Ітераційні цикли

Введемо позначення:

- i – номер повторення (ітерації);
- $x^{(i)}, x^{(i+1)}$ – поточне та наступне значення змінної циклу;
- $x^{(0)}$ – початкове значення змінної циклу, яке повинно бути задане до початку повторень (ітерацій);
- $x^{(i+1)} = \varphi(x^{(i)})$ – ітераційна формула, за якою ведеться обчислення наступного значення x через попереднє – залежність, з допомогою якої при кожному її виконанні визначається наступне значення змінної циклу $x^{(i+1)}$ через її попереднє значення $x^{(i)}$;
- ε – точність обчислення.

Приклади програмування задач, які реалізують ітераційні циклічні процеси

<u>Приклад1.</u> Скласти програму обчислення кореня *n*-го степеня із заданого числа *x* $y = \sqrt[n]{x}$ **з точністю** $\varepsilon>0$**, користуючись ітераційною формулою** $y_{n+1} = \dfrac{1}{n}\left(\dfrac{x}{y_n^{n-1}} + (n-1)y_n\right)$.

Програма розв’язку поставленої задачі має вигляд:

```
#include <iostream.h>
#include <math.h>

int main( )
        {
        float  x,  y,  y1,  eps;
        int n;

        cout<<”Enter start point – x, accuracy – eps  and  power – n\n”;
        cin>>x>>eps>>n;
```

```
y=x;
y1=(1.0/n)*(x/pow(y, n-1)+(n-1)*y);

while (fabs(y1-y)>eps)
        {
        y=y1;
        y1=(1.0/n)*(x/pow(y, n-1)+(n-1)*y);
        };
cout<<"root of value   is  "<<y1;

return (0);
}
```

Приклад 2. Скласти програму обчислення значення суми з нескінченним числом членів ***(ряду)***, зокрема, обчислення косинуса шляхом його розкладу в ряд:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots + = \sum_{n=0}^{\infty} y_n$$

де загальний член ряду задається формулою $y_n = (-1)^n \dfrac{x^{2n}}{(2n)!}$.

Для оптимізації обчислень представимо загальний член ряду за рекурентною формулою: $y_n = y_{n-1} \cdot p_n$.

Звідси $y_{n-1} = (-1)^{n-1} x^{2(n-1)} / (2(n-1))! = (-1)^{n-1} x^{2n-2} / (2n-2)!$

Тоді $p_n = y_n / y_{n-1} = -x^2 / ((2n-1) \cdot 2n)$

Програмна реалізація даної задачі має вигляд:

```
#include <iostream.h>
#include <math.h>

int main( )
```

```
{       float  x,  yn,  pn,  s,  eps;
int n;
cin>>x>>eps;
n=0;
yn=1;
s=yn;
        while (fabs(yn)>eps)
                {           n=n+1;
                            pn= -sqr(x) / (2*n*(2*n-1));
                            yn=yn*pn;
                        s=s+yn;         };
cout<<‘cos‘<<x<<’=  ’<<s;
return (0); }
```

Приклад 3. Скласти програму обчислення значення кореня нелінійного рівняння $x = \varphi(x)$, користуючись ітераційною формулою, яка реалізує деякий чисельний метод із заданою точністю ε, якщо відоме початкове наближення кореня x_0. Обчислювальний процес продовжувати до тих пір, поки різниця між двома послідовними наближеннями до кореня перевищує задану точність ε .

Нехай в конкретному випадку права частина нелінійного рівняння має вигляд $\varphi(x) = 3 - \ln x$, тоді ітераційна формула така

$$x_i = 3 - \ln 3x_{i-1}\,, \text{ де } i = 1, 2, \ldots$$

Алгоритм, що реалізує дане завдання на звичайній мові, наступний:

1. Потрібно ввести початкове наближення кореня x_0, та точність обчислення виразу ε $(x_0=3;\ \varepsilon=0{,}01)$;
2. Значенню x присвоїти початкове наближення кореня x_0 $(x=x_0)$;
3. Обчислити значення функції y при початковому наближенні кореня x_0 $(y=3-\ln x)$;
4. В тілі циклу ***while*** обчислюємо наступне наближення до кореня поки $|y-x| > \varepsilon$. Якщо дана умова виконується (точність недосягнута), то значенню x присвоюється попереднє значення y, а y обчислюється за рекурентною формулою

($x = y; y = 3 - \ln x;$). У випадку не виконання умови (точність досягнута) - виводиться попереднє та наступне наближення до кореня - значення x та y.

5. Кінець.

Програма, що реалізує даний алгоритм має вигляд:

```
#include <iostream.h>
#include <math.h>

int main( )
    {       float x0, x, y, eps;
            cout<<"Enter start point – x0 and accuracy – eps\n";
    cin>>x0>>eps;
    x=x0;
    y=3-log(x);

    while (fabs(y-x)>eps)
                    {       x=y;
                            y=3-log(x);         };
    cout<<" root of  the equation is"<<y;
    return (0);    }
```

<u>Приклад 4.</u> Скласти програму для обчислення границі функції (в точці або в нескінченності) методом табулювання заданої функції. Область табулювання вибирається так, щоб змінюючи на ній відповідним чином значення аргумента ***x***, можна було б встановити наближене значення границі функції ***f(x)*** (під цим розуміється, що така границя існує). При аналізі границі функції в точці, крок табулювання вибирається так, щоб значення аргумента наближалось до граничного. Звичайно, при цьому крок табулювання стає змінною. Точно так проводиться аналіз границі функції, якщо $x \to \pm\infty$ з використанням швидкого збільшення кроку табулювання, що дозволяє аналізувати поведінку функції при достатньо великих за абсолютною величиною значеннях аргумента.

Розглянемо конкретний приклад визначення границі функції

$$\lim_{x\to a}\frac{1-\cos(x-a)}{0{,}5\cdot(x-a)}.$$

Розв’язок буде мати наступний вигляд: вираз під знаком границі при $x = a$ дає так звану „невизначеність типу $\frac{0}{0}$”. Закон зміни аргумента задамо за формулою

$$x_k = a + \frac{1}{2^k}.$$

Тоді через деяке число кроків k, на кожному з яких визначається і виводиться на друк значення функції $f(x)$, може виникнути одна з наступних ситуацій:

а) модуль різниці $\left|f(x_k) - f(x_{k-1})\right|$ стає меншим деякого малого, наперед заданого числа ε. В такому випадку вважаємо, що функція має кінцевий результат в точці $x = a$ і приймаємо його приблизно рівним $f(x_k)$;

б) модуль різниці $\left|f(x_k) - f(x_{k-1})\right|$ стає більшим деякого досить великого наперед заданого числа M. Якщо при цьому отриманий ряд значень функції не є знакозмінним, можна стверджувати, що функція прямує до нескінченності при $x \to a$;

в) якщо після наперед заданої, довільно великої кількості ітерацій N не вдалося реалізувати варіант а) чи варіант б), то будемо вважати, що дана функція не має границі в точці $x = a$.

При розробці алгоритму розв’язку задачі необхідно врахувати необхідність збереження як поточного, так і попереднього значення функції. Позначимо їх f та y відповідно. На кожному кроці (крім першого) проводитимемо їх порівняння.

Програмна реалізація даної задачі має вигляд:

```
#include <iostream.h>
#include <math.h>

int main( )
    { float eps, M, N, a, x, f, y;
    int k;
    cout<<” Enter accuracy eps:”;
    cin>>eps;
    cout<<” Enter big number M:”;
    cin>>M;
    cout<<” Enter max number of iteration N:”;
    cin>>N;
    cout<<” Enter number a ”;
```

```
        cin>>a;
        k=1;
m1:     x=a+1.0/pow(2,k);
        f=(1-cos(x-a))/(0.5*(x-a));
        cout<<"\n x="<<x<<"   f="<<f;
        if  (k==1)   goto  m2;
                else {
                        if (fabs(y-f)<eps)
                                {
                                        cout<<"\n f="<<f;
                                        goto m3;
                                };
                                else {
                        if ((y*f>0)&&(fabs(y-f)>M))
                        {
                        cout<<"\n function tends to infinity"; // ф-я прямує до нескінченності
                        goto m3;
                        };
                                else {
                        if (k>N)
                        {
                        cout<<"\n the border does not exist";
                        goto m3;
                        };
                        else {
        m2:                     y=f; k=k+1;
                                goto m1;
                        }; }; };  };
        m3: return (0);   }
```

У випадку визначення границі функції на нескінченності програма буде мати аналогічний вигляд, за виключенням визначення кроку. З ціллю введення швидко збільшуючого кроку, закон зміни *x* доцільно задавати у вигляді $x_k = 2^k$.