# Example Report on
# Lesson 4 - Numerical Integration

Almaz Khabibrakhmanov*, Mario Galante†, Alexandre Tkatchenko

*Theoretical Chemical Physics group, FSTM,*
*Campus Limpertsberg, University of Luxembourg*

## Part 1 - Midpoint and Trapezoidal Rules

As a first task, I used the `midpoint.py` script from Moodle to calculate the integral of $f(x) = sin(x)$ over the interval $[0, \pi]$. I used $n = 5$ integration points and obtained the value $S_{mid} = 2.0523443060$. Analytical value of the integral can be easily calculated as $S_{an} = 2$. Therefore, the error of integration in this case is $\Delta = S_{mid} - S_{an} = 0.0523443060$.

Next, I did the same for the number of grid points from the array $\mathtt{N} = [5, 10, 20, 30, 50, 100]$. The obtained results are presented in Figure 1. As one can see, the integral is overestimated for any $n$. I observe gradual convergence of $S_{mid}$ to its analytical value. For example, when $n = 100$ the numerical result is correct up to the 4th digit after a comma: $S^{100}_{mid} =$ **2.0000**839191. However, the corresponding integration step $h = \pi/99 \approx 0.0317$ required to achieve this accuracy is fairly small.

```
Example 1: f(x) = sin(x)
Midpoint rule results
n           S                Error
5       2.0523443060    5.23443e-02
10      2.0101901160    1.01901e-02
20      2.0022801201    2.28012e-03
30      2.0009782980    9.78298e-04
50      2.0003425929    3.42593e-04
100     2.0000839191    8.39191e-05
============================================
```

```
Example 1: f(x) = sin(x)
Trapezoidal rule results
n           S                Error
5       1.8961188979    -1.03881e-01
10      1.9796508112    -2.03492e-02
20      1.9954413183    -4.55868e-03
30      1.9980436910    -1.95631e-03
50      1.9993148493    -6.85151e-04
100     1.9998321639    -1.67836e-04
============================================
```

Figure 1: The results for the different number of grid points $n$ for the midpoint (left) and trapezoidal (right) integration rules.

Next, I implemented the trapezoidal rule by myself and repeated the same calculations as described for the midpoint method above (please find the code in a separate '.py' file). The results are presented in Figure 1. In this case, I also observe convergence, however, the integral is now underestimated. To make the comparison of the two methods more illustrative, I plotted the integration error $\Delta$ vs. the integration step $h$ (Figure 2, left). It can be seen that both midpoint and trapezoidal rules have similar scaling of the error with $h$ as they should have theoretically. However, as I noted above, the difference between them is in the sign of the error.

Then, I have repeated the same calculations for a new function $g(x) = 3x^2$ over the interval $[0, 2]$ using both midpoint and trapezoidal rules. The results retrieved are also presented in Figure 2 (right). Again, we see similar scaling of errors for both methods, but trapezoidal rule behaves slightly worse for larger $h$. Comparing to the plot for $f(x) = sin(x)$, I can conclude that the sign of the integration error depends on the particular function that we are integrating. This is true for both midpoint and trapezoidal rules.

*almaz.khabibrakhmanov@uni.lu
†mario.galante@uni.lu

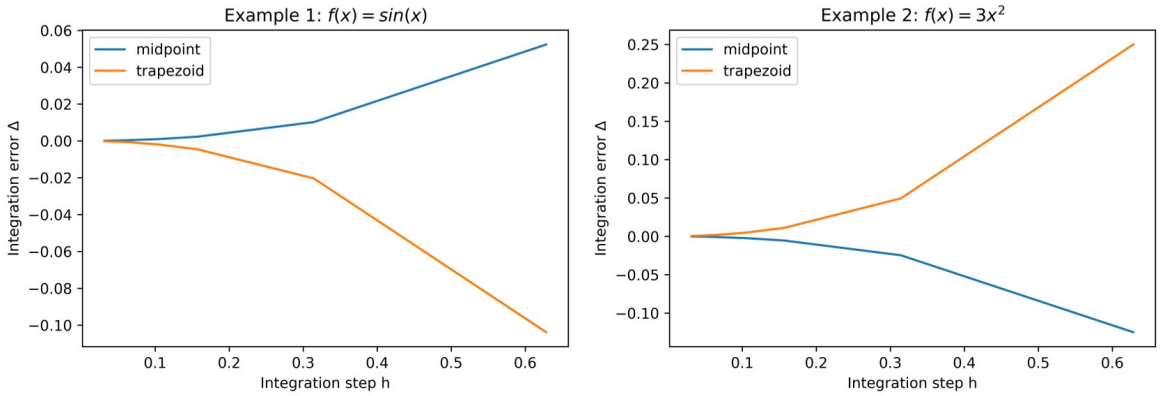


Figure 2: The errors of the midpoint and trapezoidal integration rules plotted against the integration step $h$.

## Part 2 - Let's calculate something more interesting

For this part, I implemented the algorithm `Integrator` given in the task (please find it in the separate '.py' file). Then, using this fucntion I calculated the integral of the function $f(x) = \sqrt{1-x^2}$ over $[0, 1]$ interval with the relative tolerance of $\varepsilon = 10^{-6}$. Figure 3 shows the output of the program.

```
Integrator is in work...
------------------------------------------------------------
j           S           Abs.Error       Rel.Error
1       0.7959823052    7.95982e-01     1.00000e+00
2       0.7885635703    7.41873e-03     9.40791e-03
3       0.7864341999    2.12937e-03     2.70763e-03
4       0.7857511029    6.83097e-04     8.69356e-04
5       0.7855206920    2.30411e-04     2.93322e-04
6       0.7854410943    7.95978e-05     1.01342e-04
7       0.7854132737    2.78206e-05     3.54216e-05
8       0.7854034937    9.77995e-06     1.24521e-05
9       0.7854000458    3.44787e-06     4.38996e-06
10      0.7853988286    1.21727e-06     1.54987e-06
11      0.7853983985    4.30063e-07     5.47573e-07
------------------------------------------------------------
The integral is successfully calculated
Your result for the integral: S = 0.785398398505008
------------------------------------------------------------
Final number to calculate: S*4 = 3.141593594020032
```

Figure 3: The output of the `Integrator` program.

Clearly, the calculated number is one of the fundamental mathematical constants, $\pi = 3.14159265359....$ If we compare this value to the one calculated numerically, we indeed see that the difference is of the order $10^{-6}$.

It is fairly easy to understand why we have obtained $\pi$ when calculating the above stated integral. Indeed, the graph of the function $f(x) = \sqrt{1-x^2}$ on the interval $[0, 1]$ is nothing but a *quarter-circle* of a unity radius (see Figure 4). Therefore, recalling that integral is the area under the function's graph and the school formula for the circle area ($S = \pi r^2$), one can immediately understand that the analytical value for our integral is simply $\pi/4$.

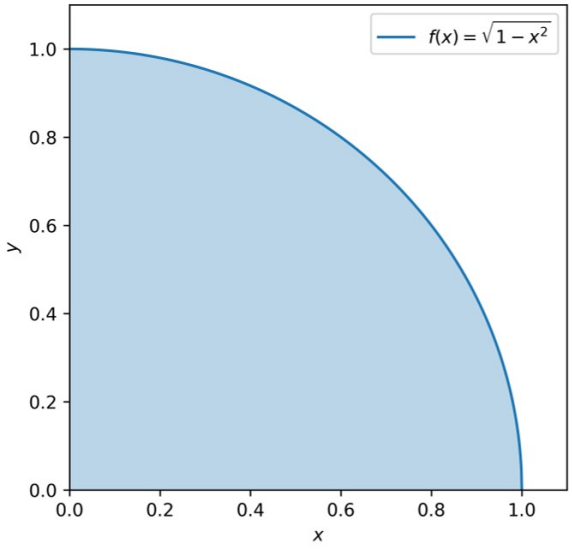


Figure 4: The graph of the function $f(x) = \sqrt{1 - x^2}$ on the interval $[0, 1]$.

However, if I set the relative tolerance to $10^{-20}$, it already takes significant time from a computer to calculate the integral. This is not surprising, since at every iteration number of grid points doubles. At the last iteration, there are already more than 2.5 million points (see Figure 5). Moreover, the algorithm fails to converge to this accuracy (which is pretty high) within 20 iterations specified in the code. Thus, I can conclude that the accurate evaluation of $\pi$ by means of numerical integration of some function is extremely inefficient.

```
Integrator is in work...
============================================================
j       n              S              Abs.Error       Rel.Error
1       5          0.7959823052    7.95982e-01     1.00000e+00
2       10         0.7885635703    7.41873e-03     9.40791e-03
3       20         0.7864341999    2.12937e-03     2.70763e-03
4       40         0.7857511029    6.83097e-04     8.69356e-04
5       80         0.7855206920    2.30411e-04     2.93322e-04
6       160        0.7854410943    7.95978e-05     1.01342e-04
7       320        0.7854132737    2.78206e-05     3.54216e-05
8       640        0.7854034937    9.77995e-06     1.24521e-05
9       1280       0.7854000458    3.44787e-06     4.38996e-06
10      2560       0.7853988286    1.21727e-06     1.54987e-06
11      5120       0.7853983985    4.30063e-07     5.47573e-07
12      10240      0.7853982465    1.51996e-07     1.93527e-07
13      20480      0.7853981928    5.37292e-08     6.84101e-08
14      40960      0.7853981738    1.89944e-08     2.41845e-08
15      81920      0.7853981671    6.71526e-09     8.55013e-09
16      163840     0.7853981647    2.37414e-09     3.02285e-09
17      327680     0.7853981639    8.39369e-10     1.06872e-09
18      655360     0.7853981636    2.96754e-10     3.77839e-10
19      1310720    0.7853981635    1.04886e-10     1.33545e-10
20      2621440    0.7853981634    3.70695e-11     4.71983e-11
============================================================
Failed to converge to desired accuracy within 20 iterations
Try to reduce the tolerance
The value after the last iteration: S = 0.7853981634178132
============================================================
Final number to calculate: S*4 = 3.141592653671253
```

Figure 5: The output of the `Integrator` program when the relative tolerance is set to $10^{-20}$.

## Part 3 - Simpson's Rule

I implemented the Simpson's rule (please find the code in the separate '.py' file) and repeated the tasks 4) - 8) from the Part 1. The results for Simpson's rule together with its comparison to midpoint and trapezoidal methods are shown in Figure 6.

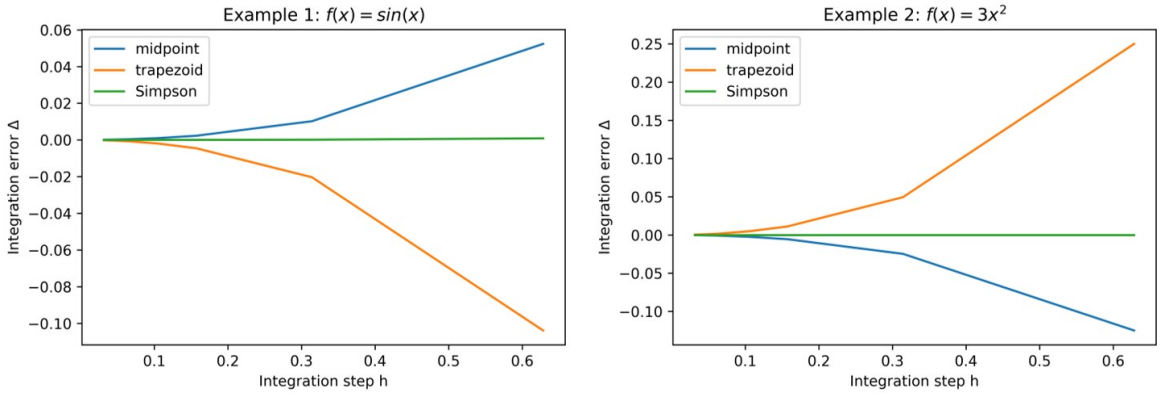


Figure 6: The comparison of the Simpson's rule with the previously studied methods.

If we take closer look at the case of $g(x) = 3x^2$ on the interval $[0, 2]$, we notice that for Simpson's rule the error in this case is equal to zero, *i.e.* it gives the exact value of the integral for any number of grid points. This is true even for minimally possible three grid points (see Figure 7, left)).

If we try to integrate $f(x) = x^3$ over $[0, 2]$, we see the same picture: the error is equal to zero. However, already in the case of $f(x) = x^4$ this effect vanishes, and we observe a normal picture of numerical error converging to zero but never reaching it (Figure 7, right).

```
Example 2: f(x) = 3*x**2
Simpsons rule results
n           S               Error
3       8.0000000000    0.00000e+00
10      8.0000000000    0.00000e+00
20      8.0000000000    0.00000e+00
30      8.0000000000    1.77636e-15
50      8.0000000000    1.77636e-15
100     8.0000000000    0.00000e+00
==========================================
```

```
Example 3: f(x) = x**3
Simpsons rule results
n           S               Error
3       4.0000000000    0.00000e+00
10      4.0000000000    8.88178e-16
20      4.0000000000    8.88178e-16
30      4.0000000000    0.00000e+00
50      4.0000000000    0.00000e+00
100     4.0000000000    -4.44089e-16
==========================================
```

```
Example 4: f(x) = x**4
Simpsons rule results
n           S               Error
3       6.4166666667    1.66667e-02
10      6.4004266667    4.26667e-04
20      6.4000266667    2.66667e-05
30      6.4000052675    5.26749e-06
50      6.4000006827    6.82667e-07
100     6.4000000427    4.26667e-08
==========================================
```

Figure 7: The comparison of the Simpson's rule with the previously studied methods.

Based on these empirical observations, I can assume that Simpson's method is exact by construction for the polynomials of the order $p \leq 3$, however, this does not hold true for the polynomials of higher degrees.