▸ GRBackend()

```
# импорты библиотек
begin
using Plots;
using PlutoUI;
using LaTeXStrings;
## настройка плотс
gr()   # set the backend to GR
end
```

Автоматическое дифференцирование

Мнимые числа

мнимое число задается выражением $a + bi$

допустим мы хотим посчитать функцию $f(x) = x^3$ для комплексного числа

$$f(a+i) = (a+i)^3 = a^3 + 3ai^2 + 3a^2i + i^3$$

мы можем записать

$$f(x+ih) = f(x) + f'(x)ih - \frac{1}{2}f''(x)h^2 + \mathcal{O}(h^2)$$

Мотивация! Проблемы чисел с плавающей запятой

```
begin
    println("Float32 eps is ",eps(Float32))
    println("Float64 eps is ",eps(Float64))

end
```

```
Float32 eps is 1.1920929e-7
Float64 eps is 2.220446049250313e-16
```

0.001953125

```
eps(10000000000000.)
```

999.9990001

```
@bind sl Slider(0.0000001:0.001:1000, show_value=true)
```

1.1368683772161603e-13

```
eps(sl)
```

что там с ассоциативностью? $(a+b)+c = a+(b+c)$

1.9999999999999998

```
(2+eps())-eps()
```

2.0

```
2+(eps()-eps())
```

```
begin
    e=1e-10;
    println(e);
    println(1+e);
    e2 = (1+e)-1;
    println(e2);
    println(e-e2)
end
```

```
1.0e-10
1.0000000001
1.000000082740371e-10
-8.274037096265818e-18
```

Обычное численное дифференцирование

формула для нахождения производной выглядит следующим образом:

$$f'(x_0) = lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{dx}$$

Легко это переписать эту формулу для любого языка программирования

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h}$$

Немного по-другому можно переписать последнюю формулу следующим образом

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h}$$

Последняя формула считается лучше, так как результат будет точнее.

Для примера посмотрим какая ошибка будет набегать в зависимости от шага для каждого из двух методов расчета производной

Пусть у нас есть функция $f(x) = x^3$. Её производная $f'(x) = 3x^2$

```
f′ₙ₁ (generic function with 1 method)
```

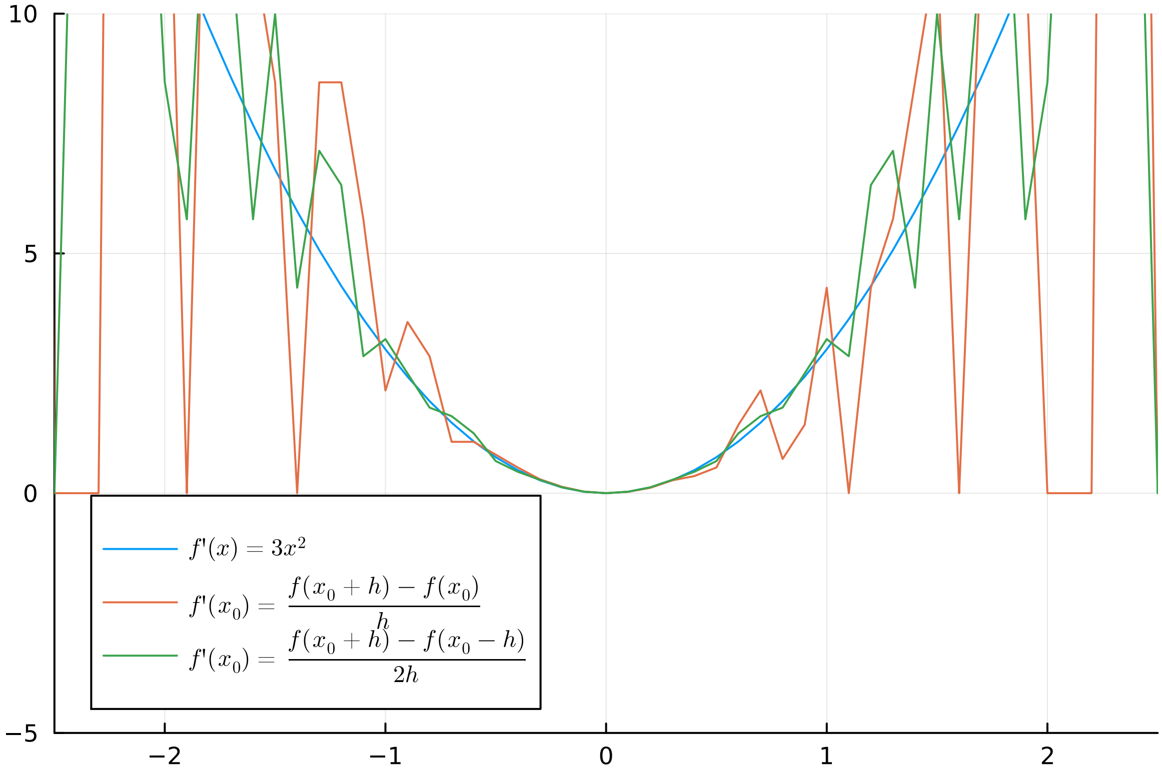
```
begin
    f₁(x) = x^3; # Функция
    f′(x) = 3x^2; # Производная (истинная)
    f′ₙ(f,x,h) = @.(f(x+h)-f(x))/h; # Численно вычисленная производная
    f′ₙ₁(f,x,h) = @.(f(x+h)-f(x-h))/2h; # Численно вычисленная производная центральной разностью
end
```

Посмотрим как меняется значение численно вычисленных производных, если мы будем менять шаг h

Для наглядности шаг сделаем равным $\exp(-p)$ **(будем изменять показатель степени p)**

p = 36.4

h = 1.554822650349588e-16



Дуальные числа

Хотелось бы избавиться от ошибок при вычислении производной из-за выбора шага. В этом нам помогут **дуальные числа**

Дуальные числа можно рассматривать как аналог комплексных чисел. Записывается дуальное число следующим образом:

$$a + b\varepsilon$$

здесь a - реальная часть, $b\varepsilon$ - дуальная часть, причем $\varepsilon^2 = 0$, но $\varepsilon \neq 0$

Арифметика дуальных чисел

сложение

$$(a + b\varepsilon) + (c + d\varepsilon) = (a + c) + (b\varepsilon + d\varepsilon)$$

вычитание

$$(a+b\varepsilon)-(c+d\varepsilon)=(a-c)+(b\varepsilon-d\varepsilon)$$

умножение

$$(a+b\varepsilon)\cdot(c+d\varepsilon)=ac+(bc+ad)\varepsilon$$

деление

$$\frac{a+b\varepsilon}{c+d\varepsilon}=\frac{a}{c}+\frac{bc-ad}{c^2}\varepsilon$$

Как же они нам помогут?

давайте вычислим значение нашей функции, используя вместо обычных чисел дуальные наша функция:

$$f(x)=x^3$$

Подставим дуальное число

$$f(a+b\varepsilon)=(a+b\varepsilon)^3=a^3+3a^2(b\varepsilon)+3a(b\varepsilon)^2+(b\varepsilon)^3=a^3+3a^2(b\varepsilon)$$

Перепишем короче

$$f(a+b\varepsilon)=a^3+3a^2(b\varepsilon)$$

Здесь тоже выделяется реальная часть и дуальная.

Можно заметить, что реальная часть соответствует просто нашей функции x^3, а дуальная часть $3a^2$ соотствует производной от этой функции

Если мы скажем программе правила для арифметики данных чисел, то мы сможем при вычислении функций сразу получать пару чисел: значение функции и производную

Определим дуальное число как пару чисел (реальную часть и дуальную)

```julia
struct Dual{T} # Мы просто создаем пару чисел
    val::T   # значение
    der::T  # производная
end
```

```julia
begin
    ## Определяем операции сложения
    Base.:+(f::Dual, g::Dual) = Dual(f.val + g.val, f.der + g.der)
    Base.:+(f::Dual, α::Number) = Dual(f.val + α, f.der)
    Base.:+(α::Number, f::Dual) = f + α

    ## Операция вычитания

    Base.:-(f::Dual, g::Dual) = Dual(f.val - g.val, f.der - g.der)

    # Операция умножения
    Base.:*(f::Dual, g::Dual) = Dual(f.val*g.val, f.der*g.val + f.val*g.der)
    Base.:*(α::Number, f::Dual) = Dual(f.val * α, f.der * α)
    Base.:*(f::Dual, α::Number) = α * f

    # Деление
    Base.:/(f::Dual, g::Dual) = Dual(f.val/g.val, (f.der*g.val - f.val*g.der)/(g.val^2))
    Base.:/(α::Number, f::Dual) = Dual(α/f.val, -α*f.der/f.val^2)
    Base.:/(f::Dual, α::Number) = f * inv(α) # Dual(f.val/α, f.der * (1/α))

    ## Степень
    Base.:^(f::Dual, n::Integer) = Base.power_by_squaring(f, n)  # use repeated squaring for integer powers
end
```

Теперь мы можем совершать базовые арифметические операции над дуальными числами

```
6
```

```julia
begin
    fd = Dual(3, 1)
    println(fd*fd) # x^2
    (fd*fd).der ## x^2 -> 2x | при x=3 получим 2*3 =6
end
```

```
Main.var"workspace#5".Dual{Int64}(9, 6)
```

Как быть со всякими синусами косинусами и прочими функциями?

есть два пути:

- залезть глубоко в язык и переопределить базовые функции
- захардкодить известные функции

```
sin (generic function with 21 methods)
```

```julia
begin
    import Base: exp
    exp(f::Dual) = Dual(exp(f.val), exp(f.val) * f.der) # производная от экспоненты это тоже экспонента (f.der в конце нужно добавить, чтобы у нас был chain rule)

    import Base: sin
    sin(f::Dual) = Dual(sin(f.val), cos(f.val)*f.der) # производная от синуса косинус
end
```

Попробуем реализовать алгоритм для нахождения корня

$$x_{i+1} = \frac{1}{2}(x_n + \frac{2}{x_n})$$

$$f\prime(\sqrt{x}) = \frac{1}{2\sqrt{x}}$$

$$\sqrt{(2)} = 1.414\ldots$$

$$f\prime(\sqrt{2}) = 0.3535\ldots$$

```
▶ Dual(1.414213562373095, 0.35355339059327373)
```

```
begin
    function newtons(x)
        a = x
        for i in 1:300
            a = 0.5 * (a + x/a)
        end
        a
    end
    @show newtons(2.0)
    @show (newtons(2.0+sqrt(eps())) - newtons(2.0))/ sqrt(eps())
    @show newtons(Dual(2.0,1.0))
end
```

```
newtons(2.0) = 1.414213562373095
(newtons(2.0 + sqrt(eps())) - newtons(2.0)) / sqrt(eps()) = 0.3535533994436264
newtons(Dual(2.0, 1.0)) = Main.var"workspace#5".Dual{Float64}(1.414213562373095, 0.35355339059327373)
```

Более высокие размерности

Перейдем к размерности повыше

Рассмотрим функцию типа $f : \mathbb{R}^2 \to \mathbb{R}$

То есть это функции которые на вход берут 2 аргумента, а на выходе отдают один

Например, рассмотрим функцию $f(x, y) = x^2 + xy$

```
fquad (generic function with 1 method)
```

```
fquad(x, y) = x^2 + x*y
```

Посчитаем частные производные с помощью дуальных чисел

$$f(x, y) = x^2 + xy$$

$$\frac{\partial f}{\partial x} = 2x + y$$

$$\frac{\partial f}{\partial y} = x$$

```
derivative (generic function with 1 method)
```

```
# Сделаем хитрость №1
derivative(f, x) = f(Dual(x, one(x))).der ## так будет удобнее сразу выделять производные
```

```
10.0
```

```
begin
    ## Сделаем хитрость №2
    a, b = 3.0, 4.0;
    fquad_1(x) = fquad(x, b);
    derivative(fquad_1, a) ## ∂f/∂x  = 2x+y = 2×3+4 = 10 #(a+e)^2+(a+e)*b
end
```

```
▸Dual(21.0, 10.0)
```

```
# На самом деле мы сделали это
fquad(Dual(a, one(a)), b)
```

```
3.0
```

```
begin
    # мы можем сделать тоже самое с другой переменной (y)
    fquad_2(y) = fquad(a, y)
    derivative(fquad_2, b) #∂f/∂y  = x = 3
end
```

Запишем наши производные в столбик и получим

$$\begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} = \nabla f$$

Необходимо сделать n вычислений для нахождения градиента

Высокоразмерные дуальные числа

То как я это представляю

допустим $x \to a + b\varepsilon$

Если мы хотим вычислять градиент от функции двух переменных, то можно переписать следующим образом

$$x \to x + \bar{\varepsilon} \to a + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \end{bmatrix}$$

А еще можно записать вот так

$$x \to a + \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$y \to b + \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Также $\varepsilon^2 = \varepsilon_i \varepsilon_j = 0$

Функции от мультидуальных чисел будут давать следующую вещь

$$f(a + \varepsilon) = f(a) + \nabla f(a)\varepsilon$$

только теперь $a \in \mathbb{R}^m$, а дуальная часть дает нам градиент скалярной функции

Для создания *мультидуальных чисел* будем использовать библиотеку `StaticArrays`

```
using StaticArrays
```

```
## определим структуру
struct MultiDual{N,T}
    val::T
    derivs::SVector{N,T}
end
```

```
* (generic function with 349 methods)
```

```
begin
    import Base: +, *
function +(f::MultiDual{N,T}, g::MultiDual{N,T}) where {N,T}
    return MultiDual{N,T}(f.val + g.val, f.derivs + g.derivs)
end

function *(f::MultiDual{N,T}, g::MultiDual{N,T}) where {N,T}
    return MultiDual{N,T}(f.val * g.val, f.val .* g.derivs + g.val .* f.derivs)
end
end
```

Рассмотрим пример. У нас есть функция $g(x, y) = x^2 y + x + y$

$$\nabla g(x, y) = \begin{pmatrix} \frac{\partial g}{\partial x} \\ \frac{\partial g}{\partial y} \end{pmatrix} = \begin{pmatrix} 2xy + 1 \\ x^2 + 1 \end{pmatrix}$$

допустим, что x = 1, y = 2. Тогда:

$$\nabla g(x = 1, y = 2) = \begin{pmatrix} 5 \\ 2 \end{pmatrix}$$

```
gcubic (generic function with 1 method)
```

```
gcubic(x, y) = x*x*y + x + y
```

```
▾MultiDual{2, Float64}(
    val = 5.0
    derivs = ▸StaticArraysCore.SVector{2, Float64}: [5.0, 2.0]
)
```

```julia
begin
    (aa, bb) = (1.0, 2.0)
    xx = MultiDual(aa, SVector(1.0, 0.0))
    yy = MultiDual(bb, SVector(0.0, 1.0))
    gcubic(xx, yy)
end
```

Также мы можем вычислить Якобиан J функции $\mathbb{R}^n \to \mathbb{R}^m$

$$\mathbf{f}(\mathbf{x}, \mathbf{y}) = (x^2 + y^2, x + y) = \begin{pmatrix} x^2 + y^2 \\ x + y \end{pmatrix}$$

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{pmatrix} = \begin{pmatrix} \nabla f_1(x, y) \\ \nabla f_2(x, y) \end{pmatrix}$$

```julia
begin
    fsvec(x, y) = SVector(x*x + y*y , x + y)

    println(fsvec(xx, yy)[1].derivs)
    println(fsvec(xx, yy)[2].derivs)
end
```

```
[2.0, 4.0]
[1.0, 1.0]
```

Автоматическое дифференцирование реализовано в некоторых библиотеках. Как пример можно привести библиотеку `ForwardDiff`

```julia
using ForwardDiff
```

```
▸[6, 2]
```

```julia
ForwardDiff.gradient( xx -> ( (x, y) = xx; x^2 * y + x*y ), [1, 2]) #Первый аргумент здесь это функция (в нашем случае анонимная), а второй аргумент это наш вектор
```

Производная по направлению и градиент функции $f : \mathbb{R}^n \to \mathbb{R}$

Производную по направлению можно записать следующим образом

$$\lim_{\epsilon \to 0} \frac{f(\mathbf{x} - \epsilon \mathbf{v}) - f(\mathbf{x})}{\epsilon} = [\nabla f(\mathbf{x})] \cdot \mathbf{v}$$

В нашем случае, вектор задаем мы сами, когда создаем дуальный вектор

```
MultiDual(a,[v_1,v_2])
```

$$f(\mathbf{a}+\mathbf{v}) = a + \nabla\mathbf{f}(\mathbf{a_1})v_1 + \nabla\mathbf{f}(\mathbf{a_1})v_2 + \dots$$

Можно посмотреть по-другому

наш дуальный вектор

$$d = d_0 + e_1\epsilon_1 + e_2\epsilon_2 + e_3\epsilon_3 + \dots$$

$$f(d) = f(d_0) + Je_1\epsilon_1 + Je_2\epsilon_2 + Je_3\epsilon_3 + \dots$$

Массивы дуальных чисел

$$D_0 = [d_1, d_2, d_3, \dots, d_n]$$

$$\Sigma = \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1n} \\ d_{21} & d_{22} & & \vdots \\ \vdots & & \ddots & \vdots \\ d_{m1} & \cdots & \cdots & d_{mn} \end{bmatrix}$$

$$\epsilon = [\epsilon_1, \epsilon_2, \dots, \epsilon_m]$$

$$D = D_0 + \Sigma\epsilon$$

допустим у нас есть функция $f(x) = Ax$ (матричное умножение)

тогда

$$f(D) = f(D_0) + f'(D_0)\Sigma\epsilon$$

где $f'(D_0)$ - полный якобиан

Вторые производные

Подход с дуальными числами может быть расширен на производные более высоких порядков (например вторые производные)

$$f(a+\varepsilon) = f(a) + \varepsilon f\prime(a) + \frac{1}{2}\varepsilon^2 f\prime\prime(a) + \dots$$

Для примера для функции $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}$ получим

$$f(x + \varepsilon v) = f(x) + \varepsilon[\Sigma_i(\partial_i f)(x)v_i] + \frac{1}{2}\varepsilon^2[\Sigma_i \Sigma_j(\partial_{i,j}v_i v_j)]$$

Таким образом можно вычислить коэффициенты матрицы Гесса

```
#ForwardDiff.
```