x} x^2 + y^3 \, dy \, dx \tag{1}$$

The computation is conducted using three methods: analytical, standard numerical integration and Monte Carlo integration. All methods are computed through a MATLAB code. The analytical method includes both a coded procedure and a manual handwritten calculation. Furthermore, it briefly analyses the properties of the integral and the integrand $f(x, y) = x^2 + y^3$. The standard numerical integration is implemented using the Simpson's rule, this selection of method is explained in that section. Lastly, the Monte Carlo method is employed. To assess the accuracy of the results, the method is computed multiple times to examine the variance.

2 Analytical Calculation

2.1 Methodology of Analytical Calculation

Before the actual calculation, it is essential to provide some information about the nature of the integrand $f(x, y) = x^2 + y^3$. As is evident, the function is a two-variable polynomial. It consists of power functions of the variables, with the highest degree being 3rd. As a polynomial function, it is continuous everywhere. The only critical point of the function occurs when $\partial f/\partial x = 2x$ and $\partial f/\partial y = 3y^2$ are both equal to zeros, which happens when $x = 0$ and $y = 0$.

The region of integration is defined by $y \in [x, 2x]$ and $x \in [0, 1]$. Within those bounds, the function does not exhibit symmetry with respect to any axis. The numerical result of the integral within these bounds represents the volume under the defined $f(x, y)$ surface.

The calculation of the integral (1) is straightforward, given that the function is a polynomial. Firstly, the inner interval is calculated with respect to y, followed by the outer interval with respect to x.

$$\int_0^1 \int_x^{2x} x^2 + y^3 \, dy \, dx = \int_0^1 \left[x^2 \, y + \frac{y^4}{4} \right]_x^{2x} dx = \int_0^1 \left(x^3 + \frac{15}{4} x^4 \right) dx = \left[\frac{x^4}{4} + \frac{3\,x^5}{4} \right]_0^1 = 1 \tag{2}$$

The result of the double integral within the specified region in the xy-plane is 1.

2.2 Code Overview of Analytical Calculation

Prior to any calculations, it is essential to define the function and the variables. Thus, the MATLAB 'syms' function is employed to symbolically define the x and y variables, thereby enabling symbolic computations. Afterwards, the function $f(x, y)$ is also symbolically defined, given the symbolic nature of x and y. The last aspects that require definition are the boundaries for both x and y. As

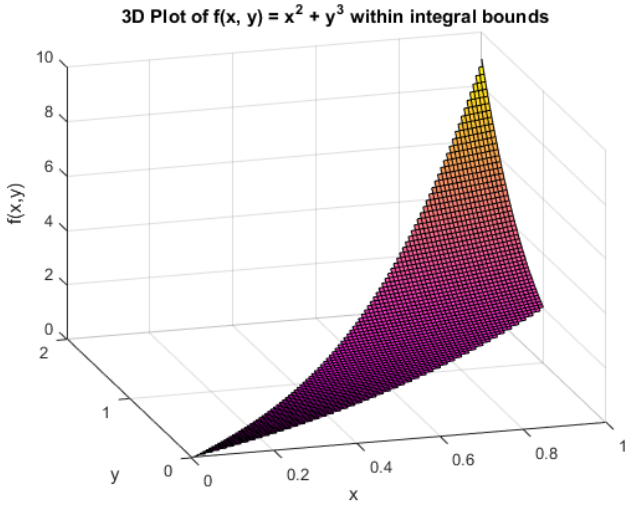


Figure 1: 3D plot of $f(x, y) = x^2 + y^3$. In the x-axis are the x values that range from 0 to 1. In the y-axis are the y values that range from x to $2x$. In the z-axis are the corresponding values of $f(x, y)$. This represents the surface that undergoes integration. The infinitesimal surface elements are purple near 0 and yellow near 9.

mentioned above, the y variable ranges from a lower value of x and an upper value of $2x$, while the x variable spans from a lower value of 0 and an upper value of 1.

Another step that needs to be performed before the integral calculation is the creation of a 3D plot of the $f(x, y)$ function. This allows us to visualize the nature of the function. Firstly, the 'linspace' MATLAB function is used to generate 100 values within the bounds of x and y. To assign values to y, the 'subs' function is employed to compute the lower value using the minimum x ($y = 0$) and the upper value using the maximum x ($y = 2$). Subsequently, by utilizing the 'meshgrid' function, two matrices are generated to assign values to the $f(x, y)$ function and then create the 3D plot using the 'surf' function.

The analytical integration is simple in MATLAB. Following the same rationale as in the manual computation, the inner integral is calculated and subsequently the outer one. For the inner integral, the result is stored in a new function $g(x)$, utilizing the MATLAB function 'int' to integrate $f(x, y)$ with respect to y for the specified bounds. Similarly, the $g(x)$ function is integrated with respect to x and the result is displayed.

2.3 Results of Analytical Calculation

Upon executing the code, the initial output is the 3D plot shown is Figure 1. From this figure, the surface that is subjected to integration is clearly observable. The earlier statement that the surface does not exhibit any symmetry for the specified bounds holds true. As anticipated, the overall shape is curved, particularly as the function approaches zero. Also, it is evident that the maximum value that the $f(x, y)$ function can attain is 9, while the minimum is 0.

The result obtained from the analytical integration is displayed to the user and it is equal to 1. This outcome is consistent with the result attained from the manual calculation.

3 Standard Numerical Integration

3.1 Methodology of Standard Numerical Integration

For the standard numerical integration, the selected method is Simpson's rule, or also known as Simpson's 1/3 rule. The trapezoid rule connects two points with a straight line, whereas Simpson's rule utilizes three points for a quadratic fit. (ref. Gezerlis [2020]) Hence, the result obtained is a more accurate approximation of the integral. The integral can be calculated using Lagrange polynomials and the resulting formula is as follows.

$$\begin{aligned}\int_{a=x_0}^{b=x_n} f(x)dx &= \sum_{i=0}^{n-2} \int_{x_i}^{x_{i+2}} P_2(x)dx = \sum_{i=0}^{n-2} \frac{h}{3}(f_i + 4f_{i+1} + f_{i+2}) \\ &= \frac{h}{3}(f_1 + 4f_2 + 2f_3 + ... + 2f_{n-2} + 4f_{n-1} + f_n)\end{aligned} \tag{3}$$

Where a and b represent the bounds of the interval and h the step. In order to execute this method, an even number of segments of equal width is mandatory. If that condition is not met and the number of segments is divisible by 3, then the Simpson's 3/8 rule is employed. (ref. Kiusalaas [2010]) This number can be easily found using the following equation.

$$n = \frac{b-a}{h} \tag{4}$$

The error for Simpson's rule can be calculated through a Taylor series approach. (ref. Gezerlis [2020]) After a few lines of calculation, that are excluded for the sake of focus, we conclude that the error of Simpson's rule is determined by the following formula.

$$\mathcal{E} = \frac{b-a}{180} h^4 f^{(4)}(\xi) \tag{5}$$

Where ξ is within the (a, b) interval. As is readily apparent, the error contains a fourth derivative, signifying that for polynomials up to 3rd degree the Simpson's rule is exact. This also means that the results produced are accurate, even for cubic polynomials, despite being derived from a parabola. (ref. Chapra [2021]) This is promising for our analysis, since the integrand under study involves terms up to the 3rd degree. (ref. Gezerlis [2020]) This error is significantly lower than that of trapezoid, and slightly lower than the Simpsons 3/8 rule. This is the rationale behind the choice to employ Simpson's rule for the computation.

3.2 Code Overview of Standard Numerical Integration

Though the analytical calculation is straightforward, the computation of a double integration with Simpson's rule is not that simple. In this context, in order to achieve high accuracy, the computation of the inner and outer integral has to be simultaneous, rather than sequential. The pseudo-code of the Simpson's rule applied for a double integral, which forms the foundation of the provided code, can be referenced in (ref. Burden [2010]). The methodology of it is that, for a given value of x, the inner and then outer integral is computed, subsequently advancing to the next x value.

The program inputs the predefined bounds and integrand $f(x, y)$ in a function named 'doubleIntSimpson', where the computation is executed. Within this function, the initial step is utilizing 'syms' for the symbolic representation of variables x and y. Afterwards, the n and m parameters are defined as the number of segments for the x and y integration respectively. Both parameters are set to 100 to strike a balance between accuracy and time efficiency. Prior to entering the iterations, a few additional variable are defined. The step h_x, associated with the x values, is determined by solving the equation (4) for h. The variable I_{fl}, I_{even} and I_{odd} represent components of the integral result and are initialized to zero.

The computation occurs concurrently for both integrals, implemented through two nested 'for' loops. The outer loop is configured for i to iterate from 0 to n. Within the loop the x_{val} variable is defined to incrementally take values starting from its lower limit, with a step applied in each iteration. Same as before, a few variable need to be defined before entering the nested 'for' loop. The step h_y is defined by equation (4). The K_{fl} variable incorporates the first and last terms of the Simpson's rule for the inner integral. Variables K_{even} and K_{odd} are initialized to zero.

The nested 'for' loop iterates over j from 1 to $m - 1$, since the first and last terms are already incorporated to K_{fl}. In the loop the y_{val} is defined similarly to x_{val} and the numerical result of $f(x_{val}, y_{val})$ is incorporated to variable Q. Based on the term's index j the value of Q is added to K_{even}, if j is even, or to K_{odd}, if j is odd. This approach allows us to gather terms that share the same factor in the Simpson's rule. With this, the nested loop concludes.

The value of the inner integral for a given x is assigned to variable L, using equation (3) of Simpson's rule. To determine which term of the Simpson's rule L represents, we examine the index i. If i is equal to 0 or n, then the value of L is added to I_{fl}, if it is even, L is added to L_{even} and if it is odd, L is added to I_{odd}. After the completion of the outer loop, the result of the integral is given by equation (3) and is stored in the variable I, which is then returned in the main program.

Upon returning to the main program, the error is calculated by subtracting the result and true value of the integral. Both the result and the error are displayed to the user with a proper message.

3.3 Results of Standard Numerical Integration

When the code is executed, the displayed result is 1, as expected, and the error is calculated as $5 \cdot 10^{-9}$. The error is closely connected to the step through equation (5). If we use this equation to calculate the error for our specified bounds and step, it is revealed that the expected value of the error is significantly lower. Since the function under study is of degree less than four, the derivative can be disregarded. Upon using the equation (5) and ignoring the fourth derivative, the expected error is calculated to be $5.56 \cdot 10^{-11}$. As evident, the computed error is two orders of magnitude larger than the expected error. This discrepancy arises due to the nature of the computation of the double integration, compounded by the oversight of error propagation.

4 Monte Carlo Integration

4.1 Methodology of Monte Carlo Integration

The Monte Carlo method is a prominent numerical technique for assessing multidimensional integrals. It involves choosing random points within a specified boundary region to obtain the weighted data and, from that, determine the estimate value of the integral. (ref. Pang [2006]) This method is also called dartboard Monte Carlo integration, due to its resemblance to the game of darts. For a one-dimensional integral, the function $f(x)$ is confined within a rectangular target, the dimensions of which depend on the integral bounds and the maximum value of the function. Random values of x and y are drawn and they are categorized as below or above the function. The integral estimate is determined by the area multiplied with the ratio of points beneath the function. To enhance accuracy the numbers of points has to increase. (ref. Chapra [2021])

For a multidimensional Monte Carlo integration, the relative error decreases as $1/\sqrt{N}$, due to it being statistical. This holds true, even if the N points are distributed across D dimensions. However, employing these N points for D separate one-dimensional integrals (using methods like Simpson's rule), results in a reduction of N/D points for each integration. This means that as D increases the number of points is decreasing, causing an escalation of error with each integration. The total error is amplified and is roughly N times the error of each integral. By analyzing the information above, it is revealed that when $D \approx 3-4$, the error of the Monte Carlo integration converges to that of conventional methods. However, the Monte Carlo method demonstrates higher accuracy when applied for higher number of dimensions. (ref. Landau et al. [2015])

4.2 Code Overview of Monte Carlo Integration

Prior to entering the 'doubleIntMonteCarlo' function, a variable n is assigned a value of 100, representing the number of times the Monte Carlo simulation is to be executed. To store these results, the $results_c$ vector is initialized with zero values and size $(n, 1)$. Subsequently, a 'for' loop is initiated for n iterations, during which the simulation is run.

Once again, the function accepts as input the integrand $f(x, y)$ and the integral bounds. The first step is to define symbolically x and y. Afterwards, the number of samples is set to 5000 and the variable *count* is initialized to zero. The values of n and *samples* are chosen as is to achieve a better balance of time efficiency and accuracy.

A 'for' loop ranging from 1 to *samples* is initiated, where the variables x and y acquire random values within their specified bounds. Using these randomly generated values, the value of function $f(x, y)$ is calculated and the result is assigned to f_{val}. To generate a random value in the z axis, we need to determine the minimum and maximum values that the $f(x, y)$ function can assume. This occurs when both x and y simultaneously assume their respective minimum and maximum values. Following that, the randomly generated z value is compared to the $f(x, y)$ value., if z_{rand} is lower the count is increased by 1. After being executed for *samples* iterations, the loop is concluded.

Before calculating the integral, it is essential to determine the area. As evident from the 3D plot in Figure 1, the area of the xy-plane is triangular, so it is calculated accordingly. To finally compute the integral, the product of the count, area and the range of $f(x, y)$ is divided by the number of samples. Upon completion of the computation, the outcome is conveyed back to the main program as the output.

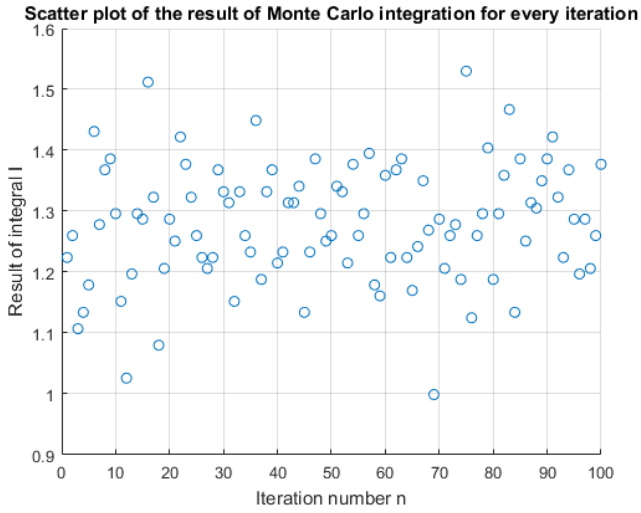


Figure 2: Scatter plot of the integration results of the Monte Carlo integration. In the x-axis is the number of iterations n and in the y-axis is the resulting value of the integration I. The mean value is $I = 1.279890$ and the variance is $\sigma^2 = 0.009403$.

Over the course of n iterations, the results are accumulated in the vector $results_c$. The final result of the Monte Carlo integration is given by the mean value of the results, which is then presented to the user. To calculate the variance of the result, we utilize the MATLAB 'var' function and afterwards, the variance is displayed. Lastly, the program produces a scatter plot displaying all the results from the multiple runs of the simulation.

4.3 Results of Monte Carlo Integration

Due to the substantial number of samples and the iterations of the simulation, this method stands out as the most time-consuming of all. The result of the integral for 100 iterations, is 1.279890. This substantial deviation for the true value of the integral suggests a potential flaw in the computation. The variance of the results given by the simulation is 0.009403. For a clearer insight into these values, the scatter plot of Figure 2 is generated. As evident from both the variance and the scatter plot, the majority of values are close to the mean value of the results, with few exhibiting substantial deviations.

5 Discussion

As is apparent from the outcomes of all the methodologies employed in this assignment, the analytical method is the most accurate. This is anticipated, given the integral under study is simple to solve analytically. The analytical method is easily computable, both in code and manually, in this particular instance. However, this is not the case for the majority of double integrals, that can be more complex. This the rationale behind incorporating methods such as Simpson's rule and Monte Carlo integration in the assignment.

The Simpson's rule demonstrates a notably satisfactory level of accuracy, with a rather low number of segments. Nevertheless, writing the code for Simpson's rule can be somewhat more intricate due to the simultaneous solution of both integrals.

On the other hand, the Monte Carlo integration adheres to a completely different rationale. Although it is computationally straightforward, the are numerous factors that render it unpredictable. For instance, the number of samples needs to be relatively large, and the simulation must be executed multiple times to achieve a reliable result. Hence, this method is more time-consuming. This happens because the method relies on random numbers and is inherently statistical in nature. While the method may seem excessive for this particular two-dimensional integral, it proves to be quite valuable in higher dimensions, particularly beyond the fourth dimension, or in more complex integrands.

Although these methods inherently differ in their approaches, for the solution of this specific double integral, we can conclude that the analytical method is not only the most accurate, but also the most computationally efficient, followed closely by Simpson's rule. The Monte Carlo integration exhibits less accuracy and is ranked last in this context.

6 Conclusion

The integrand under study is a two-variable polynomial, that has a highest degree of 3. The function is continuous everywhere and within the bounds, $y \in [x, 2x]$ and $x \in [0, 1]$, it does not exhibit any symmetry.

For the analytical calculation, the manual calculation is consistent with the cod computation, both yielding a result of 1. As mentioned earlier, for the double integral under study, the analytical method proves to be the most efficient method of calculation.

The standard numerical integration is performed through Simpson's rule, chosen for its optimized error in comparison to other standard methods, such as the trapezoidal rule and the Simpson's 3/8 rule. The result of the integration is 1, with an error of $5 \cdot 10^{-9}$. This value of the error is two orders of magnitude larger than the expected, due to error propagation. The Simpson's rule is deemed satisfactorily efficient for solving the double integral.

Although the Monte Carlo integration is computationally simpler than Simpson's rule, the results exhibit variability, due to its statistical nature. The result of the Monte Carlo integration is 1.279890, with a variance of 0.009403. The excessively large error indicates a potential fault within the computation. The Monte Carlo method is a more suitable tool for more complex integrands and in higher dimensions.

Appendix: Code Listings

```
% Ypologistika Mathhmatika I
% Project 3
% Martha Papadopoulou
% AEM 4438

clear; clc; close all;

syms x y %define variables symbolically
f = x^2 + y^3; % define integrand

%define integral bounds
x_lower = 0;
x_upper = 1;
y_lower = x;
y_upper = 2*x;

%% 3D plot of integrand within bounds
x_values = linspace(x_lower, x_upper, 100);
y_values = linspace(subs(y_lower, x, x_lower), subs(y_upper, x, x_upper), 100);
[X, Y] = meshgrid(double(x_values), double(y_values)); % create a grid of values
mask = (Y >= X) & (Y <= 2*X); % restriction for y

F = double(X.^2 + Y.^3);
F(~mask) = NaN;

figure(1);
surf(X, Y, F); % 3d plot of f(x,y)
colormap(spring); % colour of surface
title('3D Plot of f(x, y) = x^2 + y^3 within integral bounds');
xlabel('x');
ylabel('y');
zlabel('f(x,y)');

%% (a) solve analytically
g = int(f, y, y_lower, y_upper);
result_a = int(g, x, x_lower, x_upper);
result_a = double(result_a);
fprintf('The analytical result of the double integral is: %d\n', result_a);

%% (b) solve numerically with Simpson
result_b = doubleIntSimpson(f, x_lower, x_upper, y_lower, y_upper);
accuracy_b = abs(result_a - result_b);
fprintf('The numerical result of the double integral, using Simpson, is: %f\n',
    result_b);
fprintf('Accuracy of Simpsons rule is %e\n', accuracy_b);

%% (c) solve numerically with Monte Carlo
n = 100; % number of iterations
results_c = zeros(n,1); % initialize results vector

for i = 1:n
    results_c(i) = doubleIntMonteCarlo(f, x_lower, x_upper, y_lower, y_upper);
end
result_c = mean(results_c); % mean value of results
variance_c = var(results_c); % variance of results
fprintf('The numerical result of the double integral, using Monte Carlo integration %
    d times, is %f\n', n, result_c);
fprintf('The variance of the method is %f\n', variance_c);
```

```
57
58 % scatter plot of results for each iteration
59 figure(2);
60 scatter((1:n),results_c);
61 xlabel('Iteration number n');
62 ylabel('Result of integral I');
63 title('Scatter plot of the result of Monte Carlo integration for every iteration');
64 grid on;
```

Listing 1: Main Program

```
1  function I = doubleIntSimpson(f, x_l, x_u, y_l, y_u)
2
3      syms x y; % define symbolically
4
5      % set number of segments
6      n = 100;
7      m = 100;
8
9      h_x = (x_u - x_l)/n; %set step of x
10     I_fl = 0; % sum of first and last term for outer integral
11     I_even = 0; % sum of even terms for outer integral
12     I_odd = 0; % sum of odd terms for outer integral
13
14     % calculate outer integral
15     for i = 0:n
16         x_val = x_l + i*h_x; % value of x with each iteration
17         h_y = (y_u - y_l)/m; % set step of y
18         K_fl = subs(subs(f, y, y_l), x, x_val) + subs(subs(f, y, y_u), x, x_val); %
       sum of first and last term for inner integral
19         K_even = 0; % sum of even terms for inner integral
20         K_odd = 0; % sum of odd terms for inner integral
21
22         % calculate inner integral
23         for j = 1:(m-1)
24             y_val = y_l + j*h_y; % value of y with each iteration
25             Q = subs(subs(f, y, y_val), x, x_val); % f(x_val, y_val(x_val))
26
27             if mod(j, 2) == 0 % check if term is even or odd
28                 K_even = K_even + Q;
29             else
30                 K_odd = K_odd + Q;
31             end
32         end
33
34         L = subs(h_y, x, x_val)/3 * (K_fl + 2*K_even + 4*K_odd); % value of inner
       integral for a value of x
35
36         if i == 0 || i == n % check if term is first or last, even or odd
37             I_fl = I_fl + L;
38         elseif mod(i,2) == 0
39             I_even = I_even + L;
40         else
41             I_odd = I_odd + L;
42         end
43     end
44
45     I = double(h_x/3 * (I_fl + 2*I_even + 4*I_odd)); % value of the double integral
46
```

```
end
```

Listing 2: Simpson's Rule

```
function I = doubleIntMonteCarlo(f, x_l, x_u, y_l, y_u)

    syms x y; % define symbolically

    samples = 1000; % number of random samples
    count = 0; % sum of samples that are above f(x,y)

    for i = 1:samples

        % calculate f for random x, y
        if isnumeric(x_u)
            x_rand = x_l + (x_u - x_l) * rand();
            y_rand = subs(y_l, x, x_rand) + (subs(y_u, x, x_rand) - subs(y_l, x,
    x_rand)) * rand();
        else
            y_rand = y_l + (y_u - y_l) * rand();
            x_rand = subs(x_l, y, y_rand) + (subs(x_u, y, y_rand) - subs(x_l, y,
    y_rand)) * rand();
        end

        % calculate function f value for random x and y
        f_val = double(subs(f, [x, y], [x_rand, y_rand]));

        % calculate maximum and minimum value function f can take
        if isnumeric(x_u)
            f_max = double(subs(subs(f, y, y_u), x, x_u));
            f_min = double(subs(subs(f, y, y_l), x, x_l));
        else
            f_max = double(subs(subs(f, x, x_u), y, y_u));
            f_min = double(subs(subs(f, x, x_l), y, y_l));
        end

        % calculate random value for comparison
        z_rand = f_min + (f_max - f_min) * rand();

        % check if random value is lower than the function value
        if z_rand <= f_val
            count = count + 1;
        end

    end

    % calculate the area
    if isnumeric(x_u)
        area = 1/2 * (x_u - x_l) * (subs(y_u, x, x_u) - subs(y_l, x, x_l));
    else
        area = 1/2 * (y_u - y_l) * (subs(x_u, y, y_u) - subs(x_l, y, y_l));
    end

    %calculate integral
    I = double(count * area * (f_max-f_min)/samples);

end
```

Listing 3: Monte Carlo Integration

References

Douglas Burden, Richard L. , Faires. *Numerical Analysis*. Brooks/Cole Cengage Learning, 9 edition, 2010. ISBN 978-0-538-73351-9.

Raymond P. Chapra, Steven C., Canale. *NUMERICAL METHODS FOR ENGINEERS*. McGraw-Hill Education, 8 edition, 2021. ISBN 978-1-260-23207-3.

Alexandros Gezerlis. *Numerical Methods in Physics with Python*. Cambridge University Press, 1 edition, 2020. doi: 10.1017/9781108772310.

Jaan Kiusalaas. *Numerical Methods in Engineering with Python*. Cambridge University Press, 2 edition, 2010. doi: 10.1017/CBO9780511812224.

R.H. Landau, M.J. Páez, and C.C. Bordeianu. *Computational Physics: Problem Solving with Python*. EBL-Schweitzer. Wiley, 2015. ISBN 9783527413157. URL https://books.google.gr/books?id=gR6zCQAAQBAJ.

Tao Pang. *An Introduction to Computational Physics*. Cambridge University Press, 2 edition, 2006. doi: 10.1017/CBO9780511800870.