Zadanie numeryczne NUM8

1 Wstęp

Zadanie polegało na napisaniu dwóch procedur bibliotecznych liczących całkę z zadanej funckji f na przedziale [a, b] z tolerancją bezwzględną ϵ. Napisane procedury powinny bazować na złożonej kwadraturze Newtona-Cotesa z $n = 2$ oraz na metodzie Romberga. Następnie należało sprawdzić ich poprawność obliczając całkę $\int_0^1 sin(x)dx$

2 Rozwiązanie

2.1 Kwadratura Newtona-Cotesa

Dana jest kwadratura postaci:

$$S(f) = \sum_{i=0}^{n} A_i f(x_i)$$

Z polecenia wiadomo, że $n = 2$. Współczynnik kwadratury Newtona-Cotesa wynosi:

$$A_i = h\frac{(-1)^{n-i}}{i!(n-i)!}\int_0^n \frac{t(t-1)...(t-n)}{t-i}dt$$

Kwadratura dla $n = 2$ wygląda następująco:

$$S(f) = A_0 f_0 + A_1 f_1 + A_2 f_2$$

Liczę współczynnik kwadratury dla $i = 0$:

$$A_0 = h\frac{(-1)^{2-0}}{0!(2-0)!}\int_0^2 \frac{t(t-1)(t-2)}{t-0}dt$$
$$A_0 = h\frac{1}{2}\int_0^2 (t-1)(t-2)dt$$

Liczę całkę:

$$\int_0^2 t^2 - 3t + 2dt$$
$$[\frac{t^3}{3} - \frac{3t^2}{2} + 2t]|_0^2 = \frac{2}{3}$$

Czyli

$$A_0 = h\frac{1}{2} * \frac{2}{3} = \frac{h}{3}$$

Analogicznie postępuję dla $i = 1, 2$, podstawiam do wzoru na współczynnik kwadratury i otrzymuję:

$$S(f) = \frac{h}{3}f_0 + \frac{4h}{3}f_1 + \frac{h}{3}f_2$$
$$S(f) = \frac{h}{3}(f_0 + 4f_1 + f_2)$$

gdzie h to długość podprzedziałów: $h = \frac{b-a}{n}$.
Otrzymany wzór to wzór Simpsona. Niski rząd kwadratury może nie zapewnić wymaganej dokładności, dlatego zastosowałem kwadrature złożoną. Przedział całkowania [a, b] dzieli się na m podprzedziałów (m jest parzyste) w których stosuje się wzór Simpsona. Otrzymane wyniki cząstkowe sumuje się. Złożony wzór Simpsona moża zapisać jako:

$$S(f) = \frac{h}{3} \sum_{i=1}^{m/2} (f_{2i-2} + 4f_{2i-1} + f_{2i})$$

Czyli

$$S(f) = \frac{h}{3}[f_0 + 4(f_1 + f_3 + ... + f_{m-1}) + 2(f_2 + f_4 + ... + f_{m-2}) + f_m]$$

Zastosowanie kwadratury złożonej prowadzi do zmniejszenia błędu. Mając podaną tolerancję ϵ stosuję procedurę iteracyjnego zagęszczania podziałów do czasu, gdy kolejne znalezione przybliżenia całki różnią się od siebie wystarczająco mało, czyli $|I_{k+1} - I_k| < \epsilon$.

2.2 Metoda Romberga

Obliczenie całki metodą Romberga zacząłem od skonstruowania trójkąta:

$$\begin{array}{cccccc}
R_{0,0} & & & & & \\
R_{1,0} & R_{1,1} & & & & \\
R_{2,0} & R_{2,1} & R_{2,2} & & & \\
R_{3,0} & R_{3,1} & R_{3,2} & R_{3,3} & & \\
R_{4,0} & R_{4,1} & R_{4,2} & R_{4,3} & R_{4,4} & \\
\dots & \dots & \dots & \dots & \dots & \dots
\end{array}$$

Do obliczenia przybliżeń całek $R_{n,m}$ dla $h_n = \frac{b-a}{2^n}$ skorzystałem ze wzorów:

$$R_{0,0} = h_1(f(a) + f(b)) \text{ - wzór trapezów}$$
$$R_{n,0} = \frac{R_{n-1,0}}{2} + h_n \sum_{k=1}^{2^{n-1}} f(a + (2k-1)h_n)$$
$$R_{n,m} = R_{n,m-1} + \frac{R_{n,m-1} - R_{n-1,m-1}}{4^m - 1}$$

Zauważyłem, że do obliczenia $R_{n,m}$ potrzebowałem elementu z tego samego wiersza oraz elementu z wiersza powyżej. Dzięki temu uniknąłem wielokrotnego obliczania tego samego elementu. Podobnie jak w poprzedniej metodzie, liczę przybliżenia całek w kolejnych wierszach dopóki znalezione przybliżenia całki różnią się od siebie wystarczająco mało, czyli $|R_{n,n} - R_{n-1,n-1}| < \epsilon$

3 Implementacja

Opisane metody obliczenia całki zaimplementowałem w języku python. Obie funkcje przyjmują cztery argumenty, tj. funkcja do scałkowania, granice całkowania oraz tolerancja ϵ.

3.1 Kwadratura Newtona-Cotesa

Do obliczenia całki metodą złożonej kwadratury Newtona-Cotesa z $n = 2$ służy funkcja *simpson*:

```
def simpson(function, a, b, e):
    h = (b - a) / 2
    I = function(a) + function(b)
    I += 4 * function((a + b) / 2)
    I *= h / 3

    error = e
    while error >= e:
        Iprev = I
        h /= 2
        xi = a + h
        # first_sum = f_1 + f_3 + ... + f_(n-1)
        # f_i = function(x_i)
        first_sum = 0
        while xi < b:
            first_sum += function(xi)
            xi += 2 * h
        xi = a + 2 * h
        # second_sum = f_2 + f_4 + ... + f_(n-2)
        # f_i = function(x_i)
        second_sum = 0
        while xi < b:
            second_sum += function(xi)
            xi += 2 * h
        I = (h / 3) * (function(a) + 4 * first_sum + 2 * second_sum
     + function(b))
        error = abs(I - Iprev)

    return I
```

3.2 Metoda Romberga

Do obliczenia całki metodą Romberga służy funkcja *romberg*:

```
def romberg(function, a, b, e):
    h = b - a
    # R - aktualny wiersz
    R = [(h / 2) * (function(a) + function(b))]
    n, m = 1, 1
    error = e

    while error >= e:
        n += 1
        m *= 2
        h /= 2
        # Rprev - poprzedni wiersz
        Rprev = R

        suma = sum([function(a + h * i) for i in range(1, m, 2)])
        R = [0.0] * n
        R[0] = Rprev[0] / 2 + h * suma  # R(n-1, 0)
```

```
        pow4 = 4

        for i in range(1, n):
            R[i] = R[i - 1] + (R[i - 1] - Rprev[i - 1]) / (pow4 -
1)    # reszta wiersza
            pow4 *= 4

        error = abs(R[-1] - Rprev[-1])  # |R(n-1, n-1) - R(n-2, n
-2)|

    return R[-1]
```

Program przechowuje jednocześnie tylko dwa wiersze. Wystarcza to do obliczenia następnego, a tym samy do obliczenia skrajnego elementu następnego wiersza (przybliżenia całki) i zbliżenia się do oczekiwanej wartości.

4 Wyniki

Korzystając z zaimplementowanych metod całkowania numerycznego obliczyłem całkę $\int_0^1 sin(x)dx$. Wyniki otrzymane za pomocą napisanych funkcji są sobie równe i wynoszą 0.4596976941. Do sprawdzenia wyniku skorzystałem z funkcji *quad* z biblioteki *scipy*. Jest on zgodny z tymi otrzymanymi za pomocą zaimplementowanych metod.