Отчёт по практическому заданию №1.

Козлов Кирилл 305 группа

6 октября 2024 г.

Содержание

I Постановка задачи.

1. Вычислить таблицу значений функции $y = \cos(x/10) \cdot \arctan(x)$ на сетке, полученной разбиением отрезка [0.5,1.5] на 20 равных частей:

$$y_i = \cos(x_i/10) \cdot \arctan(x_i), \quad x_i = 0.5 + i \cdot h, \quad h = \frac{1.5 - 0.5}{20}, \quad (i = 0, 1, ..., 20)$$

2. Построить 3 полинома $p_n(x) = a_0 \cdot x^n + a_1 \cdot x^{n-1} + ... + a_n$ последовательных степеней $n = 2, 3, 4$, дающих наилучшее среднеквадратичное приближение $f(x)$ на сетке x_i , т.е. для каждого n найти точку $(a_0, a_1, ..., a_n)$ в R_{n+1}, в которой достигается минимум функции

$$\Phi(a_0, a_1, ..., a_n) = \sum_{i=0}^{20} [y_i - p_n(x_i)]^2$$

Минимизацию $\Phi(a_0, a_1, ..., a_n)$ будем проводить усовершенствованным методом градиентного спуска - **метод сопряженных градиентов Флетчера-Ривса.** В качестве начальной точки будем брать $a_0^{(0)} = a_1^{(0)} = ... = a_n^{(0)} = 0$. За меру точности приближения примем

$$\triangle_{k+1} = \|x^{(k+1)} - x^{(k)}\|$$

Процесс приближений заканчивается , если $\triangle_k < \varepsilon$,где $\varepsilon = 10^{-4}$

3. Для каждого из полученных таким образом $p_n(x)$ вычислить $y_i^{(n)} = p_n(x_i) \quad (i = 0, 1, ..., 20)$ и соответствующее среднеквадратическое отклонение:

$$\sigma_n = \sqrt{\frac{1}{21} \cdot \sum_{i=0}^{20} [y_i - p_n(x_i)]^2} = \sqrt{\frac{\Phi_{min}}{21}}$$

II Метод спуска для отыскания минимума функции многих переменных(метод сопряженных градиентов Флетчера-Ривса).

1. Пусть требуется найти локальный минимум функции

$$\Phi(x) = \Phi(x_1, x_2, ..., x_n)$$

Выберем начальное приближение $x^{(0)}$ и построим последовательность $x^{(0)}, x^{(1)}, x^{(2)}, ..., x^{(k)}, ...$, в которой каждое очередное приближение $x^{(k+1)}$ получается из предыдущего $x^{(k)}$ "шагом"в направлении вектора $v^{(k)}$:

$$x^{(k+1)} = x^{(k)} + \lambda_k \cdot v^{(k)}$$

Величина шага управляется выбором λ_k. Обычно λ_k выбирается так, чтобы функция

$$\psi_{(k)} \equiv \Phi(x^{(k)} + \lambda \cdot v^{(k)})$$

принимала наименьшее значение. Точность полученного приближения к минимуму обычно оценивается по величине отклонения двух соседних приближений $x^{(k)}$ и $x^{(k+1)}$, например

$$\triangle_{k+1} = \|x^{(k+1)} - x^{(k)}\|$$

Процесс приближений заканчивается, если $\triangle_k < \varepsilon$, где ε - заданная точность.

2. В методе Флетчера-Ривса направление спуска - вектор $v^{(k)}$ - выбирается следующим образом:

$$v^{(0)} = -grad\Phi(x^{(0)})$$

$$v^{(k+1)} = -grad\Phi(x^{(k+1)}) + \beta_k \cdot v^{(k)}$$

,где

$$\beta_k = \frac{|grad\Phi(x^{(k+1)})|^2}{|grad\Phi(x^{(k)})|^2} \quad (k = 0, 1, 2, ...)$$

Таким образом , на каждом шаге спуск несколько отклоняется от "скорейшего"в направлении предыдущего шага, что даёт некоторое ускорение сходимости вблизи точки минимума (когда близок к нулю $grad\Phi(x)$). В случае квадратичной функции $\Phi(x)$ описываемые методы сходятся при любом выборе начального приближения.$x^{(0)}$

3. Для нахождения λ_k на каждом шаге спуска нужно находить минимум функции одной переменной $\psi_k(\lambda)$, определяемой равенством $\psi_{(k)} = \Phi(x^{(k)} + \lambda \cdot v^{(k)})$. В общем случае для этого можно решать уравнение $\frac{d\psi_k(\lambda)}{d\lambda} = 0$, выражающее условие минимальности $\psi_k(\lambda)$. В нашем случае для минимальности функции $\psi_k(\lambda)$, λ выражается следующим образом:

$$\lambda = \frac{\sum_{i=0}^{20}(f(x_i) \cdot V) - \sum_{i=0}^{20}(V(x_1^{(k)} \cdot x_i^n + x_2^{(k)} \cdot x_i^{n-1} + ... + x_k^{(k)}))}{\sum_{i=0}^{20}(V^2)}$$

, где

$$V = v_1^{(k)} \cdot x_i^n + v_2^{(k)} \cdot x_i^{n-1} + ... + v_k^{(k)}$$

4. Для контроля за процессом сходимости спуска на каждом шаге удобно выдавать на печать величины: k (номер шага), $\triangle_k$, $g_k = |grad(\Phi(x^k))|$. Метод спуска сходится к точке минимума $\Phi(x^k)$, если:

1. $\Phi(x^k)$ убывает;
2. $g_k \to 0$;
3. $\triangle_k \to 0$;

III Блок-схема программы.

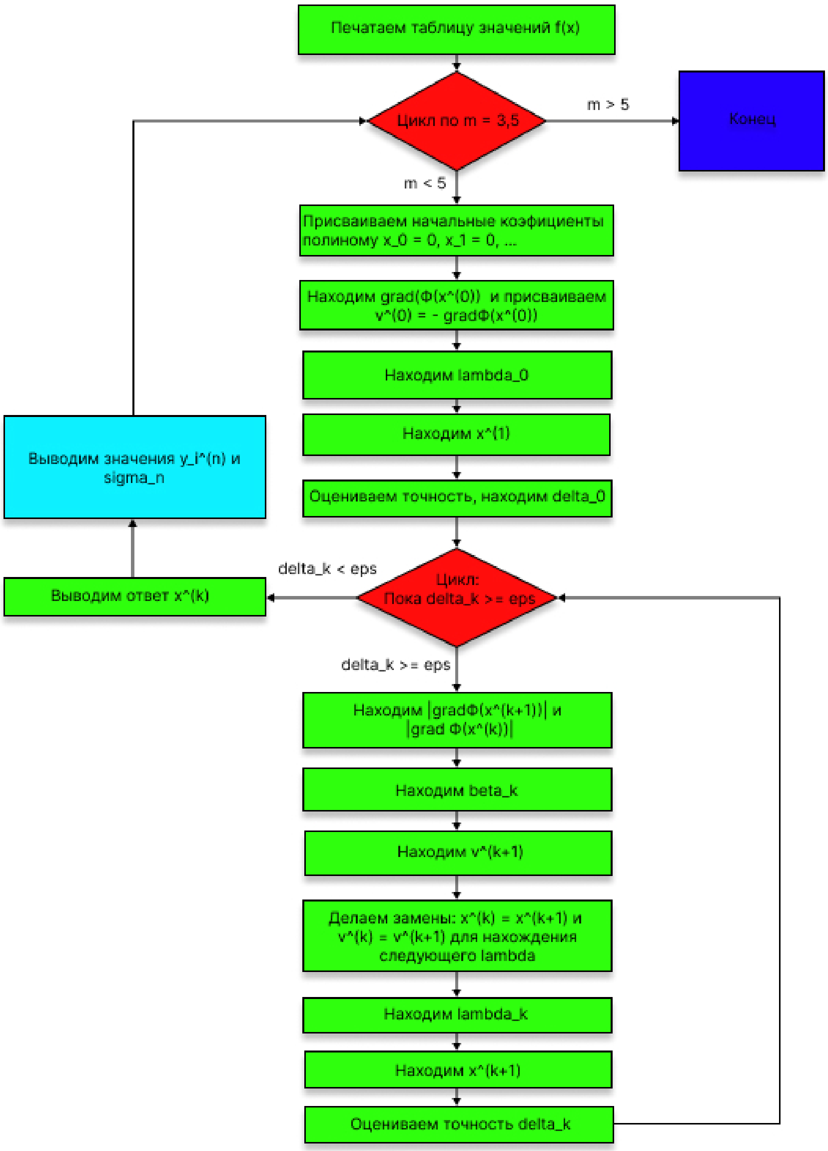

Рис. 1: Блок-схема программы

IV Программа решения задачи на Fortran.

```
! cos(x/10) * arctg(x)
! c = 0.5 , d = 1.5
! N = 20 , n0 = 2 , m = 3
module compute_module
    implicit none
contains

    function compute_func(c,i,h)
        real, intent(in) :: c, h
        integer, intent(in) :: i
        real :: compute_func
        compute_func = cos((c + i*h) /10) * atan(c+i*h)
    end function compute_func

    function compute_scalar(c,i,h,vector,size)
        ! m - number of coefficients (dimension of the polynomial)
        real, intent(in) :: c, h
        integer, intent(in) :: i, size
        integer :: order_number
        real :: vector(size)
        real :: compute_scalar

        ! calculate the sum a0*x^n + a1*x^(n-1) +... + an
        compute_scalar = 0.0
        do order_number = 1,size
            compute_scalar = compute_scalar + vector(order_number) * ((
               c + i*h)**(size-order_number))
        end do

    end function compute_scalar
end module compute_module



program call_sub
    use compute_module
    implicit none
    real :: c, d, h, eps , func, F_k , derevative_value ,  lambda_k,
       delta_k_1, beta_k, module_grad_k, module_grad_k_1 , y_i, sigma_n
        , derev_k_1,  derev_k, term_1, term_2, term_3
    integer :: N_big , n0, m_number, i, j, kol_iterations , m
    real , allocatable:: v_k(:) , v_k_1(:), x_k_1(:), x_k(:), grad_k(:)
       ,  grad_k_1(:)

    c = 0.5
    d = 1.5
    N_big = 20 ! number of split points

    n0 = 2 ! initial degree of polynomial
    m_number = 3 ! need to construct 3 polynomials with degrees 2,3,4

    eps = 0.0001 ! accuracy of approximation

    h = (d - c)/N_big ! grid step size
```

```
print *
print *,'␣␣␣␣␣␣␣␣␣␣␣␣Table␣of␣values:'
do i = 0, N_big
    func = compute_func(c,i,h)
    print "(A,␣F7.5,␣A,␣F7.5)", 'Value␣cos(x/10)*arctg(x)␣=␣',func,
        '␣with␣x␣=␣', c+i*h
end do
print *

do m = m_number, m_number+n0
    ! allocate memory for vectors
    allocate(v_k(m))
    allocate(v_k_1(m))
    allocate(x_k_1(m))
    allocate(x_k(m))
    allocate(grad_k(m))
    allocate(grad_k_1(m))

    if (m==m_number) then
        ! x_0 = 0 , if our first polynomial is n=n0
        do i = 1,m
            x_k(i) = 0.0
        end do
    else
        ! if not the first polynomial then a_0 = 0, a_1 = x_1, a_2
            = x_2, ...
        x_k(1) = 0.0
        do i = 2, m
            x_k(i) = x_k_1(i)
        end do
    end if

    delta_k_1 = 1000
    kol_iterations = 1

    ! find the gradient  v_0 = grad(F(x(0)))
    do j = 1, m
        derevative_value = 0.0
        do i = 0,N_big
            derevative_value = derevative_value +  (compute_func(c,
               i,h) - compute_scalar(c,i,h,x_k,m))*((c + i*h)**(m-j
               ))
        end do
        v_k(j) = -2.0 * derevative_value
    end do


    ! find |grad(F(x_0))|
    module_grad_k = 0.0
    do i = 1,m
        module_grad_k = module_grad_k + (v_k(i)**2)
    end do
    module_grad_k = sqrt(module_grad_k)

    ! v_0 = -grad(F(x(0)))
    do i = 1,m
        v_k(i) = -v_k(i)
```

```
end do

! find lambda_k, to find later x_k_1
term_1 = 0.0
term_2 = 0.0
term_3 = 0.0
lambda_k = 0.0
do i = 0,N_big
    term_1 = term_1 + compute_func(c,i,h)*compute_scalar(c,i,h,
       v_k,m)
    term_2 = term_2 + compute_scalar(c,i,h,v_k,m)*
       compute_scalar(c,i,h,x_k,m)
    term_3 = term_3 + compute_scalar(c,i,h,v_k,m)*
       compute_scalar(c,i,h,v_k,m)
end do
lambda_k = (term_1 - term_2)/term_3


! find x_k_1 == x_1
do i = 1,m
    x_k_1(i) = x_k(i) + lambda_k*v_k(i)
end do
! evaluate the accuracy of approximation to the minimum
delta_k_1 = 0.0
do i=1,m
    delta_k_1 = delta_k_1 + ((x_k_1(i) - x_k(i))**2)
end do
delta_k_1 = sqrt(delta_k_1)

print "(A,␣I2,␣A,␣F8.5,␣A,␣F8.5)" , "␣iteration␣=␣",
   kol_iterations,  "␣;␣|grad(x_k)|␣=␣", module_grad_k ,  "␣;␣
   delta_k_1␣=␣",delta_k_1

kol_iterations = kol_iterations + 1

do
    if (delta_k_1 < eps ) exit ! find the answer - x_k_1

    ! reset gradients
    do i = 1,m
        grad_k(i) = 0.0
        grad_k_1(i) = 0.0
    end do

    ! find grad(F(x_k_1)) and  grad(F(x_k))
    do j = 1, m
        derev_k_1 = 0.0
        derev_k = 0.0
        do i = 0, N_big
            derev_k_1 = derev_k_1 +  (compute_func(c,i,h) -
               compute_scalar(c,i,h,x_k_1,m)) * ((c + i*h)**(m-
               j))
            derev_k = derev_k +  (compute_func(c,i,h) -
               compute_scalar(c,i,h,x_k,m)) * ((c + i*h)**(m-j)
               )
        end do
        grad_k_1(j) = -2.0 * derev_k_1
        grad_k(j) = -2.0 * derev_k
```

```
end do


! find |grad(F(x_k_1))|^2
module_grad_k_1 = 0.0
do i = 1,m
    module_grad_k_1 = module_grad_k_1 + (grad_k_1(i)**2)
end do
module_grad_k_1 = sqrt(module_grad_k_1)

! find |grad(F(x_k))|^2
module_grad_k = 0.0
do i = 1,m
    module_grad_k = module_grad_k + (grad_k(i)**2)
end do
module_grad_k = sqrt(module_grad_k)

beta_k = (module_grad_k_1**2)/(module_grad_k**2) ! because
   of  |grad(F(x_k_1))|^2 / |grad(F(x_k))|^2


! find v_k_1
do i = 1,m
    v_k_1(i) = (-grad_k_1(i)) + (beta_k*v_k(i))
end do

! make substitutions to find the next one lambda_k
x_k = x_k_1
v_k = v_k_1

! find lambda_k, to find later x_k_1
term_1 = 0.0
term_2 = 0.0
term_3 = 0.0
lambda_k = 0.0
do i = 0,N_big
    term_1 = term_1 + compute_func(c,i,h)*compute_scalar(c,
       i,h,v_k,m)
    term_2 = term_2 + compute_scalar(c,i,h,v_k,m)*
       compute_scalar(c,i,h,x_k,m)
    term_3 = term_3 + compute_scalar(c,i,h,v_k,m)*
       compute_scalar(c,i,h,v_k,m)
end do
lambda_k = (term_1 - term_2)/term_3


! find x_k_1
do i = 1,m
    x_k_1(i) = x_k(i) + (lambda_k*v_k(i))
end do


! evaluate the accuracy of approximation to the minimum:
delta_k_1 = 0.0
do i = 1, m
    delta_k_1 = delta_k_1 + ( (x_k_1(i) - x_k(i)) **2)
end do
delta_k_1 = sqrt(delta_k_1)
```

```
            print "(A,␣I2,␣A,␣F8.5,␣A,␣F8.5)", "␣iteration␣=␣",
               kol_iterations,  "␣;␣|grad(x_k)|␣=␣", module_grad_k , "␣
               ;␣delta_k_1␣=␣",delta_k_1
            kol_iterations = kol_iterations + 1

        end do

        print *
        print *,"Answer:␣x_k_1␣=␣", x_k_1


        do i = 0,N_big
            y_i = 0.0
            y_i = compute_scalar(c,i,h,x_k,m)
            print "(A,␣I2,␣A,␣F8.5)", "␣y_",i, "␣=",  y_i
        end do

        F_k = 0.0
        do i= 0,N_big
            F_k = F_k + (compute_func(c,i,h) - compute_scalar(c,i,h,x_k
               ,m))**2
        end do

        sigma_n = sqrt(F_k/(N_big+1))
        print "(A,␣F7.5)","␣sigma_n␣=␣", sigma_n
        print *
        print *

        ! freeing up memory for vectors
        deallocate(v_k)
        deallocate(v_k_1)
        deallocate(x_k)
        deallocate(x_k_1)
        deallocate(grad_k)
        deallocate(grad_k_1)
    end do
end program call_sub
```

V Тесты работы программы.

1. Для функции $y = \cos(x/10) \cdot \arctan(x)$ на отрезке [0.5,1.5] получим следующие значения полиномов:

2 степень полинома, коэфициенты: $a_0 = -0.256448418, a_1 = 1.01533222, a_2 = 0.0225474089$

3 степень полинома, коэфициенты: $a_0 = 0.0746004581, a_1 = -0.480280489, a_2 = 1.22692657, a_3 = -0.0398121364$

4 степень полинома, коэфициенты: $a_0 = 0.115918584, a_1 = -0.392639726, a_2 = 0.198947981, a_3 = 0.807098985, a_4 = 0.0$

Теперь проверим нашу программу на произвольном полиноме,допустим, 3 степени:
$y = 1.122 \cdot x^3 - 0.213 \cdot x^2 + 5.323 \cdot x - 2.1231$.Если подставить эту фукцию в нашу программу, то получатся коэфициенты:

3 степень полинома, коэфициенты: $a_0 = 1.12239671, a_1 = -0.213400945, a_2 = 5.32244205, a_3 = -2.12260127$

При этом значение $|grad\Phi(x^{(k)})| \to 0$.

VI Таблицы полученных результатов.

Таблица 1: Таблица значений полинома 2 степени

Величина	Значение
a0	-0.256448418
a1	1.01533222
a2	0.0225474089
$\lvert$grad$((x^5))\rvert$	0.39869
σ_2	0.00161

Таблица 2: Таблица значений полинома 3 степени

Величина	Значение
a0	0.0746004581
a1	-0.480280489
a2	1.22692657
a3	-0.0398121364
$\lvert$grad$((x^{(9)}))\rvert$	0.00810
σ_3	0.00006

Таблица 3: Таблица значений полинома 4 степени

Величина	Значение
a0	0.115918584
a1	-0.392639726
a2	0.198947981
a3	0.807098985
a4	0.0528342798
$\lvert$grad$((x^{(8)}))\rvert$	0.07443
σ_4	0.00061

VII Перечень и характеристика всех ошибок, обнаруженных при прохождении задания.

Единственной и самой обидной ошибкой было: при возведении полинома и функции в степень , вместо **(m-j) я написал **m-j и из-за этой ошибки в моей программе получались неверные ответы, градиент не стремился к нулю. Над этой проблемой я думал несколько дней.