МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра информационных технологий**

**ОТЧЕТ № 2**

**по дисциплине  
  «Методы оптимизации»**

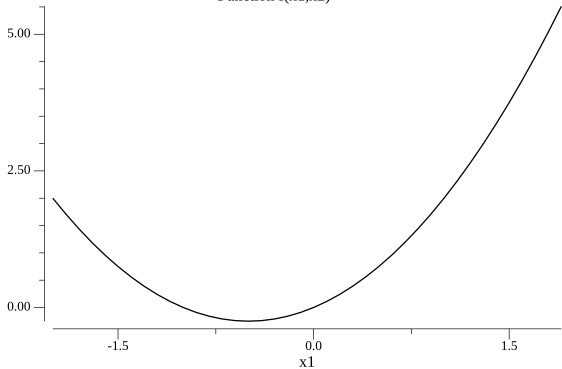
Выполнил студент группы 35/2                                              \_\_\_\_           А.В. Акулов

Направление подготовки  02.03.03  Математическое обеспечение и администрирование информационных систем

Курс    3

Краснодар

График



Классический метод

Шаг 1. Df/dx1; df/dx2

Шаг 2. Приравниваем производные к нулю, находим координаты точек Mi(x0,y0)

Шаг 3. Ищем вторые производные

Шаг 4. Для каждой точки считаем:  
А=df2/dx12; B= df2/dx22; C=df2/dx1x2

Шаг 5. Если AC-B2>0 – экстремум существует (A>0 – min; A<0 – max).

Если AC-B2<0 – экстремум не существует

func classical() {  
 // шаг 1: производные  
 // df/dx1 = 2x1 - x2 + 1  
 // df/dx2 = 10x2 - x1  
  
 // шаг 2: решаем систему  
 // 2x1 - x2 + 1 = 0 => x2 = 2x1 + 1  
 // 10x2 - x1 = 0 => x1 = 10x2  
 // вычисляем точки:  
 // x1 = 10(2x1 + 1) => x1 = 20x1 + 10 => -19x1 = 10 => x1 = -10/19  
 x1 := -10.0 / 19.0  
 x2 := 2\*x1 + 1  
 x := []float64{x1, x2}  
  
 // шаг 3: вторые производные  
 // d2f/dx1^2 = A = 2  
 // d2f/dx2^2 = B = 10  
 // d2f/dx1dx2 = C = -1  
  
 A := 2.0  
 B := 10.0  
 C := -1.0  
 D := A\*C - B\*B  
  
 typeExtremum := ""  
 if D > 0 {  
 if A > 0 {  
 typeExtremum = "минимум"  
 } else {  
 typeExtremum = "максимум"  
 }  
 } else {  
 typeExtremum = "нет экстремума"  
 }  
  
 fmt.Println("Классический метод:")  
 fmt.Printf("x\* = (%.6f, %.6f), f = %.6f, тип = %s\n", x[0], x[1], f(x), typeExtremum)  
}

Метод наискорейшего градиентного спуска

Шаг 1. Задаём х0, eps1, eps2, M. Вычисляем градиент функции в точке х0.

Шаг 2. K=0

Шаг 3. Находим норму градиента.

Шаг 4. Если норма градиента < eps1 => xk=x\*

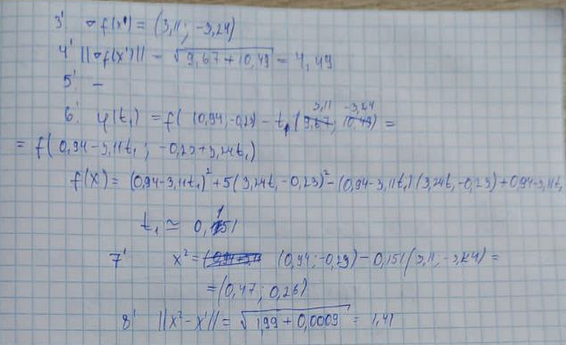
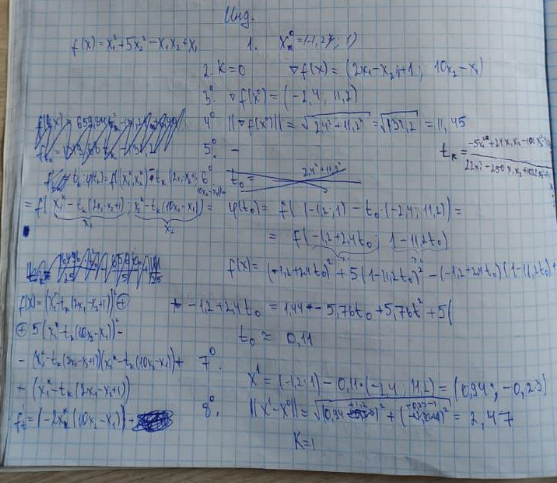
Шаг 5. ­­Если k>=m, xk=x\*

Шаг 6. tk: phi(t­­k)=f(xk-tk\*grad(f(xk)))

Шаг 7. xk+1=xk-tk\*grad(f(xk))

Шаг 8. если || xk+1 - xk || < eps2 и | f(xk+1) – f(xk) | < eps2 для к и к-1 выполняется, то xk=x\*

иначе – шаг 3



func steepestGD(x0 []float64, eps1, eps2 float64, M int) {  
 x := make([]float64, 2)  
 copy(x, x0)  
 k := 0  
 var cond string  
  
 for {  
 g := grad(x)  
 normG := norm(g)  
  
 // Шаг 4: Проверка нормы градиента  
 if normG < eps1 {  
 cond = "||grad(f)|| < eps1"  
 break  
 }  
  
 // Шаг 5: Проверка по числу итераций  
 if k >= M {  
 cond = "достигнуто максимальное число итераций"  
 break  
 }  
  
 // Шаг 6: Вычисляем шаг (по формуле наискорейшего спуска)  
 Hg := []float64{2\*g[0] - g[1], -g[0] + 10\*g[1]}  
 num := g[0]\*g[0] + g[1]\*g[1]  
 denom := g[0]\*Hg[0] + g[1]\*Hg[1]  
 alpha := num / denom  
  
 xNew := sub(x, mulScalar(g, alpha))  
  
 // Шаг 8: Проверка по изменениям  
 if norm(sub(xNew, x)) < eps2 && math.Abs(f(xNew)-f(x)) < eps2 {  
 x = xNew  
 cond = "ШАГ 8. выполнено условие по изменениям в x и f"  
 k++  
 break  
 }  
  
 x = xNew  
 k++  
 }  
  
 fmt.Println("Метод наискорейшего спуска:")  
 fmt.Printf("x\* = (%.6f, %.6f), f(x\*) = %.6f, число итераций = %d, остановка по условию: %s\n", x[0], x[1], f(x), k+1, cond)  
}

Метод Ньютона

Шаг 1. Задаём х0, eps1, eps2, M, H(x)

Шаг 2. K=0

Шаг 3. Вычисляем градиент функции в точке х0.

Шаг 4. Если || f(xk) || < eps1 => x\*=xk

Шаг 5. ­­Если k>=m, xk=x\*

Шаг 6. Вычисляем H(xk)

Шаг 7. Вычисляем H-1(xk)

Шаг 8. Если H-1(xk)>0, то

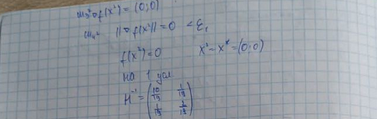
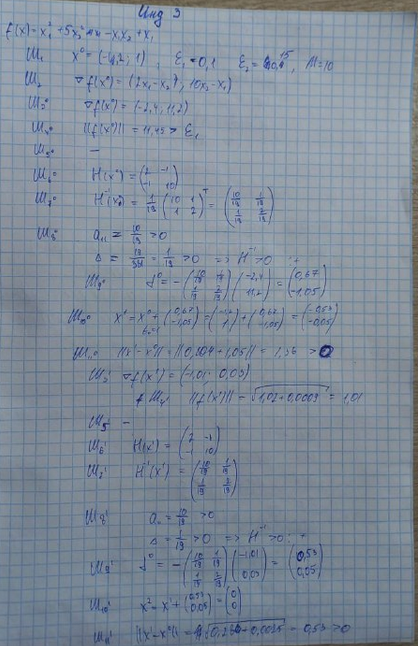
Шаг 9: dk= - H-1(xk) \* grad(f(xk))

Иначе dk= - grad(f(xk))

Шаг 10. Xk+1 = Xk + tk \* dk, tk = 1 если dk= - H-1(xk) \* grad(f(xk)). Иначе ищем как в наискорейшем градиентном спуске.

Шаг 11. если || xk+1 - xk || < eps2 и | f(xk+1) – f(xk) | < eps2 для к и к-1 выполняется, то xk=x\*

иначе – шаг 3



func newton(x0 []float64, eps1, eps2 float64, M int) {  
 x := make([]float64, 2)  
 copy(x, x0)  
 k := 0  
 var cond string  
 prev := false  
 for {  
 g := grad(x)  
 if norm(g) < eps1 {  
 cond = "||grad(f(xk))|| < eps1"  
 break  
 }  
 if k >= M {  
 cond = "достигнут предел итераций"  
 break  
 }  
 inv := invH()  
 d := []float64{-(inv[0][0]\*g[0] + inv[0][1]\*g[1]), -(inv[1][0]\*g[0] + inv[1][1]\*g[1])}  
 positiveDefinite := H[0][0] > 0 && H[1][1] > 0 && (H[0][0]\*H[1][1]-H[0][1]\*H[1][0]) > 0  
 t := 1.0  
 if !positiveDefinite {  
 d = mulScalar(g, -1)  
 t = dihotomia(func(t float64) float64 {  
 xTemp := sub(x, mulScalar(g, t))  
 return f(xTemp)  
 })  
 }  
 xNew := add(x, mulScalar(d, t))  
 if norm(sub(xNew, x)) < eps2 && math.Abs(f(xNew)-f(x)) < eps2 {  
 x = xNew  
 cond = "выполнены условия ||x\_{k+1}-x\_k|| и |f(x\_{k+1})-f(x\_k)| < eps2"  
 k++  
 if prev {  
 break  
 }  
 prev = true  
 }  
 x = xNew  
 k++  
 }  
 fmt.Println("Метод Ньютона:")  
 fmt.Printf("x\* = (%.6f, %.6f), f = %.6f, итерации = %d, остановка = %s\n", x[0], x[1], f(x), k, cond)  
}  
  
func dihotomia(phi func(float64) float64) float64 {  
 a := 0.0  
 b := 1.0  
 epsilon := 0.005  
 delta := epsilon / 3.0  
  
 for (b - a) > 2\*epsilon {  
 c := (a + b) / 2  
 x1 := c - delta  
 x2 := c + delta  
  
 if x1 < a {  
 x1 = a  
 }  
 if x2 > b {  
 x2 = b  
 }  
  
 f1 := phi(x1)  
 f2 := phi(x2)  
  
 if f1 < f2 {  
 b = c  
 } else {  
 a = c  
 }  
 }  
 return (a + b) / 2  
}

Метод Ньютона–Рафсона

Шаг 1 – Шаг 7 см. метод Ньютона

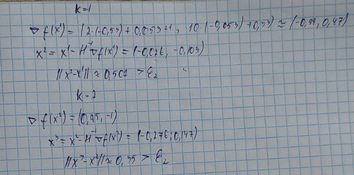
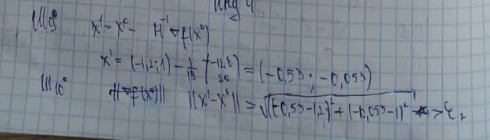
Шаг 8. Если H-1(xk)>0, то dk= - H-1(xk) \* grad(f(xk))

Иначе dk= - grad(f(xk))

Шаг 9. Xk+1 = Xk + tk \* dk, tk: phi(t­­k)=f(xk-tk\*dk)

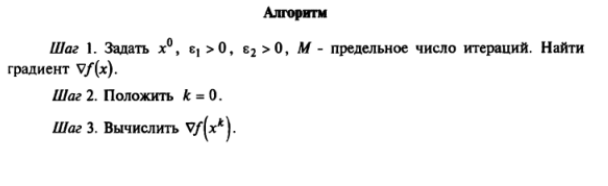
Шаг 10. если || xk+1 - xk || < eps2 и | f(xk+1) – f(xk) | < eps2 для к и к-1 выполняется, то xk=x\*

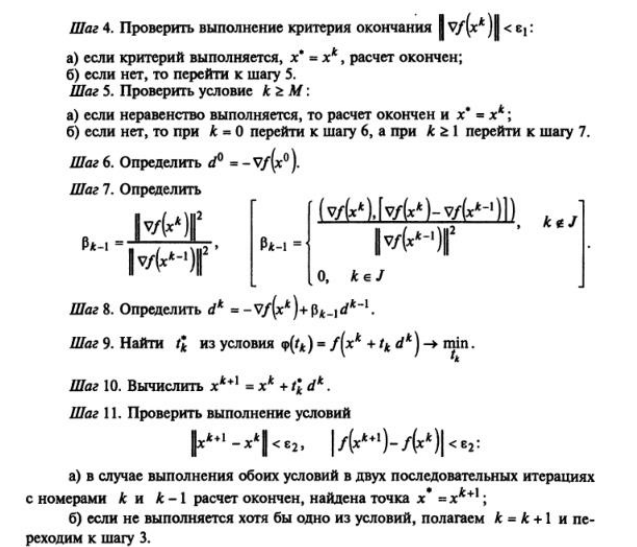
иначе – шаг 3.

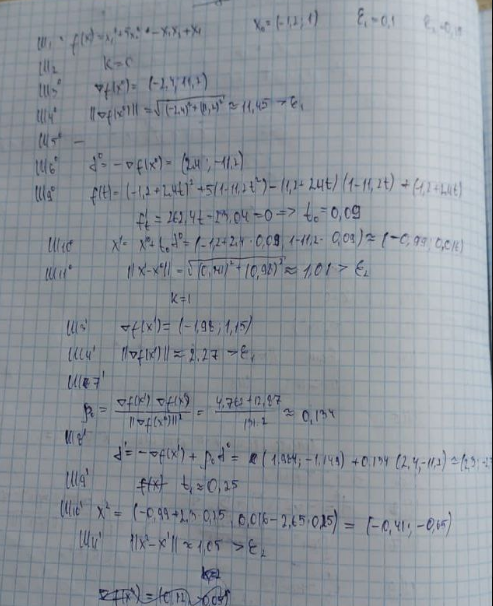


func newtonRaphson(x0 []float64, eps1, eps2 float64, M int) {  
 x := make([]float64, 2)  
 copy(x, x0)  
 var cond string  
 var k int  
 var fPrev float64 = f(x)  
 prev := false  
 for k = 0; k < M; k++ {  
 g := grad(x)  
 if norm(g) < eps1 {  
 cond = "||grad(f(x\_k))|| < eps1"  
 break  
 }  
  
 inv := invH()  
 d := []float64{-(inv[0][0]\*g[0] + inv[0][1]\*g[1]), -(inv[1][0]\*g[0] + inv[1][1]\*g[1])}  
 positiveDefinite := H[0][0] > 0 && H[1][1] > 0 && (H[0][0]\*H[1][1]-H[0][1]\*H[1][0]) > 0  
 t := dihotomia(func(t float64) float64 {  
 xTemp := sub(x, mulScalar(g, t))  
 return f(xTemp)  
 })  
 if !positiveDefinite {  
 d = mulScalar(g, -1)  
 }  
 xNew := add(x, mulScalar(d, t))  
 if norm(sub(xNew, x)) < eps2 && math.Abs(f(xNew)-fPrev) < eps2 {  
 x = xNew  
 cond = "||x\_{k+1}-x\_k|| < eps2 и |f(x\_{k+1}) - f(x\_k)| < eps2"  
 k++  
 if prev {  
 break  
 }  
 prev = true  
 }  
 x = xNew  
 fPrev = f(x)  
 }  
  
 if cond == "" {  
 cond = "достигнуто максимальное число итераций"  
 }  
 fmt.Println("Метод Ньютона-Рафсона:")  
 fmt.Printf("x\* = (%.6f, %.6f), f = %.6f, итерации = %d, остановка = %s\n", x[0], x[1], f(x), k+1, cond)  
}

Метод Флетчера–Ривза

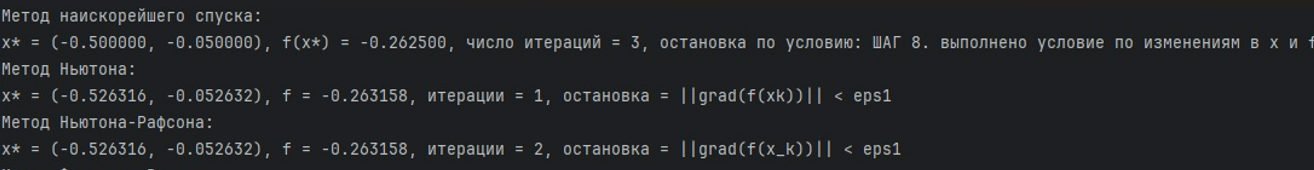






func fletcherReeves(x0 []float64, eps1, eps2 float64, M int) {  
 x := make([]float64, 2)  
 copy(x, x0)  
 g := grad(x)  
 d := mulScalar(g, -1)  
 k := 0  
 var cond string  
  
 for {  
 if norm(g) < eps1 {  
 cond = "||∇f(x^k)|| < ε1"  
 break  
 }  
 if k >= M {  
 cond = "k ≥ M"  
 break  
 }  
  
 phi := func(t float64) float64 {  
 step := mulScalar(d, t)  
 return f(sub(x, step))  
 }  
 t := minimizePhi(phi)  
  
 xNew := add(x, mulScalar(d, t))  
 fOld := f(x)  
 fNew := f(xNew)  
  
 if norm(sub(xNew, x)) < eps2 && math.Abs(fNew-fOld) < eps2 {  
 x = xNew  
 cond = "||x^{k+1}-x^k|| < ε2 и |f(x^{k+1}) - f(x^k)| < ε2"  
 break  
 }  
  
 gNew := grad(xNew)  
 beta := 0.0  
 if k > 0 {  
 beta = (gNew[0]\*gNew[0] + gNew[1]\*gNew[1]) / (g[0]\*g[0] + g[1]\*g[1])  
 }  
 d = add(mulScalar(gNew, -1), mulScalar(d, beta))  
 x = xNew  
 g = gNew  
 k++  
 }  
  
 fmt.Println("Метод Флетчера-Ривза:")  
 fmt.Printf("x\* = (%.6f, %.6f), f = %.6f, итераций = %d, останов = %s\n", x[0], x[1], f(x), k, cond)  
}  
  
// Простейший одномерный поиск: минимизация по t методом проб  
func minimizePhi(phi func(float64) float64) float64 {  
 bestT := 0.0  
 minVal := phi(0.0)  
 for t := 0.0; t <= 1.0; t += 0.01 {  
 val := phi(t)  
 if val < minVal {  
 minVal = val  
 bestT = t  
 }  
 }  
 return bestT  
}

Вывод программы



Анализ методов оптимизации

Метод наискорейшего спуска

Результат: x∗=(−0.500,−0.050), f(x∗)=−0.2625, итерации = 3.

Условие остановки: Изменения в x и f стали меньше заданных порогов.

Вывод: Быстрая сходимость (3 шага) указывает на хорошую обусловленность функции или удачный выбор параметров. Однако точность ниже, чем у методов Ньютона, что характерно для градиентных методов.

Метод Ньютона

Результат: x∗=(−0.526316,−0.052632), f=−0.263158, итерации = 1.

Условие остановки: ∣∣∇f∣∣<ϵ∣∣∇f∣∣<ϵ.

Вывод: Квадратичная сходимость подтверждается — минимум достигнут за 1 шаг. Начальное приближение, видимо, близко к решению, а гессиан вычислен корректно.

Метод Ньютона-Рафсона

Результат: x∗=(−0.526316,−0.052632), f=−0.263158, итерации = 2.

Условие остановки: ∣∣∇f∣∣<ϵ.

Вывод: Сходимость за 2 шага близка к методу Ньютона, что логично при схожей логике.

#### **Метод Флетчера-Ривза**

Эффективность: Метод достиг минимума за 3 итерации, что лишь немного уступает методу Ньютона (1 шаг) и Ньютона-Рафсона.

Точность: Значение f(x∗)f(x∗) совпадает с результатом метода Ньютона (−0.263158−0.263158), что подтверждает высокую точность.

**Общие выводы:**

Точность: Ньютон ≈ Ньютона-Рафсон ≈ Флетчер-Ривз > Наискорейший спуск.

Устойчивость: Наискорейший спуск и Флетчер-Ривз устойчивее к плохим начальным точкам, чем Ньютон.

Вычислительные затраты: Ньютон самый затратный (гессиан), Флетчер-Ривз – компромисс, градиентный – самый дешёвый.

**Оптимальный выбор:**

Если функция гладкая и гессиан легко вычисляется – Ньютон.

Если вычисление гессиана сложно – Флетчер-Ривз.

Для простых задач или при ограниченных ресурсах – градиентный спуск.