13 Задача наближеного обчислення суми степеневого ряду

13.1 Програмування обчислення з використанням рекурентних співвідношень

Розглянемо задачу наближеного обчислення нескінченної суми

$$S(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots + \frac{(-1)^n x^{2n+1}}{(2n+1)!} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

Введемо позначку $a_n = \frac{(-1)^n x^{2n+1}}{(2n+1)!}$. Тоді $S(x) = \sum_{n=0}^{\infty} a_n$.

Відомо, що послідовність часткових сум (S_k), де

$$\mathrm{S_k} = \mathrm{x} - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots + \frac{(-1)^k x^{2k+1}}{(2k+1)!} = \sum_{n=0}^{k} a_n$$

за $-1 < x < 1$ «достатньо швидко» збігається до $S(x) = \sin x$, тобто функцію $\sin x$ на інтервалі $x \in (-1;\ 1)$ можна зобразити сумою степеневого ряду

$$\sin \mathrm{x} = \sum_{n=0}^{\infty} a_n \text{, де } a_n = \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

Значення $\sin x$ за $x \in (-1;\ 1)$ можна обчислити наближено як перше S_n, для якого $|a_n| = |S_n - S_{n-1}| < \varepsilon$, де ε — деяка додатна константа. Це ε назвемо «точністю» обчислення (насправді, на відміну від цього конкретного прикладу, у більшості випадків точністю воно якраз і не буде).

Послідовність сум (S_k) задовольняє співвідношення

$S_k = S_{k-1} + a_k$ за $k > 0$, а $S_0 = a_0$.

У першому наближенні алгоритм обчислень має такий вигляд (ім'ям **eps** позначено ε).

```
a=x; s=a; k=1;     #a = a_{k-1}
while fabs(a)>=eps:
    обчислити a=a_k
    s += a
    k += 1
```

Формула наступного члена ряду a_k проста, але, якщо застосувати її безпосередньо, то для кожного члена ряду доведеться обчислювати степінь і факторіал. Проте послідовність доданків теж неважко описати рекурентно: $a_0 = x$ і $a_k = a_{k-1} \frac{-x^2}{2k(2k+1)}$ за $k > 0$. Звідси, щоб отримати наступне значення a_k, треба лише попереднє a_{k-1} помножити на $-x^2$ і поділити на $2k(2k+1)$.

Отже, обчислювані величини описуються такою системою рекурентних співвідношень.

$a_0 = x$ і $a_k = a_{k-1} \frac{-x^2}{2k(2k+1)}$ за $k > 0$

$S_0 = a_0$, $S_k = S_{k-1} + a_k$ за $k > 0$.

За цієї системи ***обчислити*** **$a=a_k$;** у тілі циклу можна уточнити так.

```
a*=(-x*x)/(2*k)/(2*k+1);
```

Якщо в циклі обчислюється деякий вираз з незмінним значенням, краще присвоїти його допоміжній змінній перед циклом і використовувати її в циклі.

Покращимо наведений алгоритм. Помітимо: щоразу відбувається множення на **-x*x**, але **x** у циклі не змінюється! Тому можна перед циклом запам'ятати **-x*x** в допоміжній змінній: **x2 = -x*x**. Так само на кожній ітерації двічі обчислюється вираз **2*k**. Уведемо поняття «подвоєний номер доданка» і в цьому значенні вживатимемо змінну **k2**, перед циклом присвоївши їй **2**.

Оформимо алгоритм обчислень як функцію з параметром **x** (**x** ≤ 1) та точністю **eps**. Імена змінних відповідають уже введеним позначенням.

```
import math
def sum(x: float, eps: float) -> float:
    """
    Обчислення функції sin(x) з точністю eps.

    pre: -1 < x < 1, eps > 0, інакше можливе зациклення
    """

    a = x
    s = a
```

```
    k2 = 2  # подвоєний номер доданка
    x2 = -x * x
    while math.fabs(a) >= eps:
        a *= x2 / k2 / (k2 + 1)
        s += a
        k2 += 2
    return s
```

Наведену функцію можна ще трохи покращити (це покращення стосується переважно компільованих мов програмування). Значення *x* використовується тільки один раз на початку циклу. Тому значення **-x*x** можна зберегти не в новій змінній **x2**, а використати для цього змінну **x**, виконавши відповідне присвоювання.

У багатьох випадках при обчисленні достатньо використовувати якесь помірковано невелике значення **ε**, наприклад 10^{-6}. Таке значення для параметра **eps** функції **sum** можна вважати «стандартним». Якщо вирішити завжди в якості **eps** використовувати значення 10^{-6} й використовувати в алгоритмі цю константу, то для обчислень з кращою точністю вже прийдеться писати іншу, аналогічну, функцію. З іншого боку, якщо «стандартної» точності достатньо, то нема бажання в кожному виклику її явно вказувати (а для цього це стандартне значення треба кожного разу згадувати чи розраховувати). Отже, задамо для точності **eps** значення за умовчанням.

```
def sum(x: float, eps: float = 10e-6) -> float:
    """
    Обчислення функції sin(x) з точністю eps.

    pre: -1 < x < 1, eps > 0, інакше можливе зациклення
    """
    a = x
    s = a
    k2 = 2       # подвоєний номер доданка
    x = -x * x  # тепер x зберігає мінус квадрат вхідного значення
    while math.fabs(a) >= eps:
        a *= x / k2 / (k2 + 1)
        s += a
        k2 += 2
    return s
```

Примітка. Найбільш розповсюджена помилка під час обчислення таких сум – зсув значення *k*.

Для перевірки функції треба забезпечити, щоб в її викликах в якості **x** були аргументи 0, значення, близькі до 1 та **−**1, а також проміжні значення, наприклад, 0.5, **−**0.5; в якості **eps** варто взяти не дуже великі додатні числа, наприклад 10^{-6}.

Додатково зауважимо: якщо значення *x* велике за модулем, то вказана в умові сума збігається повільно, і за великої кількості ітерацій навіть виникають числа, які не можна зобразити в типі **float**. Проте на відрізку [**−**π; π] обчислення безпечні, а за *x* зовні цього відрізку слід скористатися тим, що $\sin(x+2k\pi) = \sin x$ за будь-якого цілого *k*.

13.2 Стратегії обробки помилок у математичних розрахунках

Розроблена функція цілком покладається на те, що її використовують з відповідними значеннями аргументів. У загальному випадку обчислення суми степеневого ряду за великих за абсолютною величиною значень **x** сума степеневого ряду може не збігатися. При цьому значення a_k може зменшуватися дуже повільно, але цикл завершиться і ми все одно отримаємо якесь значення, яке, зрозуміло, не може бути сумою незбіжного ряду. Або значення a_k може зростати за абсолютною величиною і в якійсь момент значенням змінної **x** стане нескінченність та цикл ніколи не завершиться. За нулевого або від'ємного значення **eps** цикл також не зупиниться.

Чи варто покращити безпечність функції **sum**, додавши перевірку, що параметри виклику **x** та **eps** відповідають попереднім вимогам на виклик функції? Якщо ми збираємося використовувати в програмі невелику кількість її викликів, то є сенс зробити так, щоб функція контролювала, чи зможе вона виконати обчислення над своїми аргументами. Але такого сорту функції частіше за все використовуються в обчислювальних алгоритмах, де викликаються сотні та тисячі разів, наприклад, для послідовних значень аргументу з деяким невеликим кроком у задачі інтегрування. Тоді має сенс забезпечити, щоб той інтервал, для значень якого викликається функція, задовольняв умовам, що накладаються на функцію. Тим самим ми виконаємо перевірку на застосовність функції **один раз** перед початком великої кількості її викликів і не будемо гальмувати чисельний алгоритм багатократними зайвими перевірками. Аналогічне стосується і перевірки значення **eps**.