Поиск оптимальных условий

Исследовать поверхность отклика — это значит с помощью совокупности математических методов и на основе экспериментальных данных оптимальным образом подобрать адекватную модель исследуемого процесса и выяснить влияние отклика $\mathrm{y} = f(x_1, \mathrm{x}_2, \ldots, x_n)$, где $x_1, \mathrm{x}_2, \ldots, x_n$ — независимые переменные.

Предполагается, что эти переменные непрерывны и могут контролироваться экспериментатором с достаточно малой ошибкой, а отклик — величина случайная.

Пусть, например, необхдимо найти значения температуры (x_1) и давления (x_2), которые максимизируют выход исследуемого им процесса. Можно записать наблюдаемый отклик в виде функции $\mathrm{y} = f(x_1, \mathrm{x}_2) + \varepsilon$, где ε — случайная величина (ошибка).

Если обозначить математическое ожидание отклика $\mathrm{M}(\mathrm{y})$ как η, то поверхность, представляемая $\eta = f(\mathrm{x}_1, \mathrm{x}_2)$, называется - **поверхностью отклика**.

Отклик должен соответствовать следующим требованиям:

- параметр должен измеряться при любом изменении (комбинации) предикторов;
- параметр должен быть статистически эффективным, то есть измеряться с наибольшей точностью;
- параметр должен быть информационным, то есть всесторонне характеризовать процесс;
- параметр должен быть однозначным, то есть должна минимизироваться или максимизироваться только одна целевая функция.

К факторам обычно предъявляются следующие требования:

- Фактор **должен** оказывать влияние на параметры оптимизации;
- Фактор не должен быть коррелирован с другими факторами;
- Фактор должен быть количественным (но в целом не исключено изучение и качественных факторов);
- Совместимость факторов – при всех сочетаниях их уровней эксперимент можно поставить и он будет безопасным;
- Операциональность – экспериментатору должно быть известно, как, где, каким прибором и с какой точностью контролировать уровень фактора;
- Управляемость – экспериментатор должен иметь возможность устанавливать значение уровня фактора по своему усмотрению;

- Точность установления уровня фактора должна быть существенно, по крайней мере на порядок, выше точности определения параметра;
- Однозначность воздействия фактора на объект исследования.

Изучение поверхности отклика состоит из нескольких этапов.

1. Поиск подходящей аппроксимации.

В большинстве задач по изучению поверхности отклика вид зависимости между откликом и предикторами неизвестен, поэтому на первом этапе нужно найти подходящую аппроксимацию для f. Обычно на первых этапах используется полиномы невысокого порядка. Если отклик хорошо описывается линейной функцией независимых переменных, то аппроксимирующей функцией является модель первого порядка (линейная модель).

Если в системе заметно отклонение от линейности, то нужно использовать полиномы более высокого порядка.

Необходимо помнить - маловероятно, что какая-либо полиномиальная модель окажется разумной аппроксимацией f на всем пространстве области определения, но в относительно малых областях они достаточно хороши.

2. Оценивание коэффициентов первичной модели

Для оценивания коэффициентов аппроксимирующих полиномов применяется МНК (метод наименьших квадратов). Затем проводится анализ в терминах подобранной поверхности. Если эта поверхность является адекватной аппроксимацией f, то анализ подобранной поверхности будет приблизительно эквивалентен анализу реальной системы. Параметры модели оцениваются наиболее эффективно, если для получения данных используются оптимальные планы (ортогональные планы, центральный композиционный план и т.д.).

3. Определение кривизны поверхности и дополнение модели

Когда область оптимума определена, то при необходимости (нелинейная поверхность) можно применить более сложную модель, например поверхность отклика второго порядка, и провести анализ для локализации оптимума. Анализ поверхности отклика можно представлять себе как «подъем на холм», вершина которого является точкой максимального отклика. Если истинный оптимум — точка минимума отклика, то можно представить себе «спуск в овраг».

4. Пошаговое приближение к оптимуму

Во многих случаях первоначальная оценка оптимальных рабочих условий для системы оказывается далекой от действительного оптимума. В такой ситуации цель исследователя — быстро перейти в окрестность оптимума, применяя при этом простую и экономически эффективную процедуру экспериментирования. Когда мы находимся далеко от оптимума, то обычно полагаем, что в небольшой области

изменения переменных модель первого порядка адекватно аппроксимирует истинную поверхность.

Метод крутого восхождения — процедура последовательного перемещения по пути крутого восхождения, т.е. в направлении наибольшего увеличения отклика. Если необходима минимизация, то тогда следует применять метод крутого спуска.

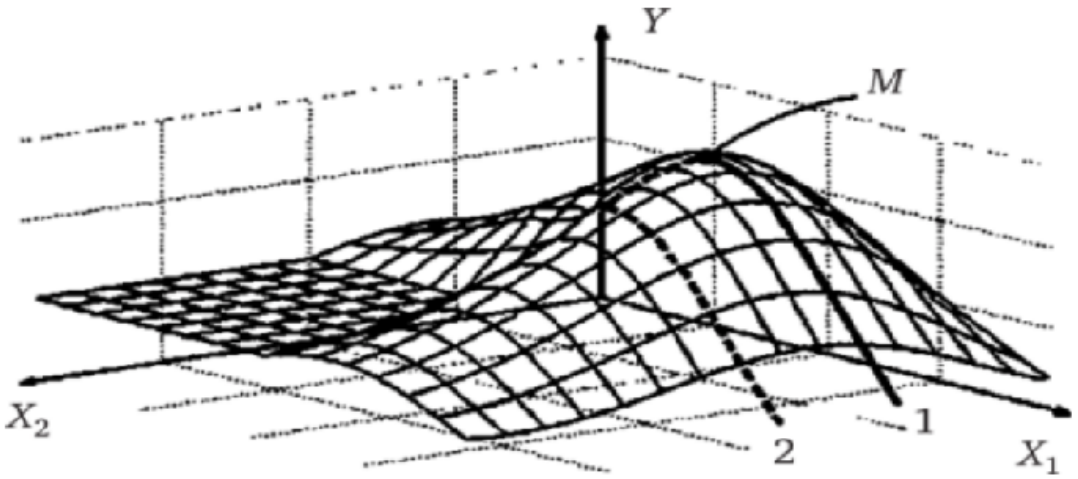
Обычно в качестве пути крутого восхождения выбирается линия, проходящая через центр области экспериментирования и нормальная к контурам подобранной поверхности. Фактическая длина шага определяется экспериментатором на основе опыта. Таким образом, шаги вдоль этого пути пропорциональны коэффициентам регрессии.

Эксперименты проводятся по линии крутого восхождения до тех пор, пока не перестанет наблюдаться увеличение отклика. Затем для описания отклика можно подобрать новую модель первого порядка и найти новую линию крутого восхождения. Продолжая процедуру таким образом, экспериментатор попадет в окрестность оптимума; об этом обычно свидетельствует неадекватность модели первого порядка. В таком случае для получения более точной оценки положения оптимума проводятся дополнительные эксперименты.

Когда исследователь относительно близок к оптимуму, то из-за кривизны истинной поверхности для описания отклика обычно требуется модель второго или более высокого порядка. В большинстве случаев адекватной аппроксимацией оказывается модель второго порядка.

Поиск оптимальных параметров может выполняться:

- Пошагово в ходе получения экспериментальных данных.
- В ходе оптимизации математической модели функции отклика.



Классификация алгоритмов поиска оптимальных значений функции:

- Градиент независимые
  - Прямые методы
    - Цикличсеский метод (Гаусса-Зейделя)
    - Метод Пауэлла
    - Симплекс-метод (Метод Нелдера—Мида)
  - Стохастические методы
    - Алгоритм имитации отжига (Simulated annealing)
  - Популяционные методы
    - Метод роя частиц (particle swarm optimization)
    - Генетические алгоритмы
    - Алгоритм светлячка
- Градиент зависимые методы
  - Метод сопряженных градиентов (Conjugate gradient method)
  - Градиентный спуск (Gradient Descent)
  - Алгоритм Бройдена—Флетчера—Гольдфарба—Шанно (BFGS / Limited-memory BFGS)
  - Другие (N-GMRES, O-ACCEL и т.д.)
- Гессиан-зависимые
  - Метод Ньютона
  - Метод Ньютона в доверенной области
- Комбинированные методы
  - Зашумленный градиент

Применение методов оптимизации:

- Прямая оптимизация функции отклика в ходе эесперимента
- Оптимизация для поиска наиболее вероятных параметров модели (ФК/ФД

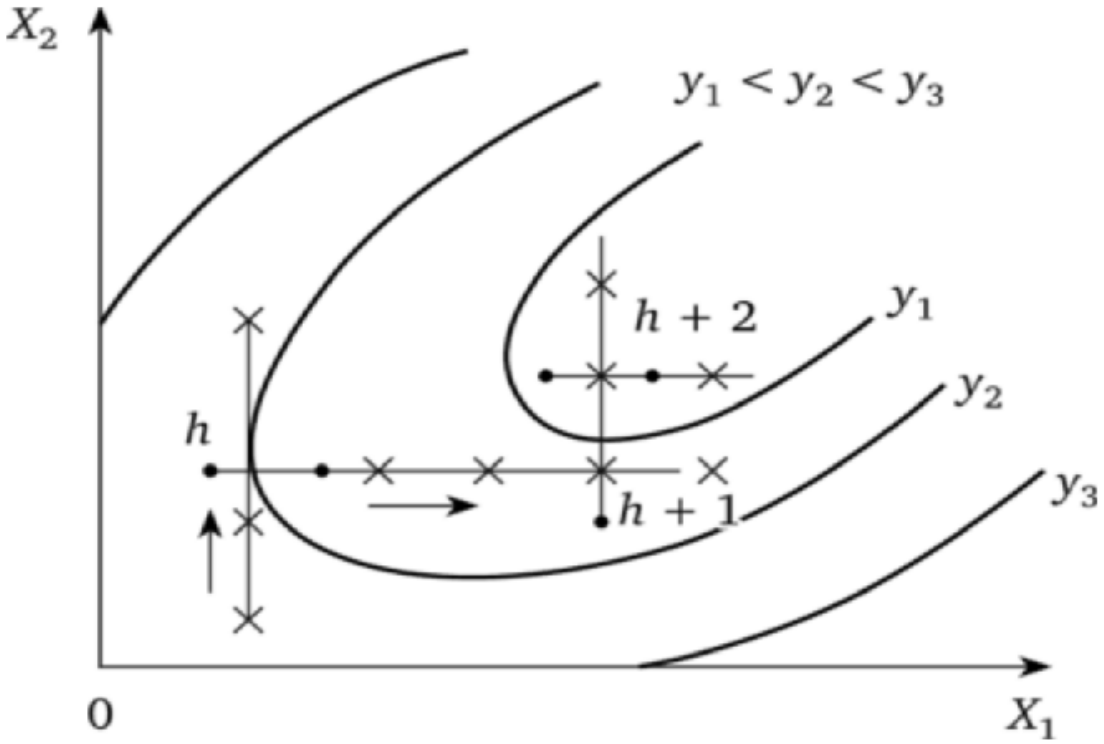
моделирования, оптимизация функции правдоподобия)
- Поиск оптимальных значений модели с уже подобранными параметрами

Метод Гаусса — Зейделя

Метод Гаусса — Зейделя — прямой метод решения задач многомерной оптимизации.

Суть метода заключается в том, чтобы на каждой итерации по очереди минимизировать функцию варьируя один из факторов, все остальные остаются фиксированными.

Достоинствами метода Гаусса — Зейделя являются простота, наглядность, отсутствие каких-либо вычислений, высокая помехозащищенность в смысле выбора направления движения. К недостаткам относится необходимость большого числа опытов. Их число резко увеличивается с ростом количества входных факторов. При этом трудно стабилизировать все управляемые факторы, кроме одного, на длительное время, что может вызвать погрешности в нахождении частных экстремумов.



Симплекс-метод

Метод многофакторного планирования эксперимента предусматривает многошаговый процесс движения к экстремуму по поверхности отклика с одновременным, если это необходимо, описанием многофакторной зависимости соответствующей части этой поверхности.

Название метода объясняется тем, что начальная серия опытов и все их совокупности, анализируемые на том или ином шаге движения к экстремуму,

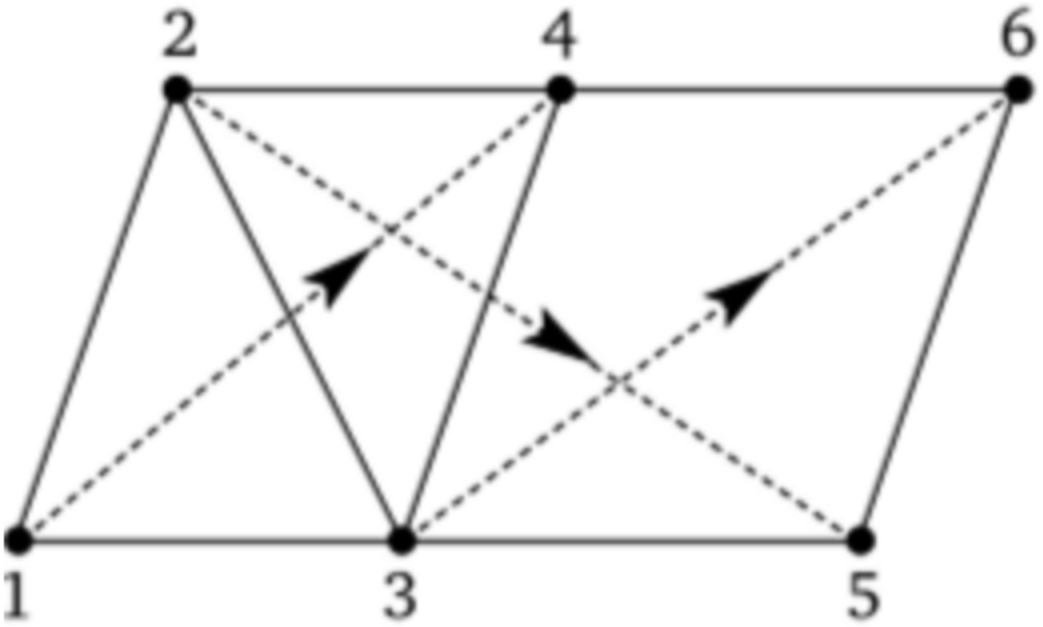
расположены в n-мерном факторном пространстве n-мерного симплекса.

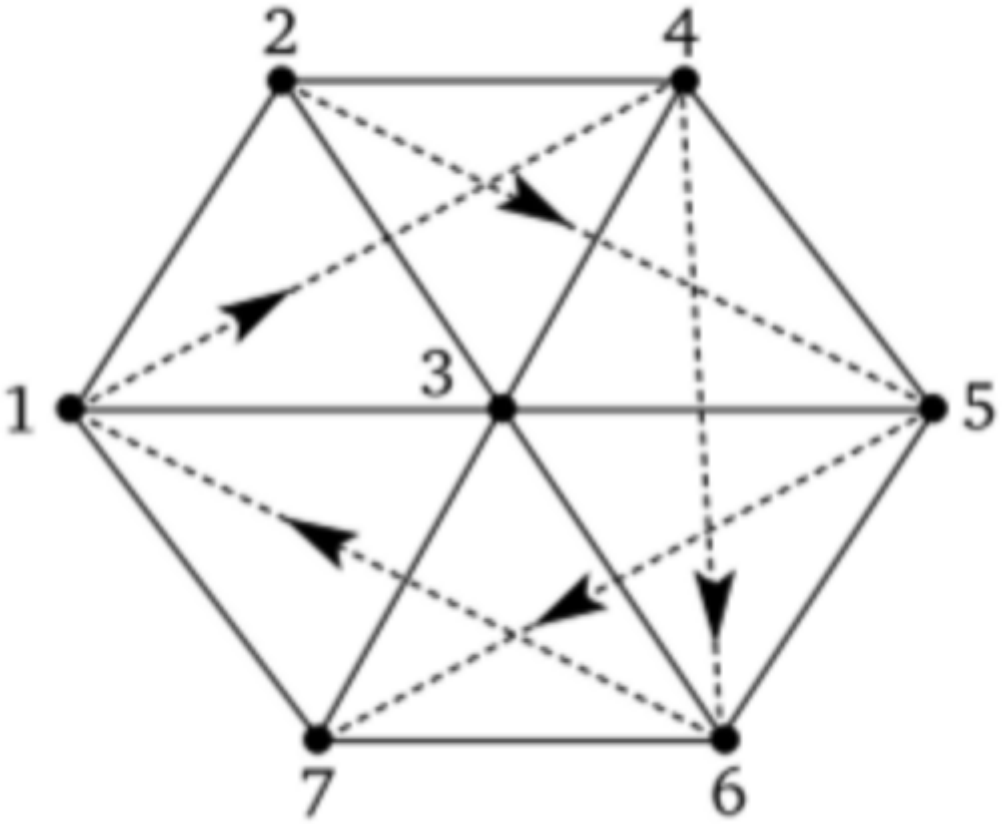
Под n-мерным симплексом подразумевают выпуклую геометрическую фигуру в n -мерном факторном пространстве, имеющую $n+1$ вершин, соединенных прямыми отрезками, называемыми рёбрами. Если вершины находятся друг от друга на одинаковом расстоянии, то такой симплекс называют правильным. Любые n вершин симплекса лежат в одной гиперплоскости. Часть такой гиперплоскости, ограниченной ребрами, будет называться гранью симплекса, противоположной вершине, не лежащей в этой гиперплоскости.

Одномерным симплексом будет отрезок прямой, двумерным — плоский треугольник, трехмерным — тетраэдр и т.д.

Описание алгоритма:

1. Вычисляется симплекс и реализуются эксперименты (число экспериментов для n-мерного симплекса равняется $n+1$);
2. Отбрасывается точка плана с наименьшим значением выходного параметра;
3. Строится новый симплекс, образуемый остальными вершинами выходного симплекса и новой вершиной, которую получают путем зеркального отражения отброшенной вершины относительно противоположной к ней грани начального симплекса;
4. Последовательно перемещается симплекс, в процессе чего на каждом шаге происходит отбрасывание вершины с наихудшим значением выходной переменной;
5. Если при перемещении симплекса на протяжении $n+1$ шагов одна из вершин сохраняет свое место, то симплекс делает оборот вокруг этой вершины. Это означает, что в данной точке находится оптимум;
6. Если выходная переменная в новой вершине симплекса выглядит хуже, чем в других вершинах, возвращаются к предыдущему симплексу и выбирают вершину, которую откинули и в которой выходная переменная имеет значение, следующее по порядку за наихудшей вершиной симплекса;
7. При достижении оптимума размер симплекса уменьшают.





Алгоритмы оптимизации применяются как в случае поиска отимальных условий в ходе эксперимента, так и для поиска оптимальных параметров функции.

Градиент независимые методы могут применяться при любых функциональных зависимостях, но сходимость таких методов, как правило хуже чем, градиент зависимых. Градиент независимые методы могут применяться когда

функциональная зависимость вообще не задана.

Градиент и гессиан зависимые методы применяются когда оптимизируются зависимости выраженные через явные дифференцируемые функции.

Стохастические, популционные и комбинированные методы применяются в случае если существуют локальные минимумы/максимумы.

Метод градиентного спуска

Частные производные

Допустим есть функция нескольких переменных: $z = 2x^2y^3 + 3x^4 + 5y - 7$

По своей сути частные производные 1-го порядка напоминают «обычную» производную: $z_x^{'}$, $z_y^{'}$ – это функции, которые характеризуют скорость изменения функции $z = f(x, y)$ в направлении осей OX и OY соответственно.

Так, например:

- функция $z_x^{'} = 4xy^3 + 12x^3$ характеризует крутизну «подъёмов» и «склонов» поверхности z в направлении оси абсцисс
- функция $z_y^{'} = 6x^2y^2 + 5$ сообщает нам о «рельефе» этой же поверхности в направлении оси ординат.

Производная по направлению

Если производная по $y^{'}$ - это скорость изменения функции $y = f(x)$ в направлении оси OX, то есть грубо говоря "по горизонтали", то при данной нам функции $z = f(x, y)$ мы можем охарактеризовать ее еще по оси OY.

Чтобы узнать скорость изменения функции $z = f(x, y)$ в каком либо направлении, мы будем пользоваться производной по направлению $\frac{\partial z}{\partial l}$. В качестве направления выступает вектор l.

Представим себе точку $M_0(x_0; y_0)$ из области определения функции. Из этой точки мы можем двигаться в сторону OX и OY, и направление нашего движения будет обозначаться лучом l, исходящим из точки M_0. Сам луч можно опрелелить с помощью угла между ним и любой осью, либо с помощью вектора.

Определение:

Производной $\frac{\partial f}{\partial l}$ функции f в точке M_0 по направлению вектора l называется предел отношения $\frac{f(M) - f(M_0)}{|l|}$, при $|l| \to 0$.

Градиент

Градиентом функции нескольких переменных f в точке M называется вектор:

$$grad f(M) = \nabla f(M) = (f'_{x_1}(M),\ f'_{x_2}(M),\ \dots\ ,\ f'_{x_n}(M))$$

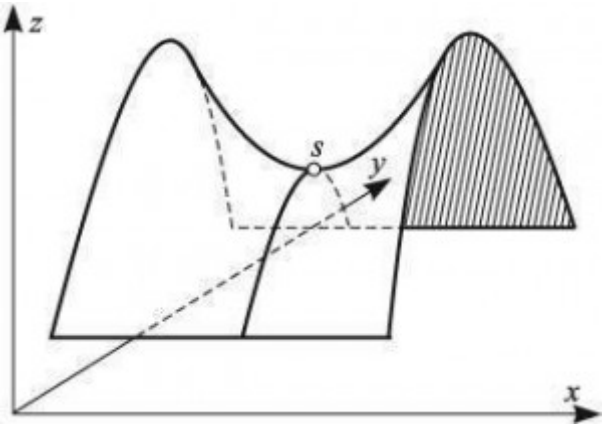
Вектор $\nabla f(M)$ указывает на направление наиболее быстрого роста функции в точке M, а его длина равна скорости этого роста.

Производная функции в направлении, перпендикулярном вектору $\nabla f(M)$, равна нулю.

В каждой точке градиент перпендикулярен линии (поверхности) уровня, проходящей через эту точку.

Стационарная (критическая) точка

Точка, в которой все частные производные первого порядка равны нулю ($\nabla f(M) = 0$), называется стационарной.



Вторая производная

Вторая производная простым языком, это производная первой производной. Так как для функции многих переменных количество частных производных равно количеству аргументов функции, то и вторые производные это также частные производные получившихся функций многих переменных.

К примеру:

$z = 2x^2y^3 + 3x^4 + 5y - 7$

Первые производные:

- $z'_x = \frac{\partial z}{\partial x} = 4xy^3 + 12x^3$
- $z'_y = \frac{\partial z}{\partial y} = 6x^2y^2 + 5$

Соответственно вторые производные:

- $\frac{\partial z}{\partial x} = 4xy^3 + 12x^3$
  - $\frac{\partial^2 z}{\partial x^2} = 4y^3 + 36x^2$
  - $\frac{\partial^2 z}{\partial x \partial y} = 12xy^2$

- $\frac{\partial z}{\partial y} = 6x^2y^2 + 5$
  - $\frac{\partial^2 z}{\partial y^2} = 12x^2y$
  - $\frac{\partial^2 z}{\partial y \partial x} = 12xy^2$

Производные $\frac{\partial z}{\partial y \partial x}$ и $\frac{\partial z}{\partial x \partial y}$ называются смешанные производные и равны друг другу.

Матрица Гессе

Матрица этой квадратичной формы образована вторыми частными производными функции. Если все производные существуют, то

$$\nabla^2 f = H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \, \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \, \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \, \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \, \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \, \partial x_1} & \frac{\partial^2 f}{\partial x_n \, \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Определитель этой матрицы называется определителем Гессе, или гессианом.

Достаточное условие экстремума

Пусть $|\nabla^2 f(M)|$ - определитель матрицы вторых производных (гессиан) в точке M.

Если $|\nabla^2 f(M)| > 0$, то функция f имеет экстремум в этой точке.

При этом для функции двух переменных:

Для функции двух переменных достаточно что бы первый диагональный элемент $\nabla^2 f(M)$ был > 0, тогда в этой точке достигается минимум. Если первый диагональный элемент $\nabla^2 f(M)$ < 0 - максимум.

Для функции многих переменных:

- если матрица $\nabla^2 f(M)$ по критерию Сильвестра положительно определенная, то в точке функция имеет локальный минимум;
- если матрица отрицательно определенная, то в точке функция имеет локальный максимум;
- если матрица неопределенная, то в точке функция не имеет локального экстремума (это – так называемая седловая точка).

Если $|\nabla^2 f(M)| < 0$, то функция f не имеет экстремума в этой точке (седловая точка).

Симметричная матрица отрицательно(положительно) определена если все собсвенные числа отрицательные(положительные).

Решение задач оптимизации методом градиентного спуска

Метод градиентного спуска:

$$X_{k+1} = X_k - t_k \nabla f(X_k)$$

k выбирается как (метод наискорейшего спуска):

$$k = argmin_k f(X_k - t_k \nabla f(X_k))$$

Пример:

$$f(x, y) = 3x^2 + y^2 - 2xy \to min$$

$x_0 = 4; y_0 = 2$

$$\frac{\partial f}{\partial x} = 6x - 2$$

$$\frac{\partial f}{\partial y} = 2y - 2$$

$\frac{\partial f}{\partial x} = 6 * 4 - 2 = 22$

$\frac{\partial f}{\partial y} = 2 * 2 - 2 = 2$

$$\nabla f(x_0, y_0) = [22, 2]$$

$$k\nabla f(x_0, y_0) = [22k, 2k]$$

$$[4, 2] - k\nabla f(x_0, y_0) = [4 - 22k, 2 - 2k]$$

$k = argmin_k f(X_k - t_k \nabla f(X_k)) = f(4 - 22k, 2 - 2k) = 3(4 - 22k)^2 + (2 - 2k)^2 - 2($

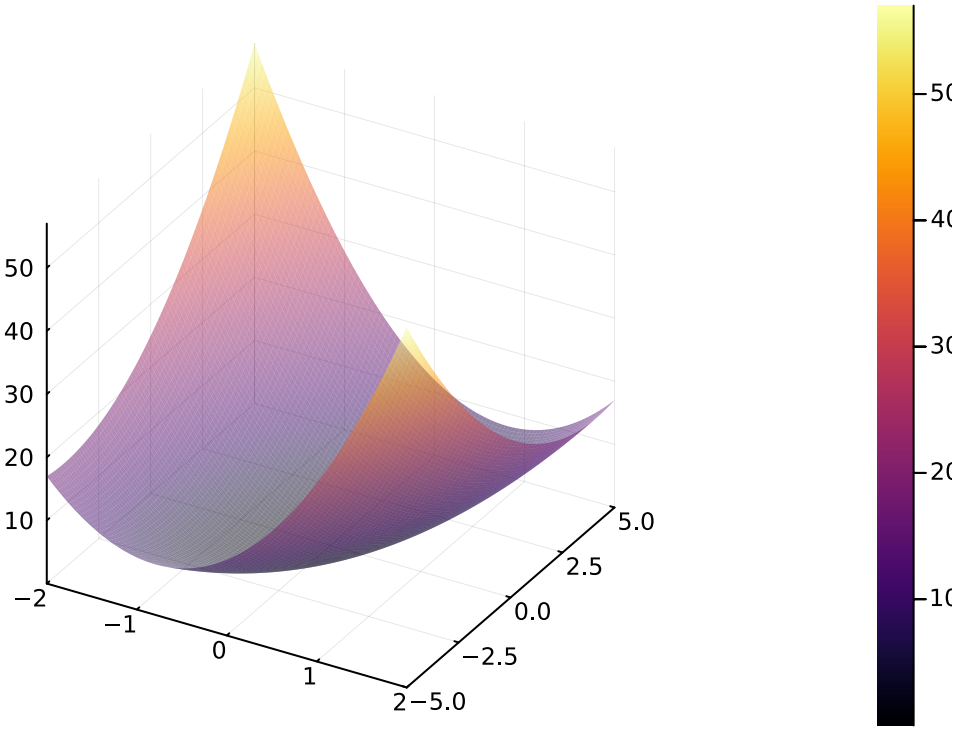
$f'(X_k - t_k \nabla f(X_k)) = 2736 \cdot k - 432 = 0$

$k = 432/2736 \approx 0.15789$

$$t_k \nabla f(X_k) = [3.47, 0.3157]$$

In [32]:
```
using Optim
using Plots

f(x) = 3x[1]^2 + x[2]^2 - 2x[1]*x[2]
fp(x, y) = f([x, y])

xs = LinRange(-2, 2, 100)
ys = LinRange(-5, 5, 100)
zs = [fp(x, y) for x in xs, y in ys]

surface(xs, ys, zs,  alpha = 0.5)
```

Out[32]:



In [33]:
```
using Optim
opt = optimize(f, [20.0, 10.0], BFGS())
```

Out[33]:
```
 * Status: success

 * Candidate solution
    Final objective value:     1.314109e-19

 * Found with
    Algorithm:     BFGS

 * Convergence measures
    |x - x'|               = 1.30e+01 ≰ 0.0e+00
    |x - x'|/|x'|          = 6.64e+10 ≰ 0.0e+00
    |f(x) - f(x')|         = 1.14e+02 ≰ 0.0e+00
    |f(x) - f(x')|/|f(x')| = 8.67e+20 ≰ 0.0e+00
    |g(x)|                 = 1.25e-09 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    2
    f(x) calls:    5
    ∇f(x) calls:   5
```

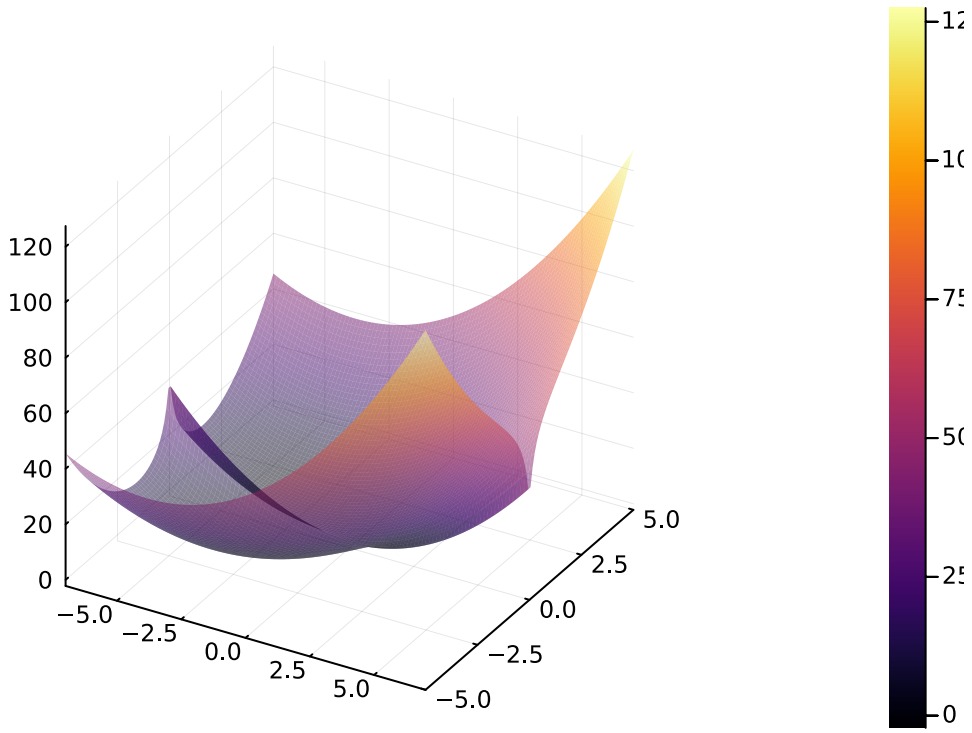
In [34]:
```
Optim.minimizer(opt)
```

Out[34]:
```
2-element Vector{Float64}:
 -1.9606360979196324e-10
  3.745093124507548e-11
```

In [35]:
```
using Optim
using Plots

f(x) =  x[1]^2 + 1.5x[2]^2 + 2x[2] + log(10x[1]^2 + 0.1)*x[2]
fp(x, y) = f([x, y])
```

```
xs = LinRange(-7, 7, 100)
ys = LinRange(-5, 5, 100)
zs = [fp(x, y) for x in xs, y in ys]

surface(xs, ys, zs,  alpha = 0.5)
```

Out[35]:



In [38]:
```
using Optim
opt = optimize(f, [20.0, 10.0], Newton())
```

Out[38]:
```
 * Status: success

 * Candidate solution
    Final objective value:     -2.206300e+00

 * Found with
    Algorithm:     Newton's Method

 * Convergence measures
    |x - x'|               = 3.06e-08 ≰ 0.0e+00
    |x - x'|/|x'|          = 1.92e-08 ≰ 0.0e+00
    |f(x) - f(x')|         = 1.33e-15 ≰ 0.0e+00
    |f(x) - f(x')|/|f(x')| = 6.04e-16 ≰ 0.0e+00
    |g(x)|                 = 0.00e+00 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    5
    f(x) calls:    13
    ∇f(x) calls:   13
    ∇²f(x) calls:  5
```

In [39]:
```
Optim.minimizer(opt)
```

Out[39]:
```
2-element Vector{Float64}:
 -1.256362758413877
 -1.5884473806906334
```

In [40]:
```
opt = optimize(f, [20.0, 10.0], NelderMead())
```

Out[40]:
```
 * Status: success

 * Candidate solution
    Final objective value:     -2.206300e+00

 * Found with
    Algorithm:     Nelder-Mead

 * Convergence measures
    √(Σ(yᵢ-ȳ)²)/n ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    48
    f(x) calls:    95
```

In [41]:
```
opt = optimize(f, [20.0, 10.0], LBFGS())
```

Out[41]:
```
 * Status: success

 * Candidate solution
    Final objective value:     -2.206300e+00

 * Found with
    Algorithm:     L-BFGS

 * Convergence measures
    |x - x'|               = 5.08e-07 ≰ 0.0e+00
    |x - x'|/|x'|          = 3.20e-07 ≰ 0.0e+00
    |f(x) - f(x')|         = 7.40e-13 ≰ 0.0e+00
    |f(x) - f(x')|/|f(x')| = 3.35e-13 ≰ 0.0e+00
    |g(x)|                 = 1.75e-10 ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    7
    f(x) calls:    19
    ∇f(x) calls:   19
```

In [42]:
```
opt = optimize(f, [20.0, 10.0], SimulatedAnnealing())
```

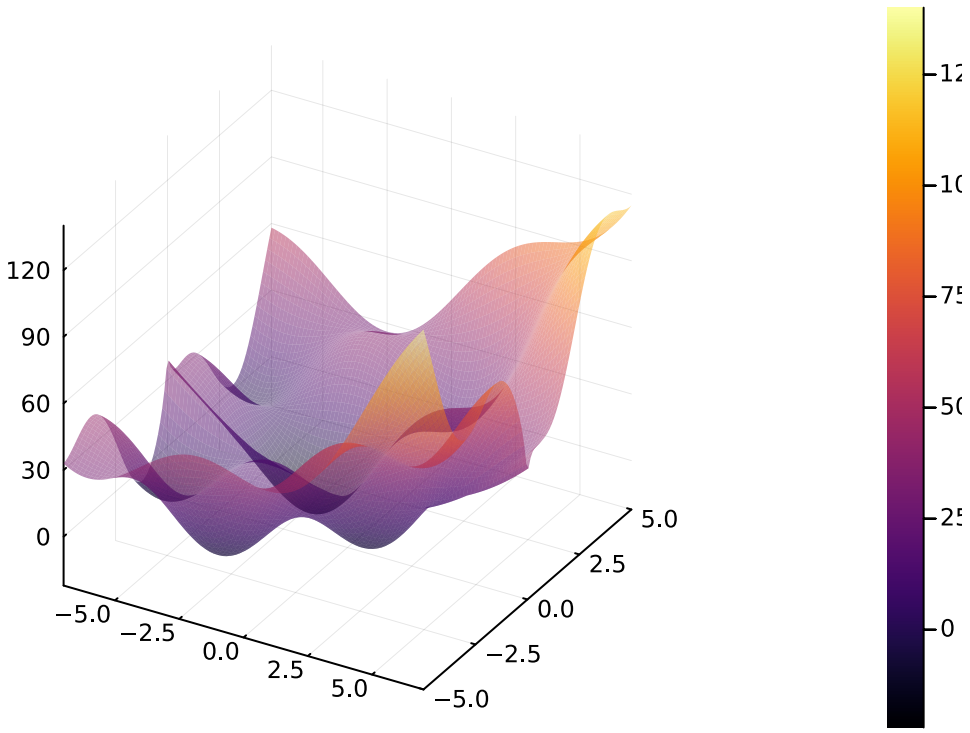
Out[42]:
```
* Status: failure (reached maximum number of iterations)

* Candidate solution
   Final objective value:     -2.201309e+00

* Found with
   Algorithm:     Simulated Annealing

* Convergence measures
   |x - x'|               = NaN ≰ 0.0e+00
   |x - x'|/|x'|          = NaN ≰ 0.0e+00
   |f(x) - f(x')|         = NaN ≰ 0.0e+00
   |f(x) - f(x')|/|f(x')| = NaN ≰ 0.0e+00
   |g(x)|                 = NaN ≰ 1.0e-08

* Work counters
   Seconds run:   0  (vs limit Inf)
   Iterations:    1000
   f(x) calls:    1001
```

In [43]:
```
opt = optimize(f, [20.0, 10.0], ParticleSwarm(n_particles  = 10))
```

Out[43]:
```
* Status: failure (reached maximum number of iterations)

* Candidate solution
   Final objective value:     -2.206300e+00

* Found with
   Algorithm:     Particle Swarm

* Convergence measures
   |x - x'|               = NaN ≰ 0.0e+00
   |x - x'|/|x'|          = NaN ≰ 0.0e+00
   |f(x) - f(x')|         = NaN ≰ 0.0e+00
   |f(x) - f(x')|/|f(x')| = NaN ≰ 0.0e+00
   |g(x)|                 = NaN ≰ 1.0e-08

* Work counters
   Seconds run:   0  (vs limit Inf)
   Iterations:    1000
   f(x) calls:    11010
   ∇f(x) calls:   0
```

In [44]:
```
f(x) =  x[1]^2 + 1.5x[2]^2 + 2x[2] + log(10x[1]^2 + 0.1)*x[2] + sin(x[1])*sin(x[
fp(x, y) = f([x, y])

xs = LinRange(-7, 7, 100)
ys = LinRange(-5, 5, 100)
zs = [fp(x, y) for x in xs, y in ys]

surface(xs, ys, zs,  alpha = 0.5)
```

Out[44]:



In [45]:
```
opt = optimize(f, [20.0, 10.0], NelderMead())
```

Out[45]:
```
 * Status: success

 * Candidate solution
    Final objective value:     -8.369366e+00

 * Found with
    Algorithm:     Nelder-Mead

 * Convergence measures
    √(Σ(yᵢ-ȳ)²)/n ≤ 1.0e-08

 * Work counters
    Seconds run:   0  (vs limit Inf)
    Iterations:    46
    f(x) calls:    91
```

In [46]:
```
opt = optimize(f, [20.0, 10.0], Newton())
```

Out[46]:
```
* Status: success

* Candidate solution
   Final objective value:     2.550431e+01

* Found with
   Algorithm:     Newton's Method

* Convergence measures
   |x - x'|               = 1.33e-06 ≰ 0.0e+00
   |x - x'|/|x'|          = 1.92e-07 ≰ 0.0e+00
   |f(x) - f(x')|         = 1.28e-11 ≰ 0.0e+00
   |f(x) - f(x')|/|f(x')| = 5.02e-13 ≰ 0.0e+00
   |g(x)|                 = 2.54e-10 ≤ 1.0e-08

* Work counters
   Seconds run:   0  (vs limit Inf)
   Iterations:    6
   f(x) calls:    19
   ∇f(x) calls:   19
   ∇²f(x) calls:  6
```

In [47]:
```
opt = optimize(f, [20.0, 10.0], SimulatedAnnealing())
```

Out[47]:
```
* Status: failure (reached maximum number of iterations)

* Candidate solution
   Final objective value:     -2.197822e+01

* Found with
   Algorithm:     Simulated Annealing

* Convergence measures
   |x - x'|               = NaN ≰ 0.0e+00
   |x - x'|/|x'|          = NaN ≰ 0.0e+00
   |f(x) - f(x')|         = NaN ≰ 0.0e+00
   |f(x) - f(x')|/|f(x')| = NaN ≰ 0.0e+00
   |g(x)|                 = NaN ≰ 1.0e-08

* Work counters
   Seconds run:   0  (vs limit Inf)
   Iterations:    1000
   f(x) calls:    1001
```

In [49]:
```
opt = optimize(f, [20.0, 10.0], ParticleSwarm(n_particles  = 10))
```

Out[49]:
```
* Status: failure (reached maximum number of iterations)

* Candidate solution
   Final objective value:     -2.204752e+01

* Found with
   Algorithm:     Particle Swarm

* Convergence measures
   |x - x'|               = NaN ≰ 0.0e+00
   |x - x'|/|x'|          = NaN ≰ 0.0e+00
   |f(x) - f(x')|         = NaN ≰ 0.0e+00
   |f(x) - f(x')|/|f(x')| = NaN ≰ 0.0e+00
   |g(x)|                 = NaN ≰ 1.0e-08

* Work counters
   Seconds run:   0  (vs limit Inf)
   Iterations:    1000
   f(x) calls:    11010
   ∇f(x) calls:   0
```

In [16]:
```
optimize(f, Optim.minimizer(opt), Newton())
```

Out[16]:
```
* Status: success (objective increased between iterations)

* Candidate solution
   Final objective value:     -2.204752e+01

* Found with
   Algorithm:     Newton's Method

* Convergence measures
   |x - x'|               = 3.26e-09 ≰ 0.0e+00
   |x - x'|/|x'|          = 2.05e-09 ≰ 0.0e+00
   |f(x) - f(x')|         = 3.55e-15 ≰ 0.0e+00
   |f(x) - f(x')|/|f(x')| = 1.61e-16 ≰ 0.0e+00
   |g(x)|                 = 0.00e+00 ≤ 1.0e-08

* Work counters
   Seconds run:   0  (vs limit Inf)
   Iterations:    1
   f(x) calls:    2
   ∇f(x) calls:   2
   ∇²f(x) calls:  1
```

Оптимизация функции максимального правдоподобия

In [17]:
```
using Distributions, Random, LinearAlgebra

dist = MvNormal([3,4], [3 2; 2 5])
```

Out[17]:
```
FullNormal(
dim: 2
μ: [3.0, 4.0]
Σ: [3.0 2.0; 2.0 5.0]
)
```

In [18]:
```
vec = [rand(dist) for i in 1:100]
```

Out[18]:
```
100-element Vector{Vector{Float64}}:
 [3.519971051588785, 6.799243635439609]
 [6.079443106094903, 5.520418203645795]
 [3.307453736880199, 5.4236910159432075]
 [0.683832496010504, 1.6089327244626657]
 [5.208336315860387, 8.683300314145944]
 [4.265583369924813, 4.097760825663072]
 [1.9169813679624412, 1.6922217351288928]
 [1.7449447082488774, 6.531122026593321]
 [7.287174400461363, 5.2971191316629636]
 [3.4882401125556455, 5.617570289236675]
 [2.374076133220708, 2.9662235871109934]
 [1.2318165562111811, 6.024148948104515]
 [3.232320997314211, 4.6817299003505735]
 ⋮
 [0.8976521180647916, 4.269065575087244]
 [-0.1739289032972886, -0.14641640572704162]
 [3.356824741084341, 4.332813475805639]
 [7.711307422858507, 6.976398655348792]
 [1.2678834869669935, -1.1183087625383656]
 [0.3748885192711726, 3.2838908724134397]
 [4.079579987922051, 3.895724266299666]
 [2.29471867441954, 2.887320005032916]
 [2.6068036086075383, 2.935096007052728]
 [-0.7538032263080647, 2.5080629579951887]
 [1.7143494190884336, 1.530575979443611]
 [1.0855379094035376, 6.233792084066112]
```

Помним функцию правдоподобия:

$$L(\theta) = \prod pdf(D(\theta), Data)$$

Где pdf - фенкция плотности вероятности, D распределение, Data - значения для статистически независимого наблюдения.

Которая после лог-преобразования превращается:

$$logL(\theta) = \sum logpdf(D(\theta), Data)$$

Почему можно использовать логарифмирование?

Так-как $log(x)$ гладкая неубывающая функция, то минимум функции $f(x)$ совпадает с минимум $log(f(x))$.

Ковариационная матрица

$$cov(\theta) = \begin{pmatrix} {\sigma^2}_1 & \sigma_1\sigma_2\rho \\ \sigma_1\sigma_2\rho & {\sigma^2}_2 \end{pmatrix}$$

In [50]:
```
likelihood(θ, data) = begin
        means = θ[1:2]
        σ²₁   = θ[3]^2
        σ²₂   = θ[4]^2
        ρ     = θ[5]
        σ₁σ₂ρ = θ[3]*θ[4]*ρ
        covm = [σ²₁ σ₁σ₂ρ;
                σ₁σ₂ρ σ²₂]
```

```
        -sum(map(x -> logpdf(MvNormal(means, Symmetric(covm)), x), data))
    end
```

Out[50]: likelihood (generic function with 1 method)

In [51]:
```
likelihood([10,32, 5, 4, 0.1], vec)
```

Out[51]: 2982.37446040261

In [52]:
```
opt = optimize(x -> likelihood(x, vec), [10., 10., 10., 10., 0.1], BFGS())
```

```
PosDefException: matrix is not positive definite; Cholesky factorization failed.

Stacktrace:
  [1] checkpositivedefinite
    @ C:\Users\vsarn\AppData\Local\Programs\Julia-1.8.2\share\julia\stdlib\v1.8\L
inearAlgebra\src\factorization.jl:18 [inlined]
  [2] cholesky!(A::Symmetric{Float64, Matrix{Float64}}, ::NoPivot; check::Bool)
    @ LinearAlgebra C:\Users\vsarn\AppData\Local\Programs\Julia-1.8.2\share\juli
a\stdlib\v1.8\LinearAlgebra\src\cholesky.jl:270
  [3] #cholesky#162
    @ C:\Users\vsarn\AppData\Local\Programs\Julia-1.8.2\share\julia\stdlib\v1.8\L
inearAlgebra\src\cholesky.jl:402 [inlined]
  [4] cholesky (repeats 2 times)
    @ C:\Users\vsarn\AppData\Local\Programs\Julia-1.8.2\share\julia\stdlib\v1.8\L
inearAlgebra\src\cholesky.jl:402 [inlined]
  [5] PDMats.PDMat(mat::Symmetric{Float64, Matrix{Float64}})
    @ PDMats C:\Users\vsarn\.julia\packages\PDMats\bzppG\src\pdmat.jl:19
  [6] MvNormal
    @ C:\Users\vsarn\.julia\packages\Distributions\fYgbJ\src\multivariate\mvnorma
l.jl:201 [inlined]
  [7] (::var"#29#30"{Matrix{Float64}, Vector{Float64}})(x::Vector{Float64})
    @ Main .\In[50]:10
  [8] iterate
    @ .\generator.jl:47 [inlined]
  [9] _collect(c::Vector{Vector{Float64}}, itr::Base.Generator{Vector{Vector{Floa
t64}}, var"#29#30"{Matrix{Float64}, Vector{Float64}}}, #unused#::Base.EltypeUnkno
wn, isz::Base.HasShape{1})
    @ Base .\array.jl:807
 [10] collect_similar
    @ .\array.jl:716 [inlined]
 [11] map
    @ .\abstractarray.jl:2933 [inlined]
 [12] likelihood(θ::Vector{Float64}, data::Vector{Vector{Float64}})
    @ Main .\In[50]:10
 [13] (::var"#31#32")(x::Vector{Float64})
    @ Main .\In[52]:1
 [14] finite_difference_gradient!(df::Vector{Float64}, f::var"#31#32", x::Vector
{Float64}, cache::FiniteDiff.GradientCache{Nothing, Nothing, Nothing, Vector{Floa
t64}, Val{:central}(), Float64, Val{true}()}; relstep::Float64, absstep::Float64,
dir::Bool)
    @ FiniteDiff C:\Users\vsarn\.julia\packages\FiniteDiff\grio1\src\gradients.j
l:318
 [15] finite_difference_gradient!
    @ C:\Users\vsarn\.julia\packages\FiniteDiff\grio1\src\gradients.jl:258 [inlin
ed]
 [16] (::NLSolversBase.var"#g!#15"{var"#31#32", FiniteDiff.GradientCache{Nothing,
Nothing, Nothing, Vector{Float64}, Val{:central}(), Float64, Val{true}()}})(stora
ge::Vector{Float64}, x::Vector{Float64})
    @ NLSolversBase C:\Users\vsarn\.julia\packages\NLSolversBase\kavn7\src\object
ive_types\oncedifferentiable.jl:57
 [17] (::NLSolversBase.var"#fg!#16"{var"#31#32"})(storage::Vector{Float64}, x::Ve
ctor{Float64})
    @ NLSolversBase C:\Users\vsarn\.julia\packages\NLSolversBase\kavn7\src\object
ive_types\oncedifferentiable.jl:61
 [18] value_gradient!!(obj::OnceDifferentiable{Float64, Vector{Float64}, Vector{F
loat64}}, x::Vector{Float64})
    @ NLSolversBase C:\Users\vsarn\.julia\packages\NLSolversBase\kavn7\src\interf
ace.jl:82
 [19] value_gradient!(obj::OnceDifferentiable{Float64, Vector{Float64}, Vector{Fl
oat64}}, x::Vector{Float64})
```

```
    @ NLSolversBase C:\Users\vsarn\.julia\packages\NLSolversBase\kavn7\src\interf
ace.jl:69
 [20] value_gradient!(obj::Optim.ManifoldObjective{OnceDifferentiable{Float64, Ve
ctor{Float64}, Vector{Float64}}}, x::Vector{Float64})
    @ Optim C:\Users\vsarn\.julia\packages\Optim\V8ZEC\src\Manifolds.jl:50
 [21] (::LineSearches.var"#φdφ#6"{Optim.ManifoldObjective{OnceDifferentiable{Floa
t64, Vector{Float64}, Vector{Float64}}}, Vector{Float64}, Vector{Float64}, Vector
{Float64}})(α::Float64)
    @ LineSearches C:\Users\vsarn\.julia\packages\LineSearches\G1LRk\src\LineSear
ches.jl:84
 [22] (::LineSearches.HagerZhang{Float64, Base.RefValue{Bool}})(φ::Function, φd
φ::LineSearches.var"#φdφ#6"{Optim.ManifoldObjective{OnceDifferentiable{Float64, V
ector{Float64}, Vector{Float64}}}, Vector{Float64}, Vector{Float64}, Vector{Float
64}}, c::Float64, phi_0::Float64, dphi_0::Float64)
    @ LineSearches C:\Users\vsarn\.julia\packages\LineSearches\G1LRk\src\hagerzha
ng.jl:139
 [23] HagerZhang
    @ C:\Users\vsarn\.julia\packages\LineSearches\G1LRk\src\hagerzhang.jl:101 [in
lined]
 [24] perform_linesearch!(state::Optim.BFGSState{Vector{Float64}, Matrix{Float6
4}, Float64, Vector{Float64}}, method::BFGS{LineSearches.InitialStatic{Float64},
LineSearches.HagerZhang{Float64, Base.RefValue{Bool}}, Nothing, Nothing, Flat},
d::Optim.ManifoldObjective{OnceDifferentiable{Float64, Vector{Float64}, Vector{Fl
oat64}}})
    @ Optim C:\Users\vsarn\.julia\packages\Optim\V8ZEC\src\utilities\perform_line
search.jl:58
 [25] update_state!(d::OnceDifferentiable{Float64, Vector{Float64}, Vector{Float6
4}}, state::Optim.BFGSState{Vector{Float64}, Matrix{Float64}, Float64, Vector{Flo
at64}}, method::BFGS{LineSearches.InitialStatic{Float64}, LineSearches.HagerZhang
{Float64, Base.RefValue{Bool}}, Nothing, Nothing, Flat})
    @ Optim C:\Users\vsarn\.julia\packages\Optim\V8ZEC\src\multivariate\solvers\f
irst_order\bfgs.jl:139
 [26] optimize(d::OnceDifferentiable{Float64, Vector{Float64}, Vector{Float64}},
initial_x::Vector{Float64}, method::BFGS{LineSearches.InitialStatic{Float64}, Lin
eSearches.HagerZhang{Float64, Base.RefValue{Bool}}, Nothing, Nothing, Flat}, opti
ons::Optim.Options{Float64, Nothing}, state::Optim.BFGSState{Vector{Float64}, Mat
rix{Float64}, Float64, Vector{Float64}})
    @ Optim C:\Users\vsarn\.julia\packages\Optim\V8ZEC\src\multivariate\optimize\
optimize.jl:54
 [27] optimize
    @ C:\Users\vsarn\.julia\packages\Optim\V8ZEC\src\multivariate\optimize\optimi
ze.jl:36 [inlined]
 [28] #optimize#89
    @ C:\Users\vsarn\.julia\packages\Optim\V8ZEC\src\multivariate\optimize\interf
ace.jl:143 [inlined]
 [29] optimize (repeats 2 times)
    @ C:\Users\vsarn\.julia\packages\Optim\V8ZEC\src\multivariate\optimize\interf
ace.jl:139 [inlined]
 [30] top-level scope
    @ In[52]:1
```

Связующие функции - неубывающие гладкие функции:

$$f(x) = exp(x)$$

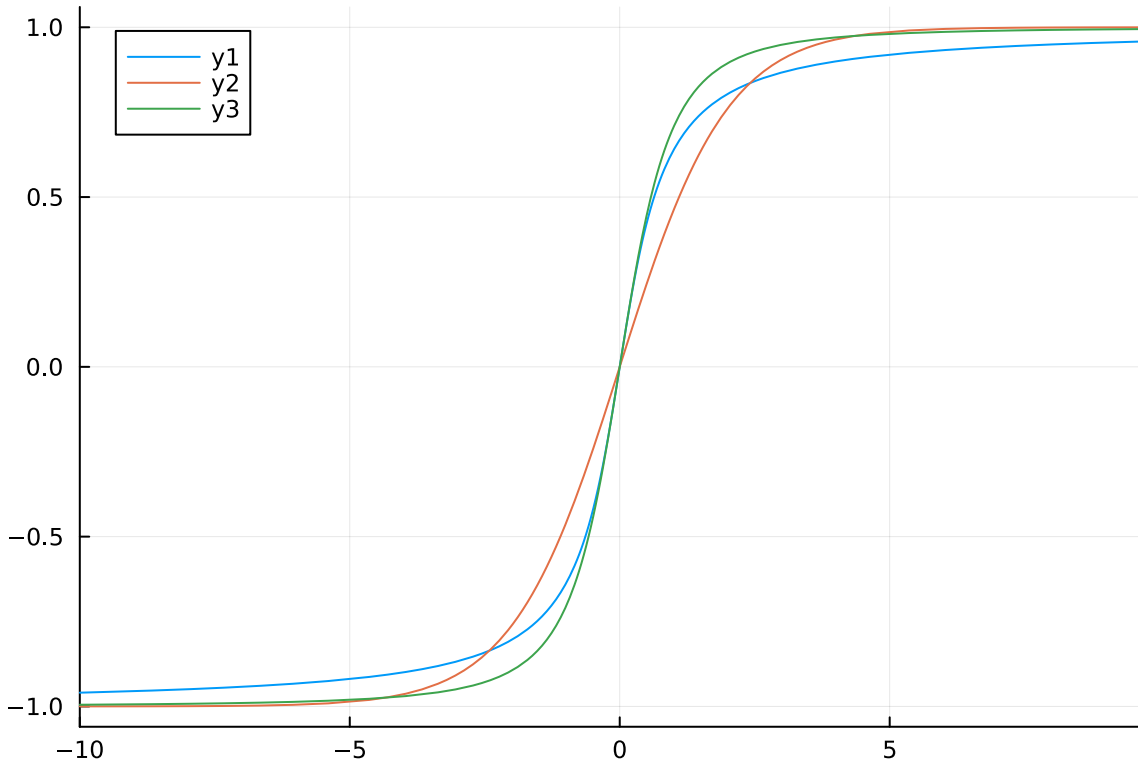
$$f(x) = \frac{1}{1 + exp(-x)}$$

$$f(x) = arctg(x)$$

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

In [29]:
```
plot(x-> 2/pi*atan(pi/2*x), xlims = (-10, 10))
plot!(x-> 2 / (1 + exp(-x)) - 1)
plot!(x-> x / sqrt(1 + x^2))
```

Out[29]:



In [23]:
```
function sigmf(x)
    2 / (1 + exp(-x)) - 1
end

function linkf(p)
    v = zeros(5)
    v[1] = p[1]
    v[2] = p[2]
    v[3] = exp(p[3])
    v[4] = exp(p[4])
    v[5] = sigmf(p[5])
    v
end
```

Out[23]:
```
linkf (generic function with 1 method)
```

In [53]:
```
optf(x) = begin
    likelihood(linkf(x), vec)
end

optf([1., 1., 10., 10., 1.1])
```

Out[53]:
```
2169.3688939500466
```

In [54]:
```
opt = optimize(optf, [10., 10., 1., 1., 1.1], Newton())
```

Out[54]:
```
* Status: success

* Candidate solution
   Final objective value:     4.079789e+02

* Found with
   Algorithm:     Newton's Method

* Convergence measures
   |x - x'|               = 1.39e-05 ≰ 0.0e+00
   |x - x'|/|x'|          = 3.47e-06 ≰ 0.0e+00
   |f(x) - f(x')|         = 3.06e-09 ≰ 0.0e+00
   |f(x) - f(x')|/|f(x')| = 7.51e-12 ≰ 0.0e+00
   |g(x)|                 = 9.39e-09 ≤ 1.0e-08

* Work counters
   Seconds run:   0  (vs limit Inf)
   Iterations:    7
   f(x) calls:    29
   ∇f(x) calls:   29
   ∇²f(x) calls:  7
```

In [55]:
```
Optim.minimizer(opt)
```

Out[55]:
```
5-element Vector{Float64}:
 2.7874801091322223
 3.993382051989382
 0.6553584606934048
 0.7489365060031238
 1.1708457675684534
```

In [56]:
```
res = linkf(Optim.minimizer(opt))
```

Out[56]:
```
5-element Vector{Float64}:
 2.7874801091322223
 3.993382051989382
 1.925832728997522
 2.1147497965634643
 0.5265957163298016
```

In [57]:
```
covmat = [res[3]^2 res[3]*res[4]*res[5]; res[3]*res[4]*res[5] res[4]^2]

MvNormal(res[1:2], covmat)

#dist = MvNormal([3,4], [3 2; 2 5])
```

Out[57]:
```
FullNormal(
dim: 2
μ: [2.7874801091322223, 3.993382051989382]
Σ: [3.7088317000780426 2.144642346314774; 2.144642346314774 4.472166702065214]
)
```

Ссылки

- **Julia**
  - Ссылка: https://julialang.org/
  - Шеррингтон Малькольм - Осваиваем язык Julia.

    - Вадим Никитин - Julia. Язык программирования. Быстрый старт.

- **Документация к основным пакетам Julia**
    - *Математика и анализ*
        - Roots.jl (нахождение корней) https://juliamath.github.io/Roots.jl/stable/
        - Optim.jl (Поиск минимума/максимума) https://julianlsolvers.github.io/Optim.jl/stable/#user/minimization/
        - ForwardDiff.jl (Дифференцирование) https://juliadiff.org/ForwardDiff.jl/stable/
        - QuadGK.jl (Численное интегрирование) https://juliamath.github.io/QuadGK.jl/stable/
        - DifferentialEquations.jl (Численное решение дифференциальных уравнений) https://diffeq.sciml.ai/stable/
        - ExperimentalDesign.jl (Оптимизация дизайна) https://github.com/phrb/ExperimentalDesign.jl
    - *Статистика*
        - Distributions.jl (Основной пакет для работы с распределениями) https://juliastats.org/Distributions.jl/stable/
        - Statistics - Базовый пакет с основными статистическими функциями (не требует установки, но надо подключать `using Statistics`)
        - HypothesisTests.jl (основные статистические критерии) https://juliastats.org/HypothesisTests.jl/stable/
        - GLM.jl (Общая/обобщенная линейная модель) https://juliastats.org/GLM.jl/stable/
    - *Другое*
        - Plots.jl (Графики) https://docs.juliaplots.org
- **Математика**
    - Математика. Базовый курс / Б.Ш. Гулиян, Р.Я. Хамидуллин. Москва : Синергия, 2013. - 712 с. - ISBN 978-5-4257-0109-1.
    - Д.Т. Письменный - Конспект лекций по высшей математике
    - The Matrix Cookbook
    - Линейная алгебра и ее применения - Стренг Г.
    - Матричный анализ - Хорн Р., Джонсон Ч.

- **Планирование эксперимента**
  - Ч. Хикс - Основные принципы планрования эксперимента.
  - Н.И. Сидняев - Теория планирования эксперимента и анализ статистических данных.
- **Теория вероятностей**
  - Фадеева Л. Н., Лебедев А. В., - Теория вероятностей и математическая статистика: учебное пособие. - 2-е изд., перераб. и доп. - М.: Эксмо, 2010. - 496 с.
  - Гмурман В. Е. - Теория вероятностей и математическая статистика, учебное пособие.
  - Гмурман В. Е. - Руководство к решению задач по теории вероятностей и математической статистике – решебник и задачник.
- **Дополнительная литература:**
  - Майкл Кохендерфер, Тим Уилер - Алгоритмы оптимизации.
  - Кохендерфер Майкл, Рэй Кайл, Уилер Тим - Алгоритмы принятия решений.
  - Математическая статистика в медицине, В. А. Медик, М. С. Токмачев, 978-5-279-03195-5.
  - Медико-биологическая статистика. Гланц. Пер. с англ. — М., Практика, 1998. — 459 с.
  - Уилл Курт - Байесовская статистика: Star Wars, LEGO, резиновые уточки и многое другое.
  - Занимательная статистика. Манга. Син Такахаси, 2009, 224с, ISBN: 978-5-97060-179-2.
  - Ю.Д. Григорьев - Методы оптимального планирования эксперимента.

In [ ]: