Numerical Laboratory
Numerical Integration

Abstract. We study the efficiency of different numerical integration methods. Key methods included are the Trapezoid and Simspson's Rule, Romberg Integration, Splines and Weighted function integrals. We classify them according to how the steps of integration are. We contrast the techniques by comparing their accuracy, computational efficiency and error convergence rates when applying them to given integrals.

The findings indicate that Romberg integration provides the hightest accuracy, but at a higher computational time. On the other hand, Spline methods are better in scenarios where the function behavior satisfies the imposed boundary conditions, while weighted function integrals give the exact result when the integrand function is a polynomial.

Jennifer Fabà-Moreno
Pablo Vega del Castillo
Group 18

Supervisor: Zekang Zhang

May 25, 2025

1 Theoretical framework

The scope of this project is to test the efficiency of some numerical integration methods. They are of the uttermost importance when solving physical problems that involve integrals, since their analytical solution is rarely known. Our problem is to approximate the value of the integral $I(f)$:

$$I(f) = \int_a^b f(x)dx \qquad a, b \text{ are the extrema of the interval.}$$

The approach is based on computing the area under the function in the interval $[a, b]$.

We are going to classify the integration methods according to how the grid points are spaced, having methods with equally spaced steps and methods with unequally spaced steps. One should choose one or the other depending on the available data for the problem.

From the first kind, we are going to study the Newton Cotes rules, which base their approach in the degree of the polynomial approximation used in each sub-interval using Taylor expansion. Therefore, we approximate the integral by evaluating it in different sub-intervals of length:

$$h = \frac{b-a}{n} \qquad x_i = a + ih \quad i = 0, ..., n$$

where n is the number of sub-intervals and $x_0 = a, x_n = b$ [1]. Thus:

$$I(f) \approx \sum_{i=0}^{n} f(x_i)\omega_i$$

where ω_i are the weights for each sub-interval. They depend upon the chosen Newton Cotes rule. In our case, we study what happens when you choose a linear and a quadratic function. We have the Trapezoid and the Simpson's rule, respectively.

The Trapezoid rule is the simplest one of the Newton Cotes family. For each sub-interval, the approximating function is an straight line connecting the 2 extrema, thus, the name of the rule. The error of this rule is of order $O(h^2)$ and the approximated integral follows:

$$I(f) \approx \frac{h}{2}(f(x_0) + f(x_n)) + h\sum_{i=1}^{n-1} f(x_i) \equiv T_n(f) \tag{1.1}$$

On the other hand, the Simpson's rule bases its approximations in the second order Taylor expansion. We have:

$$I(f) \approx \frac{h}{3}(f(x_0) + f(x_n)) + \frac{4h}{3}\sum_{i=1}^{\frac{n}{2}-1} f(x_{2k+1}) + \frac{2h}{3}\sum_{i=1}^{\frac{n}{2}-1} f(x_{2k}) \equiv S_n(f) \tag{1.2}$$

where we have taken into account that the number of intervals must be even for applying this rule, so $n \equiv 0 \pmod 2$.

The final rule that we are going to study is the Romberg Integration, which is expected to increase the Trapezoid accuracy, It considers the approximations done in step i to compute the value of the approximation in step $i+1$. In particular, in Equation 1.3, $j = 1, 2, ..., n+1$ and $i = 0, 1, ..., n-j+1$, so we obtain an upper triangular matrix whose entries are the set of approximations made at each step. The algorithm is:

1. Compute the first column via Trapezoid rule.

[1] When the size of the sub-intervals we choose is infinitesimally small $h \to 0$, we have the definition of the Riemann integral.

2. Use the iterative formula:
$$I_{ij} = \frac{4^{j-1} I_{i+1,j-1} - I_{i,j-1}}{4^{j-1} - 1} \tag{1.3}$$
to obtain the elements of the next columns.

3. The element on the last column is our best estimate for the integral.

We compare the 3 above-mentioned rules in order to choose the most efficient one. In fact, our criteria for the selection will be based both on the error of the approximations and the computational time required for obtaining them.

In addition, the chosen rule is applied to an Astrophysical scenario. We focus on the concept of Strömgren radius, which appears in the study of ionization processes in planetary nebula [1]: a single hot star surrounded by a gas cloud made up only of hydrogen. The star radiates a finite number of photons which are then capable of ionizing a finite region. This means that there has to be some spherical shell which separates the completely ionized region (HII) and the neutral hydrogen (HI). The radial distance from the star to this shell defines the Strömgren radius, and the resulting sphere satisfies the ionization equilibrium condition: the photoionization rate balances the recombination rate of electrons and ions. This is represented in the following relation, in which the integral will be computed numerically:
$$Q(H^0) = \int_{\nu_0}^{\infty} \frac{L_\nu}{h\nu} d\nu = \frac{4\pi}{3} n_H^2 \alpha_B r_S^3 \tag{1.4}$$
where the integral ($Q(H^0)$) outputs the number of ionizing photons per second given the luminosity of the star L_ν and the right hand side gives the number of recombinations per second with n_H being the hydrogen number density, α_B is a recombination coefficient and r_S is the Strömgren radius which is obtained as:
$$r_S \simeq 3.15 \times 10^{15} \left[\frac{Q(H^0)}{n_H^2}\right]^{1/3} T_4^{0.28} \; [pc] \qquad T_4 \text{ related to the local temperature of the medium.} \tag{1.5}$$
On the other hand, we also study unequal steps integration methods. They are more effective when we are working with observational data from which we cannot choose the data points we are studying. Also, if we are dealing with a function whose behavior is volatile in some intervals. In this case, we would choose more points in this kind of intervals, so we do not lose useful information. Here, we focus in splines and weight function integrals.

The spline method considers a polynomial of degree n as interpolating function. In particular, the most used splines consider a polynomial of degree 3 as interpolating function in each interval (cubic splines). Their boundary conditions are:

1. Smoothness conditions: continuity of first and second derivatives.

2. Natural boundary conditions: second derivative of the interpolating functions of the extrema of the original interval set to 0.

We will study the impact of the second condition in the accuracy of the approximations during the development of the project.

In the integration with weight functions method, we have to consider the integrand as a function with a weight, so we must split it up and decide which of the component is acting as a weight. The weight function assigns the level of importance to different parts of the domain, thus reshaping the contribution of each segment to the overall integral. Indeed, we are going to use the following theorem [2] for selecting this weight function:

Theorem 1. *For a suitable, fixed weight function $w(x)$ and the associated integration limits a and b, it is possible to find grid points $x_1, x_2, ..., x_n$ in the interval (a, b) and also weights $w_1, w_2, ..., w_n$ such that the following formula holds exactly for any polynomial $f(x)$ of degree less than or equal to $2n - 1$,*
$$\int_a^b w(x) f(x) dx = w_1 f(x_1) + w_2 f(x_2) + ... + w_n f(x_n) \tag{1.6}$$

2 Tasks

2.1 Exercise 1: Trapezoid Rule

Use a Python program that applies the trapezoid rule to evaluate the integral:

$$\int_0^{\pi/2} \sin(x)dx = 1.0 \tag{2.1}$$

We give you the analytic result so you can check the output of the program.

- **Correct the given Python code for solving the above integral. Record the answers the program gives for the integral, and compute the error and the ratio (the error is simply the difference of the actual result and the analytical value; and the last column gives the ratio of the actual and the previous error value).**

  We take the pseudocode from [3] and correct the bug in the definition of h. We see that the code follows the expression Equation 1.1

```
 1 def f(x):
 2     return_val = math.sin(x)
 3 
 4     return return_val
 5 
 6 def Trap(a, b, n, h):
 7     area = (f(a) + f(b))/2.0
 8     for i in range(1, n):
 9         x = a + i*h
10         area = area + f(x)
11     area = area*h
12     return area
13 
14 a = 0. # lower bound of integration interval
15 b = .5*math.pi # upper bound of integration interval
16 intervals = [2**i for i in range(9)]
17 for i in range(len(intervals)):
18     h = (b-a)/(intervals[i]) #The bug was here!
19     area = Trap(a, b, intervals[i], h)
20     error = 1 - area
21     if i == 0:
22         error0 = error
23     ratio = error0/error
24     print ("With n =", intervals[i], "intervals, our trapezoid estimate area from",
        a, "to", b, "= %.15f" % area, "error: ", error, "ratio:%.2f" % ratio)
25     error0 = error
26 
```

  We also add lines 15-18 to save the error in each iteration as well as the ratio between results of consecutive iterations.

  In Table 2.1, f is the sine function, T_n, n follow the notation of section 1 and $[a, b] = [0, \frac{\pi}{2}]$. As we know the analytic result of Equation 2.1, we see that the value of $T_n(f)$ approaches it with increasing n. We can see this effect either in the second or the third column of Table 2.1.

  We achieve a minimum error of order 10^{-6} for 256 intervals, so this means that the Trapezoid method has needed 257 points equally space of the interval to interpolate the function and compute the integral with this error. Although it seems a lot of 'work', the function has return all the data in Table 2.1 in: 0.998735 ms.

n	$T_n(f)$	Error	$Ratio$
1	0.785398163397448	0.21460184660255172	
2	0.948059448968520	0.0519405510314801	4.13
4	0.987115800972776	0.012884199027224486	4.03
8	0.996785171886170	0.003214828113830448	4.01
16	0.999196680485072	0.0008033195149279582	4.00
32	0.999799194320019	0.00020080567998115306	4.00
64	0.999949800092101	$5.019990789867368e-05$	4.00
128	0.999987450117526	$1.2549882474122143e-05$	4.00
256	0.999996862535288	$3.137464711922e-06$	4.00

Table 2.1: Summary of results for solving Equation 2.1 with Trapezoid rule.

- **Why do we always double n? (Note that the errors are decreasing by a constant factor of 4.)**

  We choose the number of intervals to be of the form 2^i. It means that each time we increase $i \rightarrow i+1$, we are dividing the already defined intervals in halves. This way, if we need to iterate this function a large number of times, we can reuse the value from the previous iterations, reducing the computational time and memory required.
  On the other hand, we note that the ratio of the error is 4, since the result of the integral with this method has an error of order $O(h^2)$ and $h \propto \frac{1}{n}$ so:

$$R = \frac{O\left(\frac{1}{2^i}\right)^2}{O\left(\frac{1}{2^{i+1}}\right)^2} = 4 \tag{2.2}$$

  Thus, by doubling n we obtain a constant rate of 4 and whereas increasing n by other factors, like 3 or 4 might reduce the error further, the rate could not be significant compared to the computational time required. Note that our ultimate goal is to improve the accuracy of the results and doubling n is known to be the most effective way to do so.

2.2 Exercise 2: Simpson's Rule

- **Modify the Python program from the last exercise as described above, and run the new program for the same integral as in exercise 1. Record the answers the program gives for the integral, and compare the results with the values presented in the table below.**

  The main difference in implementation between the Trapezoid rule and Simpson's rule is in the weight each point is assigned. In this case, we have to check whether the index position of the point is even or odd. Even indices will be assigned a 2/3 factor and odd indices a 4/3 factor. Both extremes have a 1/3 factor (refer to Equation 1.2).

```python
def Simp(a, b, n, h):
    """
    Input args:
    a: lower bound of integral
    b: upper bound of integral
    n: number of intervals
    h: stepsize
    """
    area = (f(a) + f(b))/3.0
    for i in range(1, n):
        x = a + i * h
        if i%2 == 0: # chcek if index is even or odd
```

```
            area += 2./3. * f(x)
        else:
            area += 4./3. * f(x)
    area *= h
    return area

# Table
intervals = [2**i for i in range(1, 9)]
for i in range(len(intervals)):
    h = (b-a)/(intervals[i])
    area = Simp(a, b, intervals[i], h)
    error = 1- area
    if i == 0:
        error0 = error
    ratio = error0/error
    print ("With n =", intervals[i], "intervals, our Simpson estimate area from", a
    , "to", b, "= %.15f" % area, "error: ", error, "ratio:%.2f" % ratio)
    error0 = error
```

Listing 1: Simpson's rule implementation.

n	$S_n(f)$	Error	Ratio
2	1.002279877492210	-0.0022798774922103693	
4	1.000134584974194	-0.00013458497419382986	16.94
8	1.000008295523968	-8.295523967749574e-06	16.22
16	1.000000516684707	-5.166847065751767e-07	16.06
32	1.000000032265001	-3.226500089326123e-08	16.01
64	1.000000002016129	-2.0161285974040766e-09	16.00
128	1.000000000126001	-1.2600120946615334e-10	16.00
256	1.000000000007875	-7.874811913666235e-12	16.00

Table 2.2: Summary of results for solving Equation 2.1 with Simpson's rule.

Comparing the results of Table 2.2 with the ones obtained using Trapezoid rule, Table 2.1, we see that we achieve less error for every choice in the number of intervals, n, with Simpson's rule. As we will see in the next question this is expected.

Another difference we can notice is that while the Trapezoid rule underestimates the value of the integral, Simpson's rule does the opposite, it overestimates this value.
This can be explained in terms of the curvature of the integrand function and the polynomial approximation of each of the methods. In Figure 2.1 we can visualize how both the Trapezoid rule and Simpson's rule work. The trapezoid rule, using a linear polynomial, is not the best fit when the slope of the function is curved and misses part of the area. Nevertheless, Simpson's rule approximates above the curve in each of these intervals, overestimating the area.

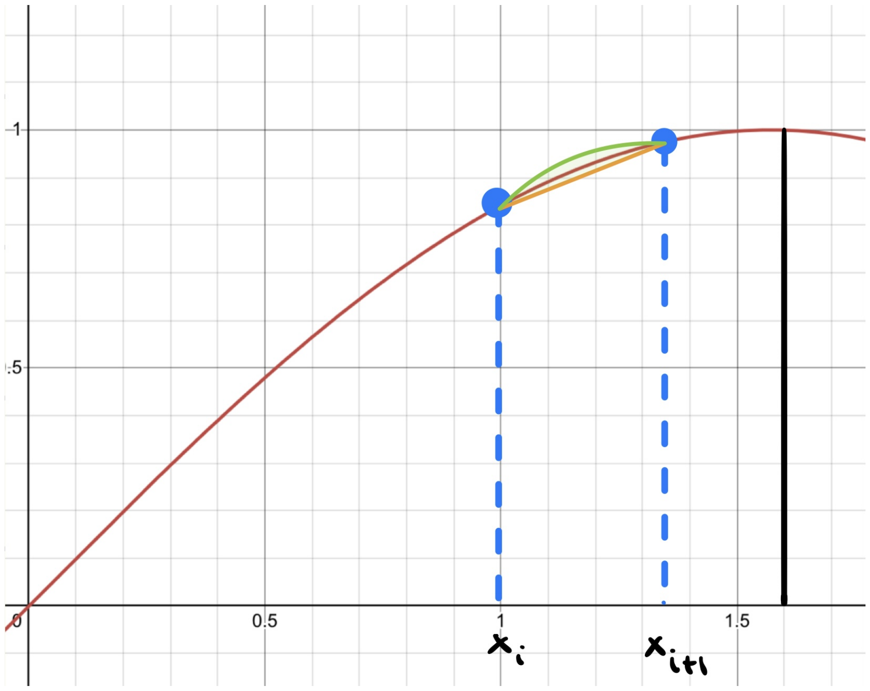

Figure 2.1: Visualization of a single sub-interval of integration for Trapezoid (orange) and Simpson's (green) rules. $\sin(x)$ (red) is the function we want to approximate. The selection of x_i and x_{i+1} is arbitrary and their separation has been exaggerated to better show the difference between both methods.

- **Note that the local error approximation using Simpson's rule is $O(h^5)$ vs $O(h^3)$ using the trapezoidal rule. Why not $O(h^4)$?**

  Rewriting the integrand, $f(x)$, using a Taylor expansion around $x_0 = 0$ we get:

  $$f(x) = f_0 + \frac{f_h - f{-h}}{2h}x + \frac{f_h - 2f_0 + f_{-h}}{2h^2}x^2 + O(x^3)$$

  If we now perform this integral in the interval $[-h, h]$ and focus on the error term:

  $$\int_{-h}^{h} O(x^3)dx = O(x^4)]_{-h}^{h} = O(h^4) - O(h^4) = O(h^5)$$

  where in the last equality we have used the fact that an order $O(h^4)$ zero can be interpreted as something non-zero of order $O(h^5)$. This means that by increasing the degree of the polynomial approximation by one, we have been able to reduce the local error by two orders of magnitude with respect to the trapezoidal rule. This reduction of the error is reflected in Table 2.2 which demonstrates the fact that less error is committed when using Simpson's rule.

- **Using Simpson's rule the ratios of successive errors have converged to 16. Why?**

  To explain this we can follow the same procedure as we did for proving that the ratio in successive errors for the Trapezoid rule was 4. For an integral computed by Simpson's rule, the global error goes like $O(h^4)$ and given that $h \propto \frac{1}{n}$, the ratio between consecutive errors is then:

  $$R = \frac{O\left(\frac{1}{2^i}\right)^4}{O\left(\frac{1}{2^{i+1}}\right)^4} = 2^4 = 16 \tag{2.3}$$

  Simpson's rule not only produces a smaller error for a given number of intervals n with respect to the Trapezoid rule, but it also reduces the error faster when doubling the number of intervals.

- **There is a great deal to be learned by doing numbers of examples. For example, are the ratios of convergence for our numerical examples typical of the trapezoidal and Simpson rules? For the integrals:**

$$I^{(1)} = \int_0^1 e^{-x^2} dx = 0.746824132812427 \tag{2.4}$$

$$I^{(2)} = \int_0^4 \frac{dx}{1+x^2} = \arctan(4) = 1.32581766366803 \tag{2.5}$$

calculate T_2,..., T_{2048} and S_2, ..., S_{2048} and write the error, the relative error and the ratio to a file - we get back to this file in exercise 3.

For computing these integrals we define the integrand functions and adapt our Trapezoid and Simpson's rules so that they also accept an integrand function as input.

```python
def gauss(x):
    return np.exp(-x**2)

def frac(x):
    return 1. / (1. + x**2)

def Trap(func, a, b, n, h):
    area = (func(a) + func(b))/2.0
    for i in range(1, n):
        x = a + i*h
        area += func(x)
    area *= h
    return area

def Simp(func, a, b, n, h):
    area = (func(a) + func(b))/3.0
    for i in range(1, n):
        x = a + i * h
        if i%2 == 0:
            area += 2./3. * func(x)
        else:
            area += 4./3. * func(x)
    area *= h
    return area
```

Listing 2: Functions needed for computing the integrals $I^{(1)}$ and $I^{(2)}$.

n	$T_n(f)$	Error	Rel. error	Ratio
2	0.731370251828563	0.015453880983864021	0.020692798083086893	
4	0.742984097800381	0.0038400350120457727	0.005141819664537058	4.0244
8	0.745865614845695	0.0009585179667318533	0.001283458748343093	4.0062
16	0.746584596788222	0.00023953602420545556	0.00032073953382223888	4.0016
32	0.746764254652294	5.9878160132975644e-05	8.017705575137688e-05	4.0004
64	0.746809163637828	1.496917459897773e-05	2.0043774620145277e-05	4.0001
128	0.746820390541618	3.7422708090151247e-06	5.01091307122374e-06	4.0000
256	0.746823197246153	9.355662744514603e-07	1.2527263559738743e-06	4.0000
512	0.746823898920948	2.3389147918440045e-07	3.131814692484567e-07	4.0000
1024	0.746824074339563	5.847286410620711e-08	7.82953596933285e-08	4.0000
2048	0.746824118194211	1.4618215860018324e-08	1.9573839700343277e-08	4.0000

Table 2.3: Summary of results for solving Equation 2.4 with Trapezoid rule.

n	$S_n(f)$	Error	Rel. error	Ratio
2	0.747180428909510	-0.00035629609708331955	0.00047708166009788973	
4	0.746855379790987	-3.124697856016212e-05	4.1839808312688976e-05	11.4026
8	0.746826120527467	-1.9877150395641863e-06	2.661557055044206e-06	15.7200
16	0.746824257435730	-1.2462330334361837e-07	1.6687101804584947e-07	15.9498
32	0.746824140606985	-7.794557888018971e-09	1.0436938960000988e-08	15.9885
64	0.746824133299672	-4.872452441517794e-10	6.524230039499224e-10	15.9972
128	0.746824132842881	-3.045408369928282e-11	4.0778119454438805e-11	15.9993
256	0.746824132814331	-1.9038104426272184e-12	2.549208520428706e-12	15.9964
512	0.746824132812545	-1.1801670751765414e-13	1.580247642416442e-13	16.1317
1024	0.746824132812435	-7.549516567451064e-15	1.0108827816022393e-14	15.6324
2048	0.746824132812428	-6.661338147750939e-16	8.919553955313876e-16	11.3333

Table 2.4: Summary of results for solving Equation 2.4 with Simpson's rule.

n	$T_n(f)$	Error	Rel. error	Ratio
2	1.458823529411765	-0.1330058657437323	0.1003198776034977	
4	1.329411764705882	-0.0035941010378497396	0.0027108562031872697	37.0067
8	1.325253402497078	0.0005642611709542056	0.0004255948509488928	6.3696
16	1.325673581732914	0.00014408193511883383	0.00010867401986500466	3.9163
32	1.325781625681882	3.60379861501503e-05	2.7181706155918092e-05	3.9981
64	1.325808653076049	9.010591983660277e-06	6.79625278089251e-06	3.9995
128	1.325815410951537	2.252716495343421e-06	1.6991148610216976e-06	3.9999
256	1.325817100484627	5.631834054664608e-07	4.247819446818553e-07	4.0000
512	1.325817522871914	1.407961187638307e-07	1.0619568785522246e-07	4.0000
1024	1.325817628468988	3.519904412385699e-08	2.6548932849841993e-08	4.0000
2048	1.325817654868275	8.799757145183662e-09	6.6372302816045495e-09	4.0000

Table 2.5: Summary of results for solving Equation 2.5 with Trapezoidal rule.

n	$S_n(f)$	Error	Rel. error	Ratio
2	1.239215686274510	0.08660197739352271	0.06531967386369557	
4	1.286274509803921	0.039543153864111114	0.02982548426358287	2.1901
8	1.323867281760810	0.0019503819072226314	0.001471078535661282	20.2746
16	1.325813641478192	4.022189840524604e-06	3.0337428371535918e-06	484.9055
32	1.325817640331539	2.333649384844705e-08	1.760158616674623e-08	172.3562
64	1.325817662207437	1.4605954223867457e-09	1.1016563305891055e-09	15.9774
128	1.325817663576701	9.133205303157865e-11	6.888734064599626e-11	15.9921
256	1.325817663662324	5.70810065880778e-12	4.305343649605361e-12	16.0004
512	1.325817663667675	3.5771385853422544e-13	2.698062247447869e-13	15.9572
1024	1.325817663668011	2.1094237467877974e-14	1.5910360863286625e-14	16.9579
2048	1.325817663668029	3.774758283725532e-15	2.8471172071144487e-15	5.5882

Table 2.6: Summary of results for solving Equation 2.5 with Simpson's rule.

For both integrals performed with the Trapezoid rule the ratios end up converging to the expected value of 4. We can mention that in Table 2.5, going from $n = 2$ to $n = 4$ results in the error decreasing by a ratio of 37.0067 which is higher than expected. This can be due to the fact that with the first subdivision, $n = 2$, the method performs much worse than when $n = 4$ resulting in a significantly higher error ratio.
Two points can be made regarding the results obtained using Simpson's rule. Firstly, when $n = 2048$

we see how the ratios deviate from the expected value of 16. For Equation 2.4 and $n = 2048$ the ratio is reduced to 11.3333 while for Equation 2.5 and $n = 2048$ the ratio is reduced to 5.5882. As stated in [3], when n exceeds 1000, rounding errors become significant. This is reflected in the ratios as the error decreases more slowly compared to previous values of n. The second point comes from Table 2.6 where the ratios of 484.9055 and 172.3562 appear between $n = 8$ and $n = 16$, and between $n = 16$ and $n = 32$, respectively. Both large ratios may have a similar explanation as the given for the Trapezoid rule case.

2.3 Exercise 3: Romberg Integration

- **Verify the example presented in the section above – i.e. $\int_0^1 ue^{2u}du = 2.0972640247327$, and write the calculated matrix to a file.**

```python
def Romberg(func, a, b, N): #func is the function where we apply the Romberg integration
    matrix_I = np.zeros((N+1, N+1))

    for i in range(N+1):
        h = (b-a) / 2**i
        matrix_I[i, 0] = Trap(func, a, b, 2**i, h)

    for j in range(1, N+2):
        for i in range(N-j+1):
            matrix_I[i,j] = (4 ** (j) * matrix_I[i+1, j-1] - matrix_I[i,j-1]) / (4 ** (j) - 1)

    return matrix_I

def exp2(x):
    return x * np.exp(2*x)

Romberg(exp2, 0, 1, 5)[0,5]
-> output
2.0972640247326986
```

Listing 3: Romberg Algorithm.

We define the function `Romberg` and compute the numerical value of the integral:

$$\int_0^1 ue^{2u}du = 2.0972640247327 \tag{2.6}$$

We see that with $N = 5$, that is 2^5 intervals, this method has been able to provide the exact result of Equation 2.6.

- **For the integrals of Equation 2.4 and Equation 2.5 calculate $I(0,12)$ - how many intervals do we have for the elements $I(0,2 \to 12)$, respectively? - and write the error, the relative error and the ratio for the elements $I(0,2 \to 12)$ to a file. This program has now to be compared with the corresponding one of exercise 2. Give an answer to the question "Which Rule is Better?". Make a plot of the relative accuracy for integrating the first integral of ?? versus the number of intervals for all methods discussed up to now.**

```python
gauss_romb = np.zeros(11)
frac_romb = np.zeros(11)

for i in range(11):
    gauss_romb[i] = Romberg(gauss, 0, 1, i)[0, i]
    frac_romb[i] = Romberg(frac, 0, 4, i)[0, i]

#we compute the relative error for each integral
```

```
rel_error_romb_gauss = np.abs(0.746824132812427 - gauss_romb) / 0.746824132812427
rel_error_romb_frac = np.abs(np.arctan(4)- frac_romb) / np.arctan(4)
```

Listing 4: Romberg integration for $I^{(1)}$ and $I^{(2)}$.

With the code above we compute the value of the integrals of exercise 2 to compare the 3 methods presented in this work up to now. We want to find which of them is better, that is, which is the one that takes the least computing time as well as provides the most accurate results.

We compute the Romberg rule from $I(0, 2 \to 12)$ to compare with the Simpson's and Trapezoid's values. Thus, we have 2^i number of intervals for $I(0, i+1)$ where $i = 1, ..., 11$.

n	$R_n(f)$	Error	Rel. error	Ratio
2	0.683939720585721	6.28844122e-02	8.42024373e-02	
4	0.747180428909510	-3.56296097e-04	4.77081660e-04	-176.49481075292258
8	0.746833709849752	-9.57703733e-06	1.28236849e-05	37.203164713384
16	0.746824018482281	1.14330146e-07	1.53088446e-07	-83.76651037572837
32	0.746824133095094	-2.82667223e-10	3.78492353e-10	-404.4690586165182
64	0.746824132812243	1.83519866e-13	2.45733711e-13	-1540.2540834845736
128	0.746824132812427	-3.33066907e-16	4.45977698e-16	-551.0
256	0.746824132812427	-1.11022302e-16	1.48659233e-16	3.0
512	0.746824132812427	-3.33066907e-16	4.45977698e-16	0.3333333333333333
1024	0.746824132812427	-4.44089210e-16	5.94636930e-16	0.75
2048	0.746824132812427	-1.11022302e-16	1.48659233e-16	4.0

Table 2.7: Summary of results for solving Equation 2.4 with Romberg Integration.

n	$R_n(f)$	Error	Rel. error	Ratio
2	2.117647058823529	-7.91829395e-01	5.97238532e-01	
4	1.239215686274510	8.66019774e-02	6.53196739e-02	-9.143317727693395
8	1.289411764705882	3.64058990e-02	2.74592050e-02	2.3787896978882177
16	1.326960160238128	-1.14249657e-03	8.61729785e-04	-31.865215104429133
32	1.325932558249954	-1.14894582e-04	8.66594141e-05	9.943868117985316
64	1.325815327337979	2.33633005e-06	1.76218052e-06	-49.177376176800585
128	1.325817669393643	-5.72561021e-09	4.31855025e-09	-408.04909323157625
256	1.325817663678667	-1.06348264e-11	8.02133404e-12	538.3830462469987
512	1.325817663668014	1.82076576e-14	1.37331536e-14	-584.0853658536586
1024	.325817663668033	-2.22044605e-16	1.67477483e-16	-82.0
2048	1.325817663668036	-3.55271368e-15	2.67963972e-15	0.0625

Table 2.8: Summary of results for solving Equation 2.5 with Romberg Integration.

Now that we have all the data, we can compare the relative errors of the 3 methods. We do so by plotting them together. We choose to represent the errors in logarithmic scale so that we can compare them better (see Figure A.1 for the original plots). We see that in both cases the Romberg integration provides more accurate results. However, we see both in Table 2.8 and in Figure 2.2 that for Equation 2.5, the last step provides a worse result than the previous one (although it is still lower than the one provided by the Trapezoid and the Simpson's rules).

Regarding the computational time required for each, we use the `time` module from Python.

```
area_trap = np.zeros(11)
area_Simp = np.zeros(11)
gauss_romb = np.zeros(11)
```

```
frac_romb = np.zeros(11)


start_time = time.time()
for i in range(len(intervals)):
    h = (1)/(intervals[i])
    #Chose only one method/integral at the same time
    # area_trap[i] = Trap(frac, 0, 1, intervals[i], h)
    #area_Simp[i] = Simp(frac, 0, 1, intervals[i], h)
    # gauss_romb[i] = Romberg(gauss, 0, 1, i)[0, i]
    #frac_romb[i] = Romberg(frac, 0, 4, i)[0, i]
end_time = time.time()

elapsed_time = end_time - start_time
elapsed_time
```

Listing 5: Code for obtaining the computational time required for each integration method.

Method	Integral	Computational time (ms)
Trapezoid		9.000778198242188
Simpson	$I^{(1)}$	2.0003318786621094
Romberg		7.020711898803711
Trapezoid		2.000093460083008
Simpson	$I^{(2)}$	3.3681392669677734
Romberg		1.9829273223876953

Table 2.9: Computational time for different integration methods.

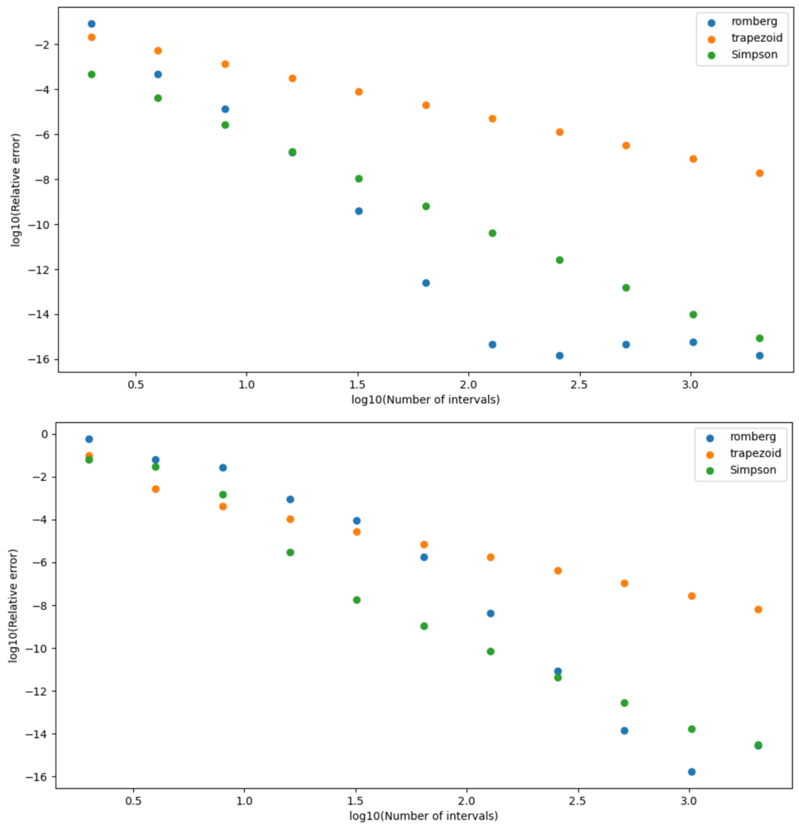

Figure 2.2: Relative errors for Trapezoid, Simpson and Romberg Rules for the integrals: upper $(I^{(1)})$, lower $(I^{(2)})$. Logarithmic scale.

In Table 2.9, we see that the computational time does not differ much between the 3 methods. However, we must take into account that each loop consists on only 11 iterations (i.e, we are studying 11 different subdivisions of the interval). In real experiments, we might need to consider more iterations in order to obtain an accurate numerical result, so the time we are getting now might be multiplied by some large factor. Thus, although the Romberg provides lower error in the results, we might choose the Simpson method only for integral $I^{(1)}$, since the error doesn't differ much with the one provided by Romberg (at least for the largest value of n we are considering). Let us quantify them. By looking at Table 2.4 and Table 2.7, we have:

$$e_S \approx 6.e-16 \qquad e_R \approx 1.e-16 \tag{2.7}$$

The errors differ by less than an order of magnitude so the reduction in computational time provided by the Simpson's method might overcome Romberg's accuracy. Overall, we notice that the Romberg method gains significantly accuracy when increasing the number of intervals.

This is just an analysis of this particular case, however, the answer for the proposed question "Which Rule is Better?", considering our problem and our discussion, might be the Romberg rule. The computational time between Simpson and Romberg rules do not differ significantly.

This answer might depend on the problem. The one conclusion that we can firmly state is that the Romberg is a better rule than the Trapezoid.

2.4 Exercise 4: Strömgren Theory

- **Integrate numerically Q(H^0) with the rule you have chosen to be the best one in Exercise 3 –note that the integral doesn't have an analytical result; thus, have a look at the example given in Sect. 3.5. Modus operandi:**
  Add a function which computes
  $$\frac{L_\nu}{h\nu}$$
  where $L\nu$ and $B_\nu(T_*)$ are described by Eqs.(33), and (34), respectively. The parameters required are:
  stellar radius : $R_*/R_\odot = 9.0$
  stellar effective temperature : $T_* = 40000K$
  and the constants required are given in Table 1 of Sect. 3.5. In order to figure out what ∞ means for Q(H^0), plot the function, choose an upper limit for the integral, and choose a step size – or test some candidates for it.
  Perform the integral.

In order to be able to calculate numerically Q(H^0), we need to integrate the expression $L_\nu/h\nu$. To do this we first need to compute the Planck function and then, from it, the luminosity L_ν. The necessary methods are listed below in the Python code snippet.

```
#constants
h = 6.6260755e-27 #erg s
c = 2.99792458e10 #cm/s
k = 1.380658e-16 #erg/K
Rsun = 6.9599e10 #cm
R = 9. * Rsun #cm
T = 40000.0 #K
nu0 = 1.0973731569e05 * c

def f_Planck(nu): # Planck function
    return 2 * h * nu **3 / c**2 * 1. / (np.exp(h * nu / k / T) - 1.)

def lum(nu): # Luminosity
```

```
    return 4 * math.pi**2 * R**2 * f_Planck(nu)

def integrand(nu):
    return lum(nu) / h / nu
```

Listing 6: Necessary constants and funtions for computing Q(H^0).

Once the integrand $L_\nu/h\nu$ is computed, we can plot it to find a suitable upper limit of integration. We already know that the lower limit corresponds to the frequency of the Lyman edge, $\nu_0 = 1.0973731569e05 \times c \approx 3.29e15$ s^{-1}, which is given by the ionization energy of the ground state of hydrogen. In Figure 2.3 we have plotted the integrand as a function of the frequency ν and we have used logarithmic scale in both axis. From this plot we can notice how fast does the integrand decrease with frequency. When the frequency increases by one order of magnitude, the integrand decreases several orders of magnitude. This means that the upper limit of the integral does not need to be much grater than the lower limit, ν_0. In our case we have chosen the upper limit to be $100\nu_0$ as it was not computationally expensive and it guarantees that the integral is computed properly. This results in the following numerical estimation for Q(H^0):

$$Q(H^0) = \int_{\nu_0}^{100\nu_0} \frac{L_\nu}{h\nu} d\nu = 9.892462206208877 \times 10^{48}\ s^{-1} \tag{2.8}$$

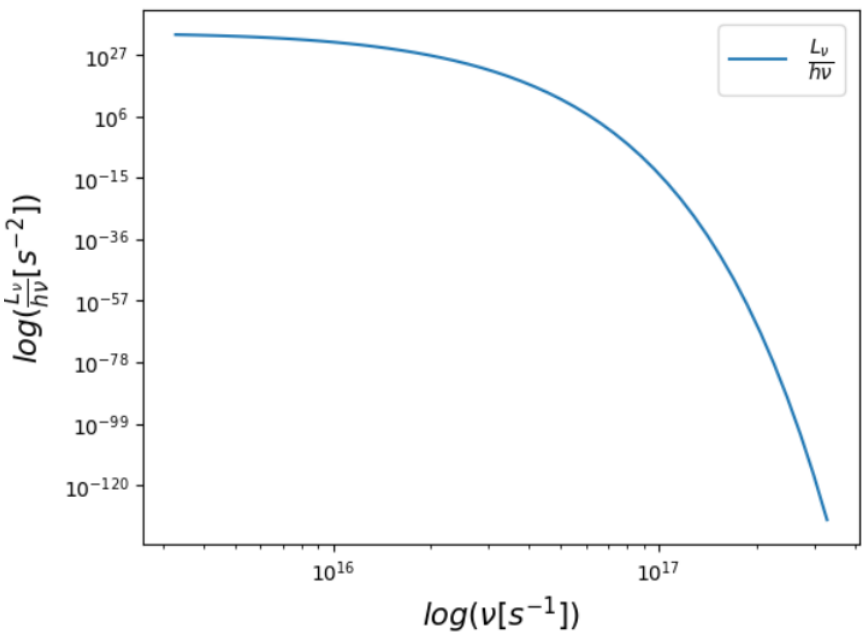

Figure 2.3: $L_\nu/h\nu$ vs. ν.

- **Calculate the Strömgren radius. The parameters required are:**
  hydrogen particle density of the nebula: $n_H = 10\ cm^{-3}$
  electron temperature of the nebula: T_e, T_4 = 7500, 0.75 K, none

  Once Q(H^0) is obtained we can compute the Strömgren radius using Equation 1.5 with $n_H = 10\ cm^{-3}$ and $T_4 = 0.75$, which results in $r_S \simeq 13.441$ pc.
  To get a sense of the magnitude of this value, the typical diameter of a planetary nebula is on the order of one light year [4], which is significantly smaller than our estimate for the Strömgren radius. This can be explained if we look at Equation 1.5 which depends inversely on the hydrogen number density. The fact that we are using a rather small value of this parameter, as typical values for n_H are in the range of 10^2 to 10^4, can make our value of the Strömgren radius bigger than expected.

In addition, we also need to take into account the characteristics of the central star of the nebula (the stellar radius was $9 \times R_\odot$ and the stellar surface temperature was 40000 K), since Q(H^0) grows with the luminosity of the star. The latter quantity grows with the radius and the temperature (as we are assuming that the emitted radiation follows Planck's function). In this case, this means that central star is highly luminous, for example, more luminous compared to the Sun. In particular, by integrating the luminosity over the frequencies we obtain that it is of order $1e+38$ erg/s, whereas the Sun's is $1e+33$ erg/s.

2.5 Exercise 5a: Splines

- **Compute the same integral as in exercise 1 and 2 (Equation 2.1) and prepare a table which can be compared with those obtained from exercise 1 and 2.**

For obtaining the numerical approximations with splines we use the already defined function `Splines` and `I_Spline` and add these lines of code:

```
intervals = [2**i for i in range(1,9)]
a_spline = np.zeros(len(intervals))
for i in range(len(intervals)):
    a_spline[i] = I_Spline(a, b, intervals[i]+1)
    print ( "%.15f" % a_spline[i])
# error_spline = [np.abs(a_spline[i]-1) for i in range(len(intervals))]
# error_spline

```

Listing 7: Spline call.

n	Method	$I_{\pi/2}$	Error
2	Trapezoid	0.948059448968520	0.0519405510314801
	Simpson	1.002279877492210	-0.0022798774922103693
	Splines	0.988724770361288	-1.12752296e-02
4	Trapezoid	0.987115800972776	0.012884199027224486
	Simpson	1.000134584974194	-0.00013458497419382986
	Splines	0.998504611458065	-1.49538854e-03
8	Trapezoid	0.996785171886170	0.003214828113830448
	Simpson	1.000008295523968	-8.295523967749574e-06
	Splines	0.999815227695472	-1.84772305e-04
16	Trapezoid	0.999196680485072	0.0008033195149279582
	Simpson	1.000000516684707	-5.166847065751767e-07
	Splines	0.999977089331897	-2.29106681e-05
32	Trapezoid	0.999799194320019	0.00020080567998115306
	Simpson	1.000000032265001	-3.226500089326123e-08
	Splines	0.999997145993769	-2.85400623e-06
64	Trapezoid	0.999949800092101	5.019990789867368e-05
	Simpson	1.000000002016129	-2.0161285974040766e-09
	Splines	0.999999643807545	-3.56192455e-07
128	Trapezoid	0.999987450117526	1.2549882474122143e-05
	Simpson	1.000000000126001	-1.2600120946615334e-10
	Splines	0.999999955509129	-4.44908713e-08
256	Trapezoid	0.999996862535288	3.137464711922e-06
	Simpson	1.000000000007875	-7.874811913666235e-12
	Splines	0.999999994440662	-5.55933755e-09

Table 2.10: Results of Equation 2.1 for different integration methods.

- **Run the new code as well as Simpson for the integral $\int_0^\pi \sin(x)dx$, which has a different upper bound, namely π, and prepare a similar table for each method.**

  Now, we want to compute the integral:

$$\int_0^\pi \sin(x)dx = 2 \tag{2.9}$$

n	Method	I_π	Error
2	Simpson	1.8137993642342176	-0.18620063576578239
	Splines	1.963495408493621	-0.03650459150637908
4	Simpson	1.933765598092805	-0.06623440190719498
	Splines	1.9986934197714494	-0.0013065802285505 69
8	Simpson	1.9796508112164832	-0.02034918878351677
	Splines	1.9999302380897719	-6.976191022811662e-05
16	Simpson	1.9943049443094645	-0.005695055690535522
	Splines	1.9999958141683214	-4.185831678604757e-06
32	Simpson	1.998489272187602	-0.0015107278123980272
	Splines	1.9999997410648092	-2.589351908444115e-07
64	Simpson	1.9996106513341514	-0.00038934866584861005
	Splines	1.9999999838582445	-1.6141755487808496e-08
128	Simpson	1.9999011507525397	-9.884924746028645e-05
	Splines	1.9999999989917916	-1.0082084056506346e-09
256	Simpson	1.9999750951844306	-2.4904815569382066e-05
	Splines	1.9999999999369973	-6.300271415682346e-11

Table 2.11: Results of Equation 2.9 for different integration methods.

- **Compare the tables obtained for both integrals with Simpson's rule and the Spline method. Regarding the accuracy: why is the Spline method better in one case and worse in the other?**

  We study the results of the integral Equation 2.1 by comparing the values in the 4th column of Table 2.10. We see that splines are better than the Trapezoid rule. However, the Simpson method is more accurate than splines and this is the relevant result here (we already expected that Simpson's rule overcomes the Trapezoid, by definition). Before finding a reasoning for this, we are going to compare the result for Equation 2.9.
  From Table 2.11, we see that the error of the spline method is some orders lower than the one for Simpson, for every number of intervals considered.

  So by comparing both results, we see that we can't assure that the Spline method is a better approach for numerical integration than Simpson's rule. It might seem so since we are using cubic polynomials to the interpolation instead of a quadratic function. However, the shapes of the functions we are working with, that is, $\sin(x)$ from 0 to $\pi/2$ and from 0 to π, impact the final prediction.

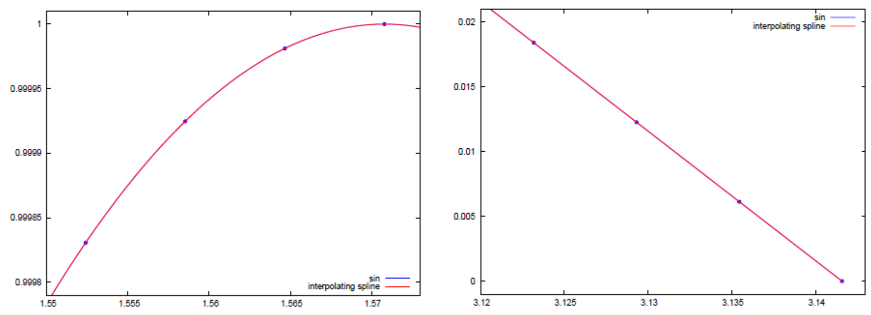

Figure 2.4: $\sin(x)$ and spline interpolation with $n = 512$. Source: [3]

From Figure 2.4, we see the behavior of the sin function on the extrema ($\pi/2$ and π). On the first case, we see that in a neighborhood of $\pi/2$, the function is non-linear, so the second derivative of it will be non-zero when evaluated in the extreme (in fact, it is -1). Thus, the boundary conditions that are used for determining the spline parameters, which are chosen to be the natural boundary conditions, are not satisfied. Therefore, it is expected that splines are not going to provide the most accurate results for this function. This is confirmed by the discussion on the previous paragraph, in which we noticed that the Simpson's rule performed better.
On the other case, we see that for π, the function is almost linear so its second derivative is consistent with the assumptions that it is 0 (it can also be argued by the fact that $-\sin(\pi) = 0$, which corresponds to the second derivative of f). Thus, splines are an effective approach for extrapolating its values, better than the Simpson's one since it considers a quadratic formula. Again, this is what we see in the error values obtained.

2.6 Exercise 5b: Weight functions

- **Compile and run the new code for the same integrals as in exercise 5a and prepare a table which can be compared with those obtained from exercise 5a.**

n	Method	$I_{\pi/2}$	Error
2	Trapezoid	0.948059448968520	0.0519405510314801
	Simpson	1.002279877492210	-0.0022798774922103693
	Splines	0.988724770361288	-1.12752296e-02
	Weight	0.9999999999999999	-1.1102230246251565e-16
4	Trapezoid	0.987115800972776	0.012884199027224486
	Simpson	1.000134584974194	-0.00013458497419382986
	Splines	0.998504611458065	-1.49538854e-03
	Weight	1	0
8	Trapezoid	0.996785171886170	0.003214828113830448
	Simpson	1.000008295523968	-8.295523967749574e-06
	Splines	0.999815227695472	-1.84772305e-04
	Weight	1	0
16	Trapezoid	0.999196680485072	0.0008033195149279582
	Simpson	1.000000516684707	-5.166847065751767e-07
	Splines	0.999977089331897	-2.29106681e-05
	Weight	1	0
32	Trapezoid	0.999799194320019	0.00020080567998115306
	Simpson	1.000000032265001	-3.226500089326123e-08
	Splines	0.999997145993769	-2.85400623e-06
	Weight	1	0
64	Trapezoid	0.999949800092101	5.019990789867368e-05
	Simpson	1.000000002016129	-2.0161285974040766e-09
	Splines	0.999999643807545	-3.56192455e-07
	Weight	0.9999999999999997	-3.3306690738754696e-16
128	Trapezoid	0.999987450117526	1.2549882474122143e-05
	Simpson	1.000000000126001	-1.2600120946615334e-10
	Splines	0.999999955509129	-4.44908713e-08
	Weight	0.9999999999999998	-2.220446049250313e-16
256	Trapezoid	0.999996862535288	3.137464711922e-06
	Simpson	1.000000000007875	-7.874811913666235e-12
	Splines	0.999999994440662	-5.55933755e-09
	Weight	1.0000000000000002	2.220446049250313e-16

Table 2.12: Results of Equation 2.1 for different integration methods.

- **Compare the tables obtained for both integrals with the Spline method. Discuss the results briefly regarding the accuracy of both methods with increasing number of intervals!**

  From Table 2.12 we can see how the weight function method gives the best result for almost all cases regardless of the number of intervals. This is expected since choosing sin(x) as the weight function results in Equation 1.6 obeying the equality.

  It is easy to see that the weight function method clearly outperforms the rest of the previous studied methods when the right choice of weight function is made.

- **Extra credit: Modify the new program in order to solve the integral $\int_0^1 xe^x dx$.**

For solving this integral, we choose the x term as the integrand and the e^x term as the weight function. In order to do this we compute the weights associated with the exponential function.

```python
 def w_exp(xl,xr,dx,xm):
     G = np.ones(len(dx)) * 1 / dx * (np.exp(xr)-np.exp(xl))
     H = np.ones(len(dx)) * 1 / dx * (np.exp(xr)+np.exp(xl) + 2 / dx * (np.exp(
xl)-np.exp(xr)))

 return G,H

```

Listing 8: Weigths associated with the exponential function.

In the following table, we compare the result of this integral when choosing the e^x term as weight function versus choosing the x term as weight function.

n	Weight	I	Error
2	e^x	1	
	x	1.020155698604	0.020155698603999905
4	e^x	1	
	x	1.0051657100063087	0.005165710006308721
8	e^x	1	
	x	1.0012994151062076	0.0012994151062075687
16	e^x	1	
	x	1.0003253540023045	0.00032535400230448985
32	e^x	1	
	x	1.000081369780349	8.136978034900544e-05
64	e^x	1	
	x	1.0000203444003182	2.0344400318172973e-05
128	e^x	1	
	x	1.0000050862222853	5.086222285344277e-06
256	e^x	1	
	x	1.000001271563209	1.2715632089488338e-06

Table 2.13: Comparison of results for different choices of weight function for solving the integral $\int_0^1 xe^x dx$.

As it can be seen, choosing e^x as the weight function always results in the exact value for the integral, i.e., 1. This goes in accordance to what was seen in 1 as when the integrand is chosen to be a polynomial Equation 1.6 is satisfied exactly. On the other hand selecting x as the weight function produces less accurate results.

3 Conclusions

This project main goal was to achieve a sense on how numerical integration is used when analytical integration is not possible, what different methods can be used and in which cases each method is better.

Among equal step methods we have digged into the Newton Cotes rules which comprises of the Trapezoid rule, Simpson's rule and an improvement of the Trapezoid rule which is Romberg's method.
Between the first two methods we were able to demonstrate how Simspon's rule produces a better outcome for the different number of intervals, n, used. It also reduces the error in a faster way, by a factor of 4 compared to the Trapezoid rule, when doubling n. Nevertheless, when exceeding 1000 subdivisions of the interval of integration, rounding errors become appreciable in Simpson's method. We conclude then that Simpson's rule, by using a higher degree polynomial approximation, achieves better accuracy when integrating smooth and well behaved functions.

Regarding the Romberg method, and comparing the relative errors of the three mentioned methods, Figure 2.2, we were able to conclude that, indeed, Romberg's achieves greater accuracy than the simple Trapezoid rule. However, discerning between Simpson's rule and Romberg's is not that easy as for the case of integral $I^{(2)}$, Simpson's rule was able to produce less error as n was reduced but the situation was reversed and Romberg achieved a smaller error for larger n. Given that Romberg is an iterative method which bases its next approximation in the previous ones, one should also take into account the computational cost of this method with respect to Simpson's.

Having studied the equally spaced methods we then applied them, in this case Romberg's, to the computation of the Strömgren radius for a given set of parameters of a planetary nebula. We tested the Romberg method in a real application by computing numerically the number of ionizing photons produced by the central star of the nebula. The result of this integral then allowed us to obtain the value of the Strömgren radius and analyze it in terms of the given parameters of the nebula and the central star.

In relation with unequally spaced steps methods, we first studied the splines interpolation. We found out that the method has a big dependence on the boundary conditions. In particular, for cubic splines, the second derivative condition (natural boundary conditions) was of the uttermost importance on the accuracy of the predictions.
Regarding the weight function integrals, we have assessed the impact of selecting the weight function in the approximations. We have found that taking a polynomial as the integrand, when possible, provides us with the exact result.

Our final conclusion is that the numerical integration rule chosen depends on the problem you are dealing with. One has to understand the data involved in the problem and the properties of the function used on the integration. The decision also depends on the required level of accuracy as well as in the available computational resources.

4 Author's note

We are aware of the maximum extension of the report (12 text pages). Hence when considering only the written part of this report (not taking into account neither codes nor figures/tables), it is about 10 pages.

On the other hand, we have decided to express the numerical results with all the decimals since we are focused on how accurate the predictions are. Thus, all significant figures are relevant.

All the formulae are from the manual provided for the laboratory session, except otherwise stated.

Bibliography

[1] Donald E. Osterbrock. *Astrophysics of gaseous nebulae and active galactic nuclei.* 1989 (cit. on p. 3).

[2] N.J. Giordano and H. Nakanishi. *Computational Physics.* Pearson/Prentice Hall, 2006. ISBN: 9780131469907 (cit. on p. 3).

[3] A.W.A. Pauldrach. *Manual.* https://www.dropbox.com/scl/fo/bpp42k8xqz5ayc4y5lpth/ALNufAk569wOkH4SkiAo9X8?rlkey=q0aojj2hdny4o49t7mac9iczf&st=buzson1b&dl=0. Accessed: Nov, 2024 (cit. on pp. 4, 10, 17).

[4] Inc. Wikimedia Foundation. *Planetary nebula.* https://en.wikipedia.org/wiki/Planetary_nebula. Accessed: Dec, 2024 (cit. on p. 14).

A Additional plots

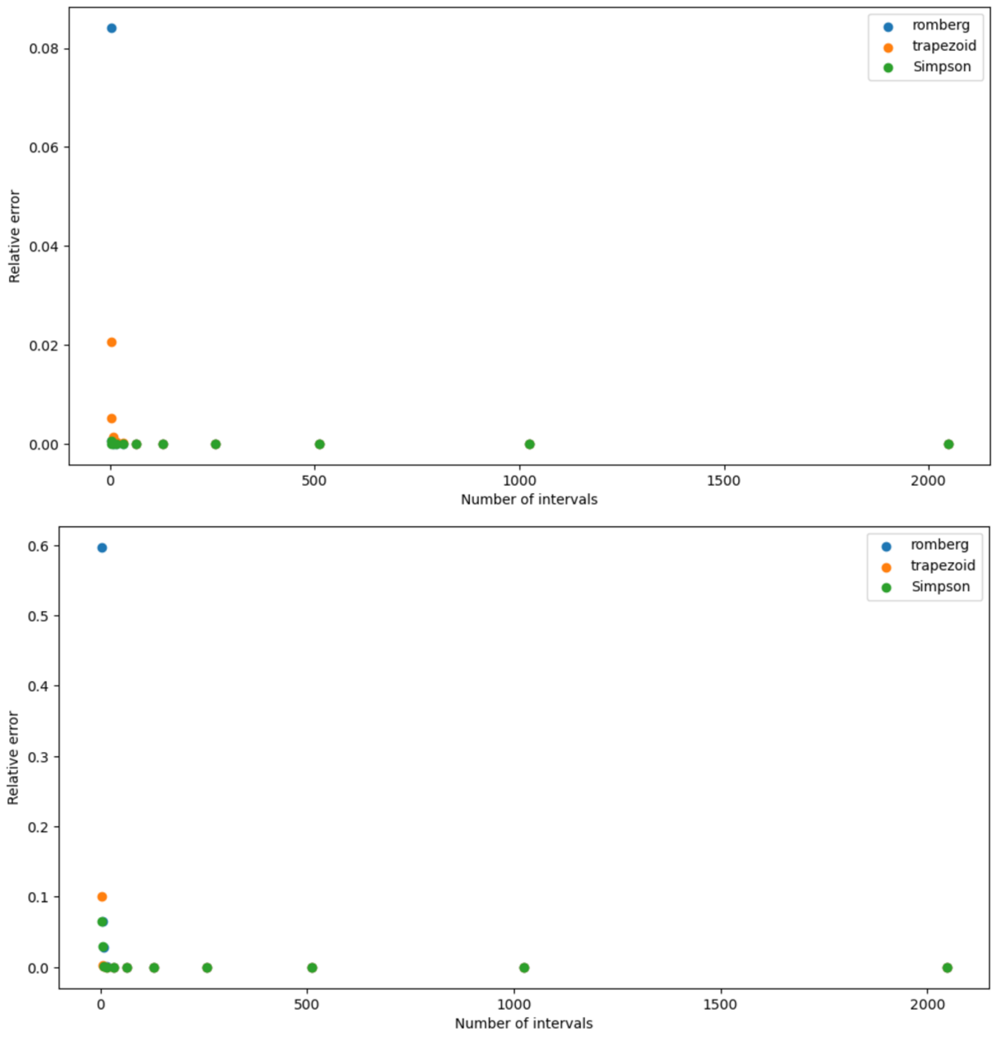

Figure A.1: Relative errors for Trapezoid, Simpson and Romberg Rules: upper ($I^{(1)}$), lower ($I^{(2)}$).