Computational Techniques and Numerical Calculations

Master Studies in Theoretical Chemistry and Computational Modelling

Numerical integration, root finding and function optimization

Evaluation exercises

Author:
José Antonio Quiñonero Gris

March 10, 2023

Part I

Numerical integration

To calculate the integral

$$I = \int_1^3 \sin\left(x^2\right) - \cos\left(2x\right) \mathrm{d}x\,, \tag{1}$$

the following numerical integrations methods have been used:

1 Composite Simpson rule

The commposite Simpson rule (CSR) is one of the Newton-Cotes integration rules, an interpolation method based on equidistant interpolation bascissas in a $[a, b]$ interval

$$I_{\text{rule}} = \int_a^b P_{n-1}\left(x\right) \mathrm{d}x = \sum_{i=1}^{n} f\left(x_i\right) \underbrace{\int_a^b \prod_{j=1, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} \mathrm{d}x}_{\omega_i}, \tag{2}$$

where n is the number of interpolating points (abscissa points). For each value of n, there is a named rule. For the Simpson rule, $n = 3$.

Also, for each one of these methods, there are two rules: simple rule and composite rule.

The simple rule considers the whole interval, $\mathrm{d}x = b - a = h$ or $\mathrm{d}x = (b - a)/2 = h$ for the Simpson method.

For the composite rule, a number of subintervals, N, is used to evaluate the integral. Defining the subinterval spacing, h, as

$$h = \frac{b - a}{2N}, \tag{3}$$

the abscissa points, x_i, where the function is evaluated are given by

$$x_i = a + ih, \qquad i = 0, \ldots, 2N \tag{4}$$

Then, the integral is evaluated as

$$I_{\text{CS}} = \frac{h}{3}\left[f\left(a\right) + f\left(b\right) + 2\sum_{i=2\,\text{(even)}}^{2N-2} f\left(x_i\right) + 4\sum_{i=1\,\text{(odd)}}^{2N-1} f\left(x_i\right)\right], \tag{5}$$

with an error of the order $\mathcal{O}\left(h^5\right)$

$$E_{\text{CS}} = -N \times \frac{f^4\left(\xi\right)}{90} \times h^5 = \mathcal{O}\left(h^5\right), \qquad \xi \in \left[a, b\right]. \tag{6}$$

To compute the integral in eq. (1), the calculation starts with $N = 1$ subinterval points, doubling in each iteration until reaching a convergence (difference between a result and the preceding one) of 10^{-8}.

Then, the basic structure of the algorithm is the following (algorithm 1).

Algorithm 1 Composite Simpson rule

```
procedure SimpsonCompositeNCM($a$, $b$, $N_0$, $\varepsilon$, $I$, $t$)
    Input. Integration limits, $a$, $b$ ; starting number of subinterval points, $N_0$ ; threshold for convergence, $\varepsilon$
    Output. Value of the integral, $I$ ; total number of iterations, $t$
    Initialize the number of subintervals, $N = N_0$, value of the integral, $I = 0$, and number of iterations $t = 0$
    while $\Delta I < \varepsilon$ do
        Calculate subinterval spacing, $h$, and abscissa points, $x_i$
        Calculate the new value of the integral, $I'$
        Calculate the convergence, $\Delta I = |I' - I|$
        Update value of the integral, $I = I'$
        Update number of iterations, $t = t + 1$
```

Table 1: Results from the composite Simpson rule

Iteration, t	Subinterval number, N	Integral value, I_{CS}	Difference, ΔI
1	1	0.09897684	0.09897684
2	2	1.05135757	0.95238073
3	4	1.06287028	0.01151271
4	8	1.05794291	0.00492737
5	16	1.05766834	0.00027457
6	32	1.05765178	0.00001656
7	64	1.05765076	0.00000103
8	128	1.05765069	0.00000006

The results are collected in table 1.

Summarizing, the results are

- Number of iterations needed: 9
- Final subinterval value: 0.00390625
- Number of subintervals: 256
- Number of abscissa points: 257
- Final value of the quadrature: 1.05765069

2 Romberg's method

The Romberg integration is an extrapolation method based in the terative Richardson extrapolation applied on the Composite Trapezoidal Rule $n = 2$ function.

The Richardson extrapolation method aims to speed up the convergence of a sequence, based on the extrapolation of two function values calculated at $\lim_{h\to 0} R\left(h\right)$ and $\lim_{h\to 0} R\left(\frac{h}{2}\right)$, eliminating the errors of the form $E\left(h\right) = Ch^n$. A function, $G\left(h\right)$, is approximated by $R\left(h_1\right)$ (where $h_1 = x_a - x_0$) and by $R\left(h_2\right)$ (where $h_2 = \frac{h_1}{2} = x_b - x_0$), resulting in less rounding errors and/or less number of calculations

$$G\left(h\right)=\frac{2^{n}R\left(h/2\right)-R\left(h\right)}{2^{n}-1}. \tag{7}$$

The Romberg method provides two mechanisms to improve the accuracy: reduce the value of the subinterval spacing, h, and apply Richardson extrapolation.

Then, the value of the integral is given by

$$I_{CT}=\int_{a}^{b}f\left(x\right)\mathrm{d}x\approx\frac{h_k}{2}\left[f\left(a\right)+f\left(b\right)+2\sum_{i=1}^{N-1}f\left(x_i\right)\right], \tag{8}$$

where the spacing $h=h_k$ is now reduced as

$$h_k=\frac{b-a}{N}=\frac{b-a}{2^{k-1}},\qquad k=1,2,3,\ldots \tag{9}$$

Richardson extrapolation is applied $k-1$ times for each k value to the previously computed approximation, so both mechanisms are applied simultaneously. Basically, k is the number of different values of h computed initially related to the number of trapezoids (subintervals).

Then, the Romberg matrix, $\mathbf{R}$, is computed as

$$R_{11}=\frac{h_1}{2}\left[f\left(a\right)+f\left(b\right)\right], \tag{10}$$

$$R_{k,1}=\frac{1}{2}\left[R_{k-1,1}+h_{k-1}\sum_{i=1}^{2^{k-2}}f\left(a+\left(2i-1\right)h_k\right)\right],\quad k>1=j, \tag{11}$$

$$R_{k,j}=R_{k,j-1}+\frac{R_{k,j-1}-R_{k-1,j-1}}{4^{j-1}-1},\quad k\geq j>1, \tag{12}$$

or

$$\mathbf{R}=\begin{pmatrix} R_{11} & & & & \\ R_{21} & R_{22} & & & \\ R_{31} & R_{32} & R_{33} & & \\ R_{41} & R_{42} & R_{43} & R_{44} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \\ \uparrow & \uparrow & \uparrow & \uparrow & \\ \mathcal{O}\left(h^2\right) & \mathcal{O}\left(h^4\right) & \mathcal{O}\left(h^6\right) & \mathcal{O}\left(h^8\right) & \cdots \end{pmatrix}. \tag{13}$$

The algorithm is the following (algorithm 2).

To compute the integral, $n=10$ is chosen so the 10×10 Romberg matrix is computed, although the convergence criteria of $\varepsilon=10^{-8}$ is followed to store both the converged value and its position in the matrix.

The resulting 10×10 Romberg matrix is printed in table 2, and the converged value is

- $R_{7,5}=1.05765069$
- Number of iterations: 25

Algorithm 2 Romberg method

```
procedure RombergIA($a$, $b$, $\varepsilon$, $n$, $\mathbf{R}$, $I$, $r_1$, $r_2$, $t$ )
    Input. Integration limits, $a$, $b$; threshold for convergence, $\varepsilon$; dimensions of the Romberg matrix, $n$
    Output. Romberg matrix, $\mathbf{R}$; Converged value of the integral, $I$ ; position of the converged element, $R(r_1, r_2)$; number of iterations until convergence, $t$
    Initialize Romberg matrix, $\mathbf{R} = 0$; number of iterations, $t' = 1$; dummy variable $d = 0$
    Compute the first element, $R_{1,1}$
    for $i \leftarrow 2, n$ do
        Compute $R_{i,1}$
        for $j \leftarrow 2, i$ do
            Compute $R_{i,j}$
            Calculate the difference with the previous value, $\Delta R = |R_{i,j} - R_{i,j-1}|$
            if $\Delta R < \varepsilon$ and $d = 0$ then
                Store the converged value, $I = R_{i,j}$
                Store the indexes of the converged value, $r_1 = i$ and $r_2 = j$
                Store number of iterations until convergence, $t = t'$
                Update a dummy variable to not reassign $r_1$ and $r_2$ after convergence, $d = 1$
            Update number of iterations, $t' = t' + 1$
```

Table 2: Resulting 10 × 10 Romberg matrix.

R	1	2	3	4	5	6	7	8	9	10
1	0.70956602									
2	0.25162414	0.09897684								
3	0.85142421	1.05135757	1.11484962							
4	1.01000877	1.06287028	1.06363780	1.06282491						
5	1.04595938	1.05794291	1.05761442	1.05751881	1.05749800					
6	1.05474110	1.05766834	1.05765004	1.05765060	1.05765112	1.05765127				
7	1.05692411	1.05765178	1.05765068	1.05765069	1.05765069	1.05765069	1.05765069			
8	1.05746909	1.05765076	1.05765069	1.05765069	1.05765069	1.05765069	1.05765069	1.05765069		
9	1.05760529	1.05765069	1.05765069	1.05765069	1.05765069	1.05765069	1.05765069	1.05765069	1.05765069	
10	1.05763934	1.05765069	1.05765069	1.05765069	1.05765069	1.05765069	1.05765069	1.05765069	1.05765069	1.05765069

3 Gauss-Legendre method

The Gaussian Quadrature is an interpolation method that, unlike the Newton-Cotes methods, picks optimal abscissa points, x_i, at which the function is evaluated, $f(x_i)$, unequally spaced.

The integration rule is exact for polynomials of degree up to $2n-1$, where n are the appropriately chosen abscissa points. Therefore, the maximum degree of accuracy is $2n-1$. Also, not only the abscissa points can be chosen, but also the weights ω_i.

The integral is evaluated as

$$\int_a^b f(x)\,\mathrm{d}x \approx \int_a^b P_{2n-1}(x)\,\mathrm{d}x = \sum_{i=1}^{n} \omega_i f(x_i), \qquad i = 1, 2, \ldots, n \tag{14}$$

The abscissa points can be found for any particular case based in the Fundamental theorem of Gaussian Quadrature [1].

Theorem 1 (Fundamental theorem of Gaussian Quadrature) *The abscissas of the n-point Gaussian quadrature formulas with weighting function $W(x)$ in the interval (a, b) are precisely the roots of the orthogonal polynomial $P_n(x)$ for the same interval and weighting function.*

The solution relates to the orthogonal polynomials generated by the weight function $W(x)$

$$\int_a^b f(x)\,\mathrm{d}x \approx \int_c^d W(t)\,P_{2n-1}(t)\,\mathrm{d}t = \sum_{i=1}^{n} \omega_i P(t_i)\,. \tag{15}$$

where t_i are the roots of the orthogonal polynomial P_{2n-1} and are not equally spaced. Therefore, $P_{2n-1}(t)$ is optimized for the specific range $t \in [c, d]$, and only the roots t_i (abscissa points) and weights, ω_i, have to be determined.

In this case, the Gauss-Legendre polynomials are used. The weight functions for these polynomials is, simply

$$W(x) = 1, \tag{16}$$

the interval is

$$-1 < x < 1, \tag{17}$$

and the recurrence relation, needed to compute the polynomials

$$(k+1)\,P_{k+1}(x) = (2k+1)\,xP_k(x) - kP_{k-1}(x)\,. \tag{18}$$

Algorithm 3 Gauss-Legendre method

```
procedure GaussQuadrature($a$, $b$, $N_0$, $N_{\text{tot}}$, $\varepsilon$, $I$, $t$)
    Input. Integration limits, $a$, $b$; starting number of quadrature points, $N_0$; total number of quadrature points, $N_{\text{tot}}$; threshold for convergence, $\varepsilon$
    Output. Value of the integral, $I$; total number of iterations, $t$
    Initialize the value of the integral, $I = 0$
    Calculate $c = (a+b)/2$ and $m = (b-a)/2$
    for $n \leftarrow N_0, N_{\text{tot}}$ do
        Compute the weights, $\omega_i = (\boldsymbol{\omega})_i$, and $t_i = (\mathbf{t})_i$ values with GauLeg($i, \mathbf{t}, \boldsymbol{\omega}$)
        Compute the integral, $I' = m\sum_{i=1}^{n} \omega_i f(c + mt_i)$
        Calculate the difference with the previous value, $\Delta I = |I' - I|$
        if $\Delta I < \varepsilon$ and $\Delta I > 0$ then exit
        Update the value of the integral, $I = I'$
        Update total number of iterations, $t = n$
```

The results are collected in table 3, and

- Number of quadrature points employed: 10
- Final value of the quadrature: 1.05765069

4 Results

The results from the three methods can be summarized as in table 4 and fig. 1.

From the table 4, the fastest method to reach convergence is the Composite Simpson rule, followed by the Romberg's method (15.3% slower than Simpson's) and, lastly, the Gauss-Legendre (437.2% slower than Simpson's).

Table 3: Results from the Gauss-Legendre method.

Quadrature points	Integral values	Difference, $\Delta I = \|I_{n+1} - I_n\|$
2	1.77932651	1.77932651
3	1.02706235	0.75226416
4	1.03224500	0.00518265
5	1.06169102	0.02944602
6	1.05749649	0.00419453
7	1.05763955	0.00014306
8	1.05765184	0.00001229
9	1.05765067	0.00000118
10	1.05765069	0.00000002

Table 4: Results from the Composite Simpson rule, Romberg's method and Gauss-Legendre method.

Method	Total number of iterations	Execution time (ms)
Composite Simpson rule	9	1.83
Romberg's method	25	2.11
Gauss-Legendre method	10	9.83

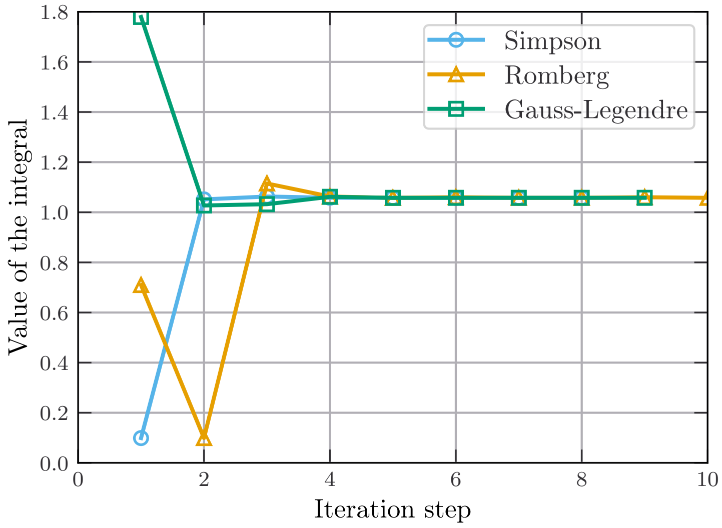

Figure 1: Integral value with increasing iterations for each of the methods used.

Part II

Root finding and function optimization

References

1. Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP Numerical Recipes in Fortran 77, 1986.