

Принцип максимального правдоподобия. Регуляризация при нелинейной оптимизации и метод Левенберга-Марквардта

Table of Contents

1.Функция правдоподобия и принцип максимального правдоподобия.....	1
Биномиальное распределение:.....	2
Распределение Пуассона:.....	5
2.Принцип максимального правдоподобия для решения задач регрессии при Гауссовой ошибке.....	9
Функция правдоподобия для коррелированной гауссовой ошибки:.....	13
Геометрия расстояния Махalanобиса.....	14
Откуда берут матрицу ковариации методе нелинейной регрессии?.....	15
Пример. Линейная регрессия для данных коррелированных во времени.....	18
3.Функция невязки из принципа максимального правдоподобия для нелинейной регрессии в предположении Гауссовой ошибки.....	21
Оптимизация градиентными методами при регуляризации	24
4. Метод Левенberга-Марквардта.....	25
Шаг алгоритма и регуляризующая матрица.....	25

Сегодня мы, во-первых, уйдем от обычного метода наименьших квадратов (**Ordinary Least Squares**) и будем рассматривать более общие модели, как для линейной так и для нелинейной регрессии. Основным способом получения таких моделей является принцип максимального правдоподобия, основная идея которого состоит в том, чтобы найти такую оценку параметров модели, которая наилучшим образом (с наибольшей вероятностью) предсказывает появление тех данных, которые наблюдаются в эксперименте. Кроме того целью будет проследить как работает регуляризация в задачах нелинейной регрессии.

1.Функция правдоподобия и принцип максимального правдоподобия

Пусть у нас есть некоторый набор экспериментальных точек в виде вектора \vec{y} с N элементами, которые являются независимыми выборками некоторой случайной величины, то есть все эти точки получены из одного распределения вероятности, характеризуемого набором параметров \vec{b} . Тогда вероятность получить наблюдаемый набор точек будет равна произведению вероятностей каждого из исходов (y_i) в отдельности:

$$\mathcal{P}(\vec{y}; \vec{b}) = \prod_{i=1}^N p(y_i; \vec{b}) \quad (1)$$

$p(y_i | \vec{b})$ - вероятность получить точку со значением y_i из некоторого распределения, характеризуемого набором параметров \vec{b} .

Если рассматривать вероятность как функцию параметров \vec{b} , то она называется функцией правдоподобия.

$$\mathcal{L}(\vec{b} | \vec{y}) = \mathcal{P}(\vec{y}, \vec{b}) \quad (2)$$

Основная идея принципа максимального правдоподобия состоит в том чтобы находить такие параметры \vec{b} , которые максимизируют вероятность получить набор точек, измеренных в эксперименте:

$$\vec{b}_{MLE} = \operatorname{argmax}(\mathcal{L}(\vec{b} | \vec{y}))$$

MLE - maximum likelihood estimator

Биномиальное распределение:

$p(y|n, p) = \binom{n}{y} \psi^y (1 - \psi)^{n-y}, \quad y = 0, 1, \dots, n$ - используется при анализе дискретных событий с бинарным

исходом типа бросания монетки, ψ - вероятность единичного положительного исхода, n - число бросаний.

Значение плотности вероятности дает вероятность получить y удачных исходов при n розыгрышах.

В качестве параметров этого распределения будем рассматривать вероятность получения положительного исхода p при единичном бросании. В качестве единичного исхода y_i будем рассматривать число выпадения "решки" при фиксированном числе бросков n . Функция правдоподобия рассчитывается по формулам (1,2).

```
%  
clearvars  
p_true = 0.42;  
N = 80;% число розыгрышей  
M = 10;% число бросаний в каждой точке  
data = rand(N,M) < p_true
```

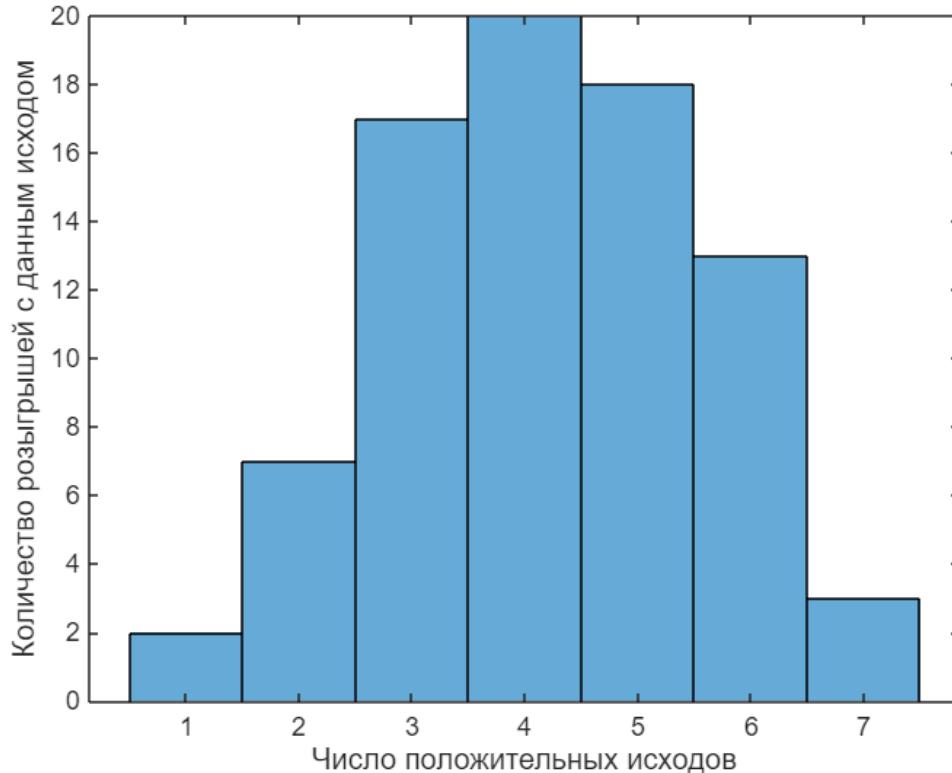
```
data = 80x10 logical array  
1 1 0 0 0 1 1 0 0 0  
0 0 1 1 0 1 1 0 1 0  
0 1 0 1 0 1 1 0 0 1  
1 1 0 1 1 0 0 1 1 0  
0 1 0 0 0 0 1 0 0 1  
0 1 0 0 0 0 0 0 1 0  
0 1 0 0 0 0 1 0 0 1  
1 1 0 0 0 1 1 0 1 1  
0 1 1 0 1 0 1 0 1 1  
0 1 1 1 1 0 0 1 1 0  
:
```

```
p = 0.41;  
bin_fun = @(k) nchoosek(M,k)*(p^k)*(1-p)^(M-k) ;  
p_i = zeros(N,1);  
y_i= zeros(N,1);  
for ii = 1:N
```

```

y_i(ii) = sum(data(ii,:)); % число удачных бросаний в ii-м розыгрыше
p_i(ii) = bin_fun(y_i(ii));% вероятность данного исхода
end
histogram(y_i)
xlabel("Число положительных исходов")
ylabel("Количество розыгрышей с данным исходом")

```



```
L = prod(p_i)
```

```
L = 3.8483e-62
```

```

psi = 0:1e-3:1;
LikeliHoodFunc = @(d,p) prod(arrayfun(@(k) nchoosek(M,k)*(p^k)*(1-p)^(M-k),
sum(d,2)))

```

```

LikeliHoodFunc = function_handle with value:
    @(d,p)prod(arrayfun(@(k)nchoosek(M,k)*(p^k)*(1-p)^(M-k),sum(d,2)))

```

```
Li = arrayfun(@(p_ii)LikeliHoodFunc(data,p_ii),psi)
```

```

Li = 1×1001
10-61 ×
    0         0         0         0         0         0         0         0     ...

```

```

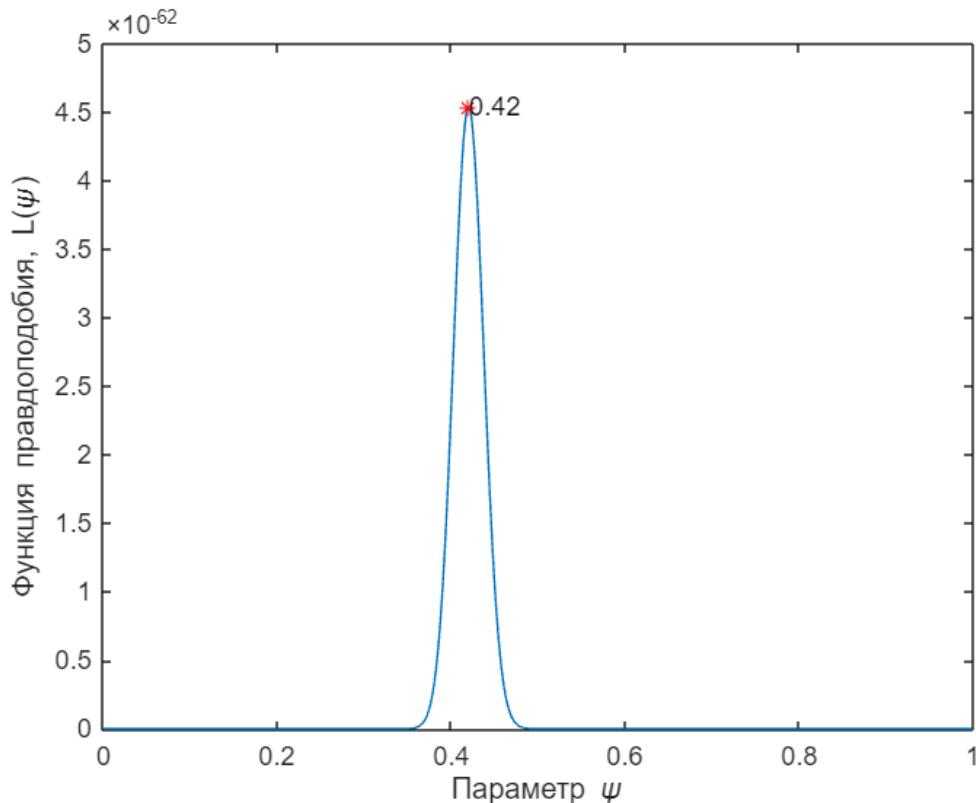
plot(psi,Li)
xlabel("Параметр \psi")
ylabel("Функция правдоподобия, L(\psi)")
hold on

```

```
[lmax,indmax] = max(Li)
```

```
lmax = 4.5385e-62  
indmax = 421
```

```
scatter(psi(indmax),lmax,"*r")  
text(psi(indmax),lmax,string(psi(indmax)))  
hold off
```



Максимизируется как правило логарифм функции правдоподобия (loglikelihood):

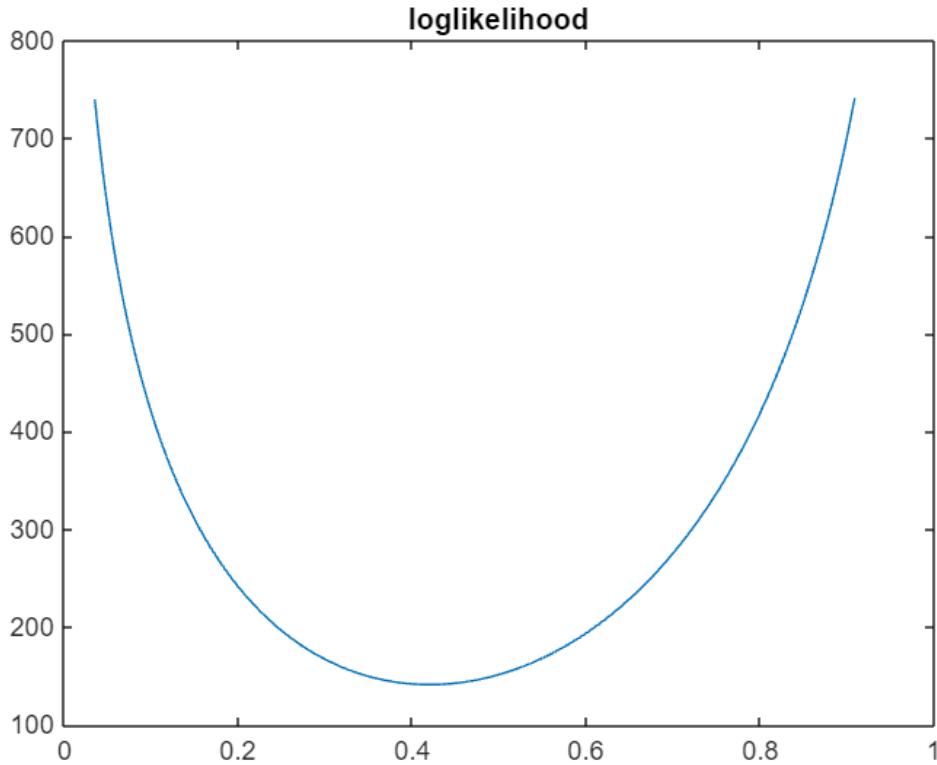
$$\ell(\vec{b}) = \log(\mathcal{L})$$

Либо минимизируется $-\ell(\vec{b})$

```
logl_fn = @(p_ii)-log(LikeliHoodFunc(data,p_ii))
```

```
logl_fn = function_handle with value:  
@(p_ii)-log(LikeliHoodFunc(data,p_ii))
```

```
plot(psi,arrayfun(logl_fn,psi))  
title("loglikelihood")
```



Одномерная оптимизация с ограничениями, чтобы найти параметр распределения

```
fminbnd(logl_fn,0,1)
```

```
ans = 0.4200
```

Распределение Пуассона:

Как уже упоминалось, принцип максимального правдоподобия может использоваться и для другой других функций распределения.

Распределение Пуассона:

$p(y|\lambda) = \frac{\lambda^y e^{-\lambda}}{y!}$, $y = 0, 1, 2, \dots$ дает вероятность получения y редких событий, где параметр λ - это среднее

число событий. Например, если производится наблюдение за некоторым процессом во времени, то λ будет средним числом событий за весь интервал времени, а y - это конкретное число событий.

Предполагаем события (измерения) независимыми, соответственно для формулировки функции правдоподобия мы можем пользоваться [формулой](#).

Вектор экспериментальных данных \vec{y} , мы предполагаем, что каждая точка получена из распределения Пуассона со своим средним значением λ_i , так как мы хотим несмешенную оценку, наша модель должна, соответственно, быть равна среднему значению для каждой из точек $f(\vec{b})_i = \lambda_i$. Функция правдоподобия:

$$\mathcal{L}(\vec{b} \mid \vec{y}) = \prod_i \frac{(f_i)^{y_i} e^{-f_i}}{y_i!}$$

loglikelihood:

$$\ell(\vec{b} \mid \vec{y}) = \log\left(\prod_i \frac{(f_i)^{y_i} e^{-f_i}}{y_i!}\right) = \sum_i [y_i \log(f_i) - \log(y_i!) - f_i]$$

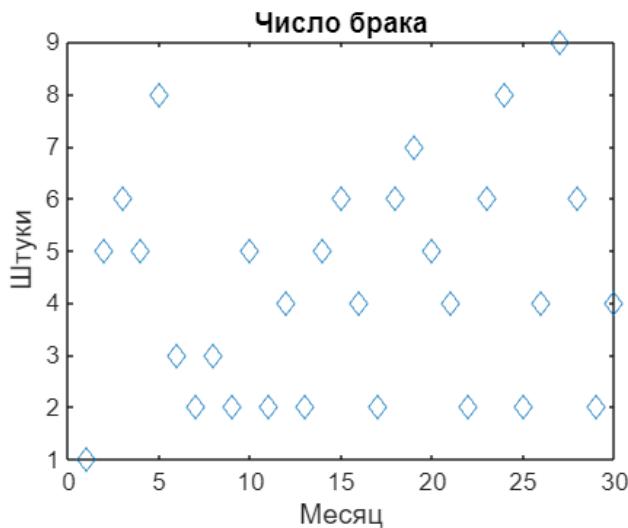
Распределение Пуассона применяется для анализа редких событий, таких, как, например, возникновение брака в технологическом цикле. Имея, например, статистику разрушений изделий, можно построить модель для анализа тренда на уменьшение или увеличение количества брака.

```
clearvars
N = 30; % длительность наблюдения
l0 = 4;
l1 = 1/N;
l2 = -5e-3/N;
l_true = [l2,l1,l0];
x = transpose(1:N);
lambda_true = polyval(l_true,x);% истинное изменение среднего значения от месяца к месяцу
y = arrayfun(@(lami)poissrnd(lami,1,1),lambda_true(:)); % вектор экспериментальных данных - число брака за 50 месяцев
table(y) % таблица количества брака в месяц
```

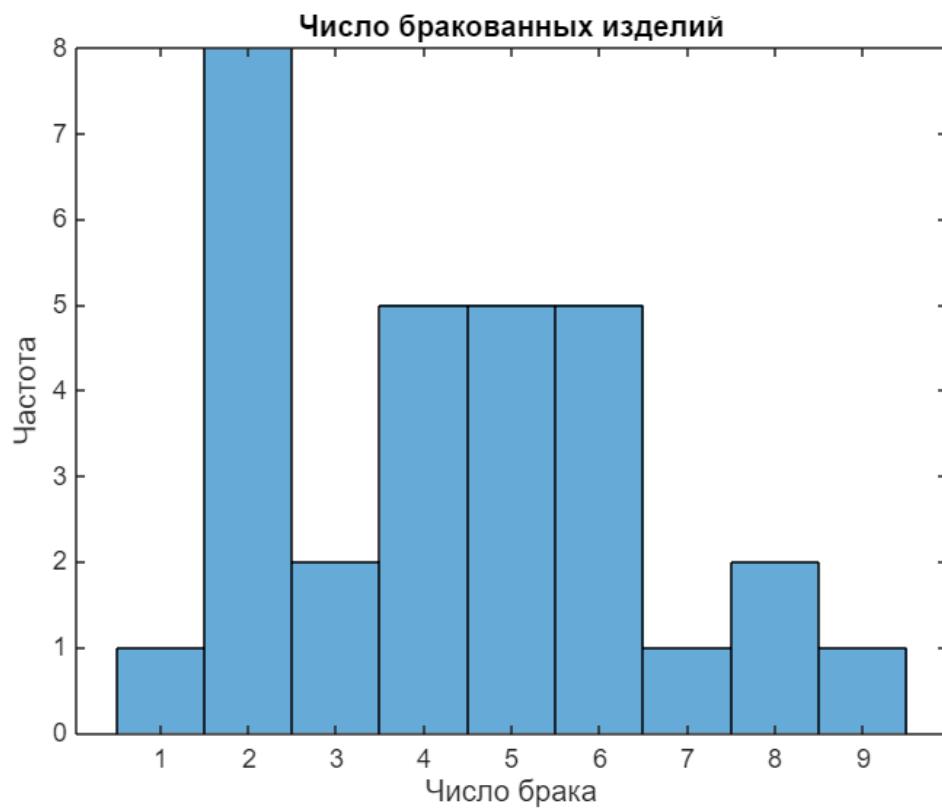
ans = 30x1 table

	y
1	1
2	5
3	6
4	5
5	8
6	3
7	2
8	3
9	2
10	5
11	2
12	4
13	2
14	5
:	

```
plot(x,y,"diamond");title("Число брака");xlabel("Месяц");ylabel("Штуки");
```



```
histogram(y);title("Число бракованных изделий");ylabel("Частота");xlabel("Число брака")
```



```
poisson_quality = 0;  
polyfit_quality = 0;  
pp_mean= 0;  
pf_mean = 0;  
trials_number = 5
```

```

trials_number = 5

for ii = 1:trials_number
    y = arrayfun(@(lami)poissrnd(lami,1,1),lambda_true(:));
    f_average = @(p) polyval(p,x);
    logl_fn = @(p) -sum(y.*log(f_average(p))) - log(gamma(y + 1)) - f_average(p)); %
вместо факториала используем
% Гамма функцию, которая распространяет понятие факториала на все
% действительные числа
    poisson_regression = fminsearch(logl_fn,[1e-3,1e-2,3]); % решение задачи
оптимизации
    polynomial_fit = polyfit(x,y,2); % обычный фитинг полиномом
    pp_mean = pp_mean + poisson_regression;
    pf_mean = pf_mean + polynomial_fit;
    poisson_quality = poisson_quality + norm(poisson_regression - l_true)/
norm(l_true);
    polyfit_quality = polyfit_quality + norm(polynomial_fit - l_true)/norm(l_true);
end
poisson_regression = pp_mean/trials_number

```

```

poisson_regression = 1×3
0.0003    0.0193    3.9170

```

```

polynomial_fit= pf_mean/trials_number

```

```

polynomial_fit = 1×3
0.0008    0.0072    3.9738

```

```

l_true

```

```

l_true = 1×3
-0.0002    0.0333    4.0000

```

```

poisson_quality= poisson_quality/trials_number

```

```

poisson_quality = 0.3039

```

```

polyfit_quality= polyfit_quality/trials_number

```

```

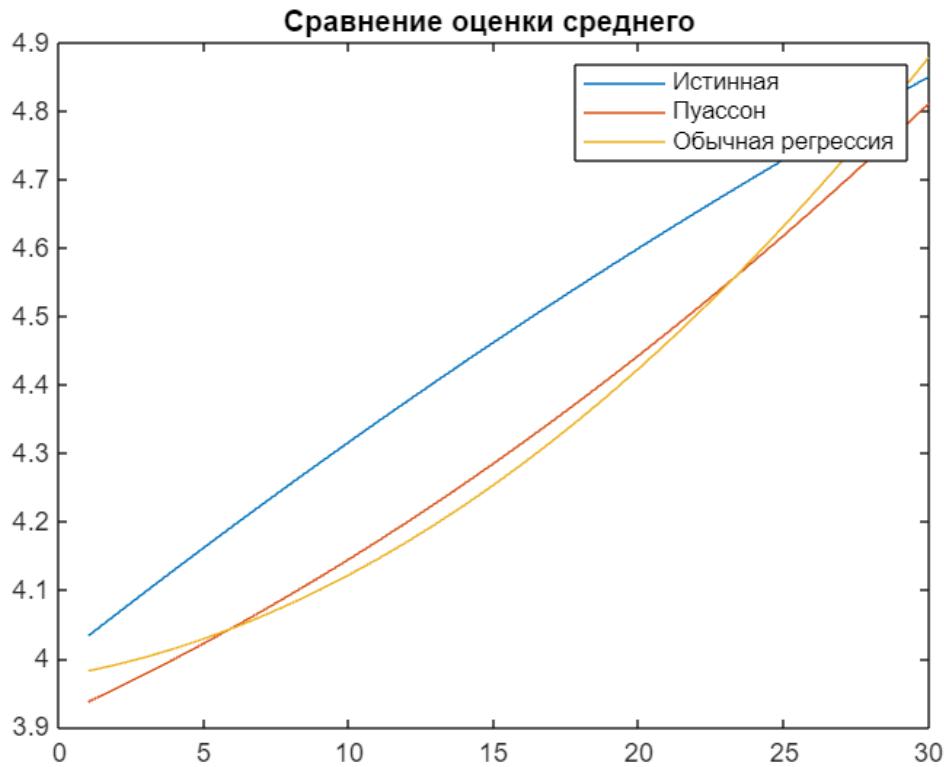
polyfit_quality = 0.3024

```

```

plot(x,lambda_true);
hold on
plot(x,f_average(poisson_regression));
plot(x,polyval(polynomial_fit,x));
hold off
legend(["Истинная", "Пуассон", "Обычная регрессия"])
title("Сравнение оценки среднего ")

```



2. Принцип максимального правдоподобия для решения задач регрессии при Гауссовой ошибке

У нас есть некоторая модель $f(\vec{\beta}, \vec{x})$, которая зависит от набора параметров $\vec{\beta}$ и независимых переменных \vec{x} . В задачах оптимизации, мы хотим найти оценку \vec{b} истинных параметров модели $\vec{\beta}$ по измеренным данным \vec{y} . Сама модель полагается верной и детерминированной, то есть, все отклонения измеренных данных от модели имеют случайную природу. Таким образом, можно сказать, что ошибка для каждого отдельного измерения получена из некоторого распределения.

Вопрос пригодности модели $f(\vec{\beta}, \vec{x})$ для описания какого-то явления или процесса может быть изучен после решения задачи регрессии путем анализа справедливости исходного предположения регрессии - случайности ошибки регрессии.

Пусть ошибка имеет нормальное распределение:

$\mathcal{N}(0, 1)$ - стандартное нормальное распределение (функция randn)

Чтобы получить переменную, взятую из $\zeta \sim \mathcal{N}(\mu, \sigma^2)$ из переменной, взятой из стандартного распределения $\eta \sim \mathcal{N}(0, 1)$

$$\zeta \sim \mathcal{N}(\mu, \sigma^2) = \mu + \sigma^2 \eta$$

Ошибка равна разнице экспериментальных данных и предсказания модели для i -й точки:

$$e_i = y_i - \vec{f}(\vec{b})_i$$

Если ошибка берется из несмещенному распределения ($\mathcal{N}(0, \sigma^2)$), то для отдельного измерения:

$$p(e_i) \propto p_0 \exp\left(-\frac{1}{2} \frac{(y_i - f_i)^2}{\sigma^2}\right)$$

Это эквивалентно тому, что $y_i \sim \mathcal{N}(f_i, \sigma)$, то есть, сами точки получены из нормального распределения со средним значением f_i

Предполагая, что ошибки не зависят другого от друга, согласно формуле (1) для loglikelihood:

$$\ell(\vec{b}) = \log(\mathcal{L}) = \log\left(\prod_i^N p_0 \exp\left(-\frac{1}{2} \frac{(y_i - f_i)^2}{\sigma^2}\right)\right) = C - \frac{1}{2\sigma^2} \sum_i^N (y_i - f_i)^2 = C - \frac{1}{2} (\vec{y} - \vec{f})^T (\vec{y} - \vec{f})$$

При условии, что σ^2 постоянна для всех точек, мы получим обычновенный метод наименьших квадратов (ordinary least squares).

Выражение для вероятности конкретного значения можно было бы переписать так:

$$\vec{y} \sim \mathcal{N}(\vec{f}, \sigma I)$$

Прицип максимального правдоподобия может быть использован для оценки параметров модели даже если ошибка получена из многомерного распределения Гаусса $\vec{y} \sim \mathcal{N}(\vec{\mu}, \Sigma)$, где:

$$\mathcal{N}(\vec{y} | \vec{\mu}, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2} (\vec{y} - \vec{\mu})^T \Sigma^{-1} (\vec{y} - \vec{\mu})\right)$$

Здесь d - число измерений, Σ - матрца ковариации размером (dxd), $\vec{\mu}$ - вектор средних значений по каждой из координат, $|\Sigma|$ - определитель матрицы ковариации.

```
MVG2 = @(x,mu,Sigma) exp(-(x(:)-mu(:))'*(Sigma\((x(:)-mu(:))/2)/
(2*pi*sqrt(det(Sigma))) % двумерное нормальное распределение
```

```
MVG2 = function_handle with value:
    @(x,mu,Sigma)exp(-(x(:)-mu(:))'*(Sigma\((x(:)-mu(:))/2)/(2*pi*sqrt(det(Sigma)))
```

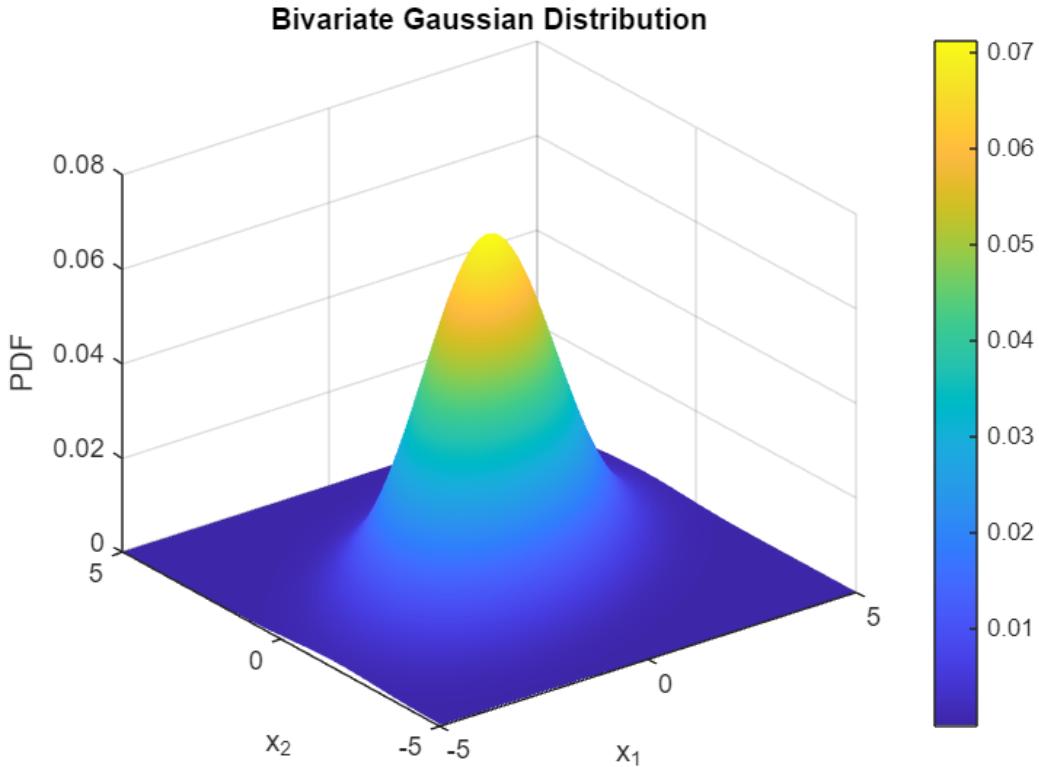
```
Sigma = [3,1; ...
         1,2]
```

```
Sigma = 2x2
      3   1
      1   2
```

```
mu = [0;0]
```

```
mu = 2x1
  0
  0

try
    det(Sigma);
    chol(Sigma);
catch ex
    disp(ex.message)
    disp("not positive definite")
    return
end
% Create grid for evaluation
x1 = -5:0.1:5;
x2 = -5:0.1:5;
[X1, X2] = meshgrid(x1, x2);
X = [X1(:), X2(:)]; %
Ngrid = size(X,1);
y = zeros(Ngrid,1);
for ii = 1:Ngrid
    y(ii) = MVG2(X(ii,:),mu,Sigma);
end
y = reshape(y, length(x2), length(x1));
surf(X1, X2, y);
xlabel('x_1'); ylabel('x_2'); zlabel('PDF');
title('Bivariate Gaussian Distribution');
colorbar; shading interp;
```



```
[u,s] = eig(Sigma)
```

```
u = 2x2
 0.5257 -0.8507
 -0.8507 -0.5257
s = 2x2
 1.3820      0
      0   3.6180
```

```
u'*u
```

```
ans = 2x2
 1.0000      0
      0   1.0000
```

```
s = diag(s);
ax = get_next_ax()
```

```
fig11
ax =
Axes with properties:

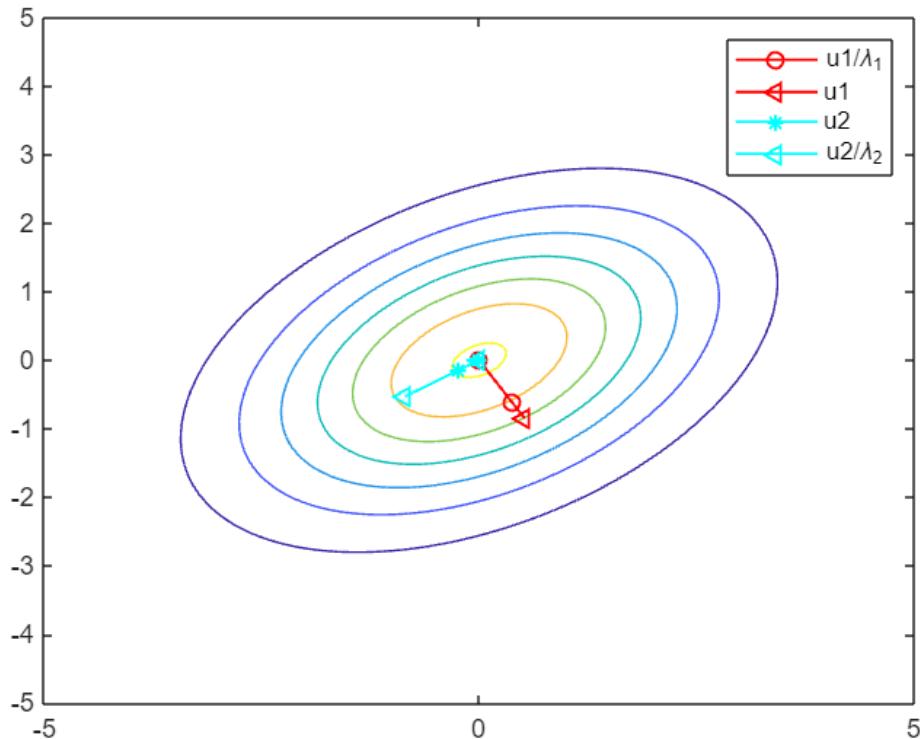
    XLim: [0 1]
    YLim: [0 1]
    XScale: 'linear'
    YScale: 'linear'
    GridLineStyle: '-'
    Position: [0.1300 0.1100 0.7750 0.8150]
    Units: 'normalized'
```

Show all properties

```

contour(ax,X1,X2,y)
hold(ax, "on");
plot(ax,[mu(1);u(1,1)/s(1)],[mu(2),u(2,1)/
s(1)],"Marker", "o", "Color", "red", "LineWidth",1)
plot(ax,[mu(1);u(1,1)],[mu(2),u(2,1)],"Marker", "<", "Color", "red", "LineWidth",1)
plot(ax,[mu(1);u(1,2)/s(2)],[mu(2),u(2,2)/
s(2)],"Marker", "*", "Color", "cyan", "LineWidth",1)
plot(ax,[mu(1);u(1,2)],[mu(2),u(2,2)],"Marker", "<", "Color", "cyan", "LineWidth",1)
hold(ax, "off")
legend(ax,[ "", "u1/\lambda_1", "u1", "u2", "u2/\lambda_2" ])

```



В матлабе есть встроенная функция которая позволяет генерировать случайные вектора из гауссова распределения :

`mvnrnd(mu, Sigma)` - первый аргумент - вектор средних, `Sigma` - матрица ковариации распределения

Помимо Гауссова, существует еще несколько распределений, которые имеют подобную компактную форму для коррелированных данных, но для случая произвольного распределения получить такое простое выражение не удается.

Функция правдоподобия для коррелированной гауссовой ошибки:

$$\mathcal{L}(\vec{b} | \vec{y}) \propto \exp\left(-\frac{1}{2}(\vec{f}_b - \vec{y})^T \Sigma^{-1} (\vec{f}_b - \vec{y})\right) \quad (3)$$

Σ - матрица ковариации входных данных, формула остается справедливой для коррелированных данных!

Геометрия расстояния Махаланобиса

Конструкцию, которая стоит в показатели экспоненты называют *Mahalanobis distance*, она имеет смысл расстояния между двумя векторами \vec{f}_b и \vec{y} .

Если вспомнить, что матрица ковариации является симметричной положительно-определенной, то матрицы ее собственных векторов являются ортонормированными, спектральное разложение этой будет иметь вид: $\Sigma = U \Lambda U^T = \sum_d \lambda_d \vec{u}_d \vec{u}_d^T$, где \vec{u}_d - собственные векторы матрицы ковариации, это единичные векторы показывающие направление главных осей информационной матрицы, тогда расстояние Махаланобиса между двумя векторами это:

$$\Sigma^{-1} = \sum_d \frac{1}{\lambda_d} \vec{u}_d \vec{u}_d^T$$

Чем меньше λ - большая вариация в данном направлении!

$$\vec{r}^T \Sigma^{-1} \vec{r} = \sum_d \frac{1}{\lambda_d} [\vec{r}^T \vec{u}_d] [\vec{u}_d^T \vec{r}] = \sum_d \frac{z_d^2}{\lambda_d}$$

Вектора \vec{u}_d - единичные векторы, следовательно

$z_d = \vec{r}^T \vec{u}_d$ - это длина проекции вектора разности на главную ось эллипса.

```
t = table([[u(:,1),u(:,2)];[norm(u(:,1)),norm(u(:,1))];s']);
t.Properties.RowNames = string({'u(1)', 'u(2)', '||u||', 'lambda'})
```

t = 4x1 table

	Var1	
	1	2
1 u(1)	0.5257	-0.8507
2 u(2)	-0.8507	-0.5257
3 u	1.0000	1.0000
4 lambda	1.3820	3.6180

При решении обратных задач нестационарного теплопереноса, в качестве измеренных данных, выступают зависимости температур от времени, измеренный при нагреве образца, а в качестве параметров модели, для коэффициентной ОЗ могут выступать, например, коэффициенты полиномиальной аппроксимации физических свойств материала (теплопроводности, теплоемкости), для граничной ОЗ коэффициенты аппроксимации теплового потока и т.п.

Модель $f(\vec{\beta}, \vec{x})$ включает в себя, также решение прямой задачи - дифференциального уравнения в частных производных (в простейшем случае одномерного нестационарного) способ решения УЧП также имеет ряд параметров, таких как, например, дискретность сетки по времени и координате, которые в

задаче регрессии напрямую не фигурируют, однако эти "скрытые" параметры также могут быть включены в задачу оптимизации.

Откуда берут матрицу ковариации методе нелинейной регрессии?

Для нахождения матрицы Σ используется априорная информация об ошибке исходных данных. Причем она не обязана быть строгой матрицей ковариации, так как, в задаче регрессии Σ это, в конечном счете, некоторый набор весов, который характеризует вклад отдельных точек ошибки в результирующую невязку.

Эмпирическим путем.

Допустим, в качестве входных данных для ОЗ выступают термопарные измерения, полученные в ходе эксперимента. Например, зависимости температур от времени в нескольких точках образца

$\vec{T}_k = T(x_k, t)$ - вектор из M элементов . В этом случае можно провести серию нагревов в одних и тех же (с экспериментальной точки зрения) условиях. Фактически в качестве оценки может быть использована воспроизводимость результатов измерений. Проведя N измерений, получим матрицу

$\Theta_k = [\vec{T}^1 - \langle \vec{T} \rangle, \dots, \vec{T}^N - \langle \vec{T} \rangle]$, где $\langle \vec{T} \rangle$ - вектор усредненной по серии нагревов зависимости температуры от времени (вектор из K элементов). Тогда матрицу ковариации можно оценить как:

$$\Sigma = \Theta_k \Theta_k^T / (N - 1)$$

```
clearvars
type = "diagonal"

type =
"diagonal"

M = 30;
N = 200;
if type=="diagonal"
    true_Sigma = diag(M:-1:1);
else
    [i, j] = meshgrid(1:M, 1:M);
    true_Sigma = 30*exp(- abs(i - j)/10)
end
Ttrue = transpose(M*(1:M)); % средняя температура (истинная)
Thetta = zeros(M,N);
for ii = 1:N
    Thetta(:,ii) = Ttrue + mvnrnd(0*Ttrue, true_Sigma)';
end
Tavrgs_sample = mean(Thetta,2);
Thetta = Thetta - Tavrgs_sample;
Sigma_sample = Thetta*Thetta'/(N-1)
```

```
Sigma_sample = 30x30
 30.6788   -1.7451    0.8334    2.7597    0.1839   -0.5264    0.9424   -0.2643 ...
 -1.7451   34.1801    1.8574   -1.6670    0.8635   -1.0289   -0.0979    0.0717
```

```

0.8334   1.8574   29.4507   2.0622   -1.6629   -4.2641   -0.4744   -0.8691
2.7597  -1.6670   2.0622   23.9765   4.5717  -1.2309  -0.2124   3.4407
0.1839   0.8635  -1.6629   4.5717  25.7055  -2.2289   0.8619   1.2286
-0.5264  -1.0289  -4.2641  -1.2309  -2.2289  31.2860   0.6083   1.4902
0.9424  -0.0979  -0.4744  -0.2124   0.8619   0.6083  26.5634   1.1679
-0.2643   0.0717  -0.8691   3.4407  1.2286   1.4902   1.1679  19.5224
-0.6768   2.5959   2.4640   0.7668   0.6310  -0.5728  -0.0508  -0.2005
-2.6814   0.4184  -1.3621   1.2383  -1.4275  -0.7961  -2.7030  -0.1568
:

```

```
norm(true_Sigma - Sigma_sample)/norm(true_Sigma)
```

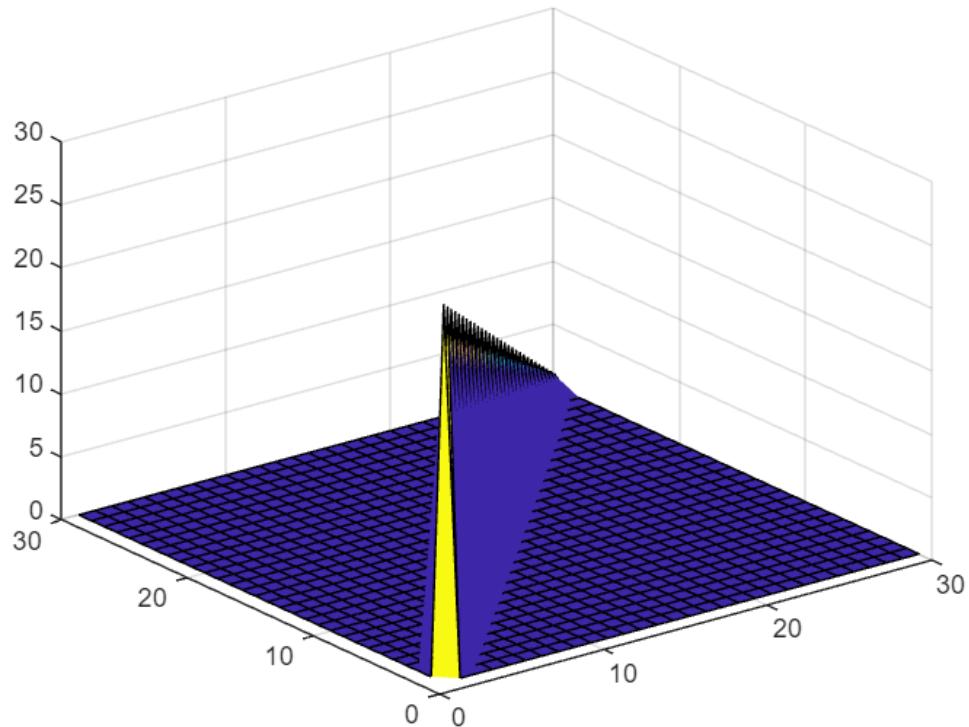
```
ans = 0.4376
```

```
norm(true_Sigma)
```

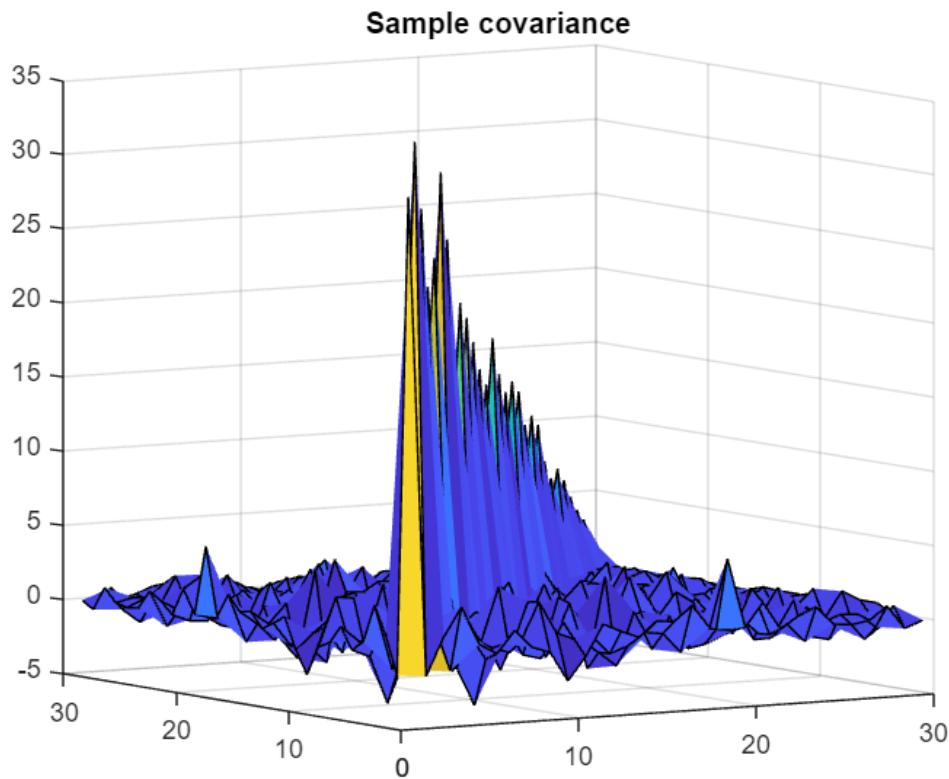
```
ans = 30
```

```
surf(true_Sigma);title("True covariance")
```

True covariance



```
surf(Sigma_sample); title("Sample covariance")
```



Погрешность прибора.

Например, для термопарных измерений, в качестве матрицы могут быть взяты погрешности термопарных измерений

Классы точности по ГОСТ Р 8.585-2001:

Для термопар типа S (ТПП, платина — 10% родий/платина) ГОСТ Р 8.585-2001 устанавливает два класса допуска: класс 1 (повышенная точность) и класс 2 (стандартный).

Погрешности для класса 1

- От 0 до 1100°C: $\pm 1,0^\circ\text{C}$.
- От 1100 до 1600°C: $\pm [1,0 + 0,003(t - 1100)]^\circ\text{C}$, где t — температура в °C.

Погрешности для класса 2

- От 0 до 600°C: $\pm 1,5^\circ\text{C}$.
- От 600 до 1600°C: $\pm 0,0025t$ °C, где t — температура в °C.

Эти пределы представляют максимальную погрешность; для моделирования в ковариационной матрице можно использовать $\sigma \approx \frac{\zeta}{3}$.

Эмпирические "предположени".

Часто предполагают, что вариация прямо пропорциональна самому значению, тогда $\sigma = \psi T$, где ψ это, например, 1e-2, тогда:

$$\Sigma = \psi \text{diag}(\vec{T}^2)$$

Данные с прибора, получаемые с прибора с конечным временем релаксации τ .

Ясно, что если частота семплования меньше времени релаксации, соседние точки по времени будут связаны друг с другом. Для определенности предположим, что речь идет об измерении температуры в зависимости от времени, термопара вместе имеет какое-то конечное время прогрева, а частота оцифровки сигнала с термопары может быть больше этого времени. Ошибка измерения складывается из двух составляющих, случайной ошибки (e_i) и ошибки "корреляции":

$$y_i = \vec{a}^T [\begin{array}{c} y_1 \\ \vdots \\ y_{i-1} \end{array}] + e_i$$

Коэффициенты \vec{a} корреляционной ошибки не должны зависеть от индекса вектора \vec{y} (то есть от времени), а должны определяться только некоторым интервалом времени, предшествующим точке y_i

Тогда ковариация может быть представлена в виде функции с релаксацией:

$$\Sigma_{ij} = \sigma^2 \exp\left(-\frac{|t_i - t_j|}{\tau}\right) = \sigma^2 \exp\left(-\frac{dt|i - j|}{\tau}\right)$$

Такого вида матрицы, которые зависят от разницы индексов называются матрицами Тёплица, они имеют постоянные элементы на главной и побочной диагоналях (элементы $-N$ и $+N$ диагоналей не обязаны быть одинаковыми, то есть, квадратная матрица Тёплица не обязательно симметрична).

Пример. Линейная регрессия для данных коррелированных во времени.

Ошибка берется из гауссова распределения с матрицей ковариации, соответствующей коррелированным данным. И сравнивается взвешенная регрессия и обыкновенный метод наименьших квадратов.

```
clearvars
sigma2 = 0.01; % variance
tau = 3.89; % relaxation time
dt = 0.1; % sampling time < tau
N = 100; % number of samples
% toeplitz
Fs = 1/dt;
ff = Fs/N*(-N/2:N/2-1);
num_trials = 10;
t = (0:N-1)*dt;
[i, j] = meshgrid(1:N, 1:N);
Sigma = sigma2 * exp(- dt*abs(i - j)/tau) %
```

```
Sigma = 100x100
0.0100    0.0097    0.0095    0.0093    0.0090    0.0088    0.0086    0.0084    ...
```

```

0.0097  0.0100  0.0097  0.0095  0.0093  0.0090  0.0088  0.0086
0.0095  0.0097  0.0100  0.0097  0.0095  0.0093  0.0090  0.0088
0.0093  0.0095  0.0097  0.0100  0.0097  0.0095  0.0093  0.0090
0.0090  0.0093  0.0095  0.0097  0.0100  0.0097  0.0095  0.0093
0.0088  0.0090  0.0093  0.0095  0.0097  0.0100  0.0097  0.0095
0.0086  0.0088  0.0090  0.0093  0.0095  0.0097  0.0100  0.0097
0.0084  0.0086  0.0088  0.0090  0.0093  0.0095  0.0097  0.0100
0.0081  0.0084  0.0086  0.0088  0.0090  0.0093  0.0095  0.0097
0.0079  0.0081  0.0084  0.0086  0.0088  0.0090  0.0093  0.0095
:

```

```

V = [ones(N,1), t'];%, t'.^2,t'.^3];
p_true = [0.2, -1.4e-2]';%,0.14e-3,-0.14e-6]';
f_true = V*p_true;
cond(Sigma)

```

ans = 3.9446e+03

```

r_mse_weighted = 0;
r_mse_unweighted = 0;
for iii = 1:num_trials % чтобы увидеть эффект прогоним регрессию несколько раз, так
% как для одного раза этот эффект может и не быть
y = f_true+ mvnrnd(0*f_true, Sigma)'; % Fixed transpose

% weighted
p_weighted = (V'* (Sigma \ V)) \ (V' * (Sigma \ y));

% unweighted
p_unweighted = (V'*V) \ (V'*y);
r_mse_weighted = r_mse_weighted + norm(p_weighted - p_true) / norm(p_true);
r_mse_unweighted = r_mse_unweighted + norm(p_unweighted - p_true) /
norm(p_true);
end
disp("Average RMSE:")

```

Average RMSE:

```
disp("Weighted =" + r_mse_weighted/num_trials);
```

Weighted =0.35527

```
disp("Unweighted =" + r_mse_unweighted/num_trials);
```

Unweighted =0.47107

```
ax = get_next_ax();
```

fig22

```

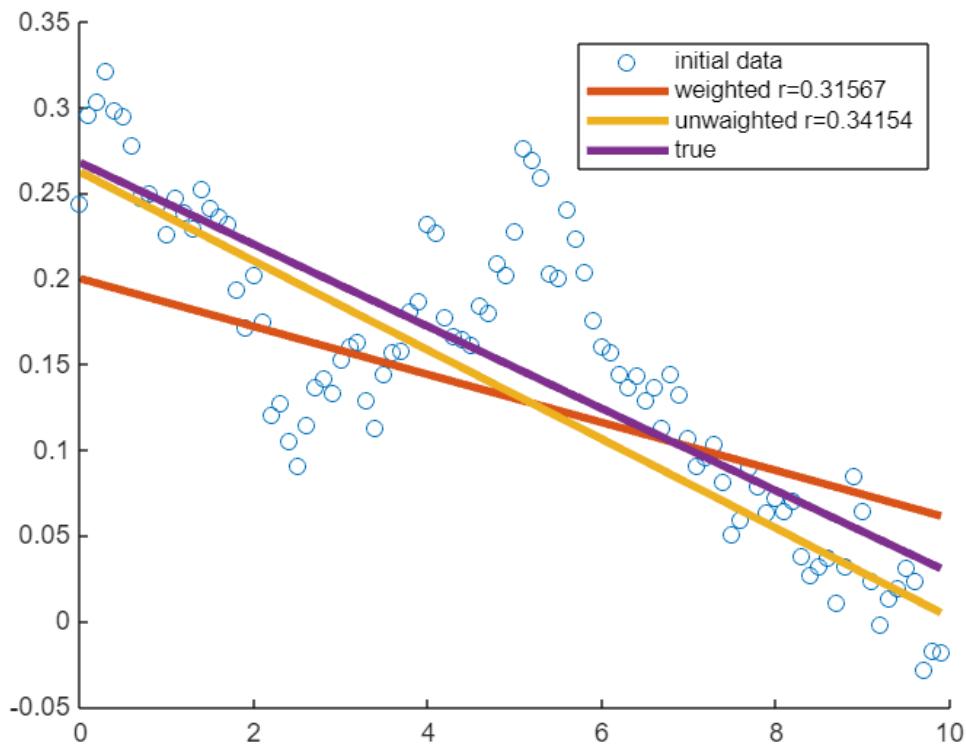
scatter(ax,t',y)
hold(ax,"on");
plot(ax,t',V*p_true,"LineWidth",3);
plot(ax,t',V*p_weighted,"LineWidth",3);
plot(ax,t',V*p_unweighted,"LineWidth",3);

```

```

hold(ax, "off")
legend(ax,[ "initial data", "weighted r="+ norm(p_true - p_weighted)/norm(p_true),
...
"unweighted r="+ norm(p_true - p_unweighted)/norm(p_true), "true"])

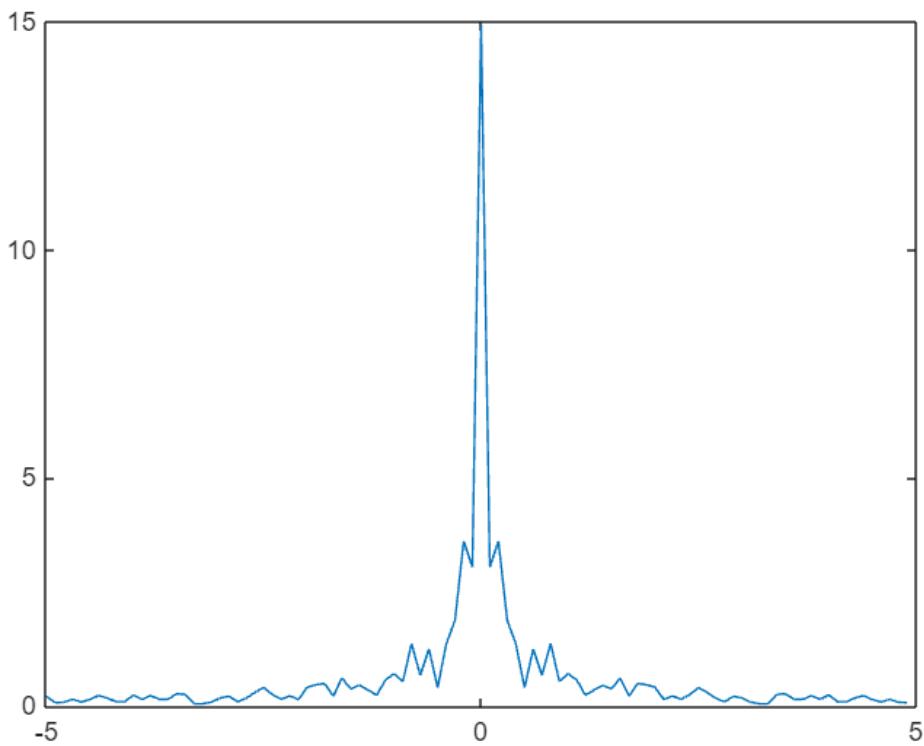
```



```
t = table(p_true,p_weighted,p_unweighted)
```

	p_true	p_weighted	p_unweighted
1	0.2000	0.2621	0.2678
2	-0.0140	-0.0260	-0.0239

```
%, "ColumnNames", {'initial data', 'weighted', 'unweighted'})
% covariance from fft
fft_y = abs(fftshift(fft(y)));
plot(ff,fft_y)
```



Видно, что, в среднем взвешенная регрессия выдает меньшую относительную ошибку оценки истинных значений параметров (хотя визуально так не скажешь)

При условии, что измерения независимы друг от друга, матрица ковариации будет диагональной:

$$\sigma_1^2 \dots 0$$

$W = (0 \ \ddots \ 0)$, диагональные элементы - вариации для каждого из измерений.

$$0 \dots \sigma_n^2$$

$W = \sigma I$, если все экспериментальные точки получены из одного и того же Гауссова распределения.

3.Функция невязки из принципа максимального правдоподобия для нелинейной регрессии в предположении Гауссовой ошибки

Все методы оптимизации, использующие данные о производной функции (как первого, так и второго порядков) строятся на вычислении производных первого и второго порядка, фундаментальных направлениях, антиградиента и шага Ньютона, поэтому вспомним основные соотношения.

Задача регрессии состоит в том, чтобы найти такой вектор параметров \vec{b} , который минимизирует скалярную функцию невязки $\Phi(\vec{b})$ - мера вектора ошибки:

$\operatorname{argmin}(\Phi(\vec{b}))$

Вектор \vec{r} - это разница между предсказанием модели ($f(\vec{b}, \vec{x})$) и экспериментально наблюдаемым (или желаемым) постоянным вектором (\vec{y}).

Статистическими предположениями, будет, что ошибка получена из Гауссова распределения.

Далее используются следующие обозначения:

ОФ - общая формулировка

МЛГ - невязка, полученная из принципа максимального правдоподобия в предположении Гауссовой ошибки входных данных

ВЛРГ - взвешенная линейная регрессия частный случай модели **МЛ** с линейной зависимостью от

вектора параметров $f(\vec{b}, \vec{x}) = A(\vec{x})\vec{b}$ (сама функция невязки $\Phi(\vec{b})$ при этом является квадратичной формой от параметров оптимизации.

$$\sigma_1^2 \dots 0$$

$W = (0 \dots 0)$ - априорная матрица ковариации входных данных, диагональные элементы - вариации для каждого из измерений. Обыкновенный метод наименьших квадратов - это частный случай

Далее будем обозначать функцию $f(\vec{b}, \vec{x})$, рассчитанную для каждого значения вектора \vec{x} как \vec{f}_b

МЛ: $\Phi(\vec{b}) = \frac{1}{2} \vec{r} W^{-1} \vec{r} = \frac{1}{2} (\vec{f}_b - \vec{y})^T W^{-1} (\vec{f}_b - \vec{y})$

f - некоторая нелинейная функция от вектора параметров оптимизации при заданной независимой переменной \vec{x} , возвращающая вектор значений для каждого значения независимой переменной.

ЛР: $\Phi(\vec{b}) = \frac{1}{2} (A \vec{b} - \vec{y})^T W^{-1} (A \vec{b} - \vec{y})$ - МНК для линейной регрессии (A - постоянная матрица, например для полиномиального вида функции это матрица Вандермонда)

Итерационная оптимизация, подразумевает, что из текущего положения в пространстве параметров \vec{b}_k перейти в новую точку \vec{b}_{k+1} , которая будет ближе к точке локального минимума \vec{b}^* :

$$\vec{b}_{k+1} = \vec{b}_k + \vec{p}_k(\vec{b}_k \dots)$$

Вектор шага между итерациями (\vec{p}) рассчитывается в соответствии с алгоритмом.

3. Метод градиентного поиска (метод первого порядка):

Как известно вектор градиента функции направлен в сторону ее максимального роста, следовательно вектор антиградиента направлен в сторону уменьшения:

$$\text{ОФ: } \vec{p}_{gradient\ descend}^k = -\mu \hat{\vec{g}}(\vec{b}^k)$$

$\hat{\vec{g}}(\vec{b}) = \frac{\nabla_b \Phi(\vec{b})}{\|\nabla_b \Phi(\vec{b})\|}$ - вектор единичной длины, направленный в сторону антиградиента

$$\text{МЛ: } \nabla_b \Phi(\vec{b}) = J^T W^{-1} (\vec{f}_b - \vec{y})$$

J - матрица Якоби, для которой $J_{ij} = \frac{\partial f_i}{\partial b_j}(x_i)$, то есть, на i -й строке стоит градиент функции $\nabla_b f(\vec{b}, x_i)$

$$\text{ЛР: } \nabla_b \Phi(\vec{b}) = A^T W^{-1} (A \vec{b} - \vec{y})$$

для задачи линейной регрессии, $J = A$. Условие равенства нулю градиента дает нормальное уравнение

для решения задачи линейной взвешенной регрессии: $\vec{b} = (A^T W^{-1} A)^{-1} A^T W^{-1} \vec{y}$

Метод Ньютона (метод второго порядка):

$$\text{ОФ: } \vec{p}_{Newton} = -(\nabla_b^2 \Phi(\vec{b}, \vec{x}))^{-1} \nabla_b \Phi(\vec{b}, \vec{x})$$

$\text{МЛ: } \nabla_b^2 \Phi(\vec{b}) = J^T W^{-1} J + H = J^T W^{-1} J + \sum_q H(x_q) r_q$, где $H_{ij}(x_q) = \frac{\partial^2 f}{\partial b_i \partial b_j}(x_q)$ - матрица вторых производных, рассчитанная в точке x_q

$\vec{p}^k = -(J^T W^{-1} J + H)^{-1} J^T W^{-1} (\vec{f}_b^k - \vec{y})$ - шаг в методе Ньютона

$\text{ЛР: } \nabla_b^2 \Phi(\vec{b}) = A^T W^{-1} A$, так как вторые производные от квадратичной формы равны нулю

$$\vec{p} = -(A^T W^{-1} A)^{-1} A^T W^{-1} (A \vec{b} - \vec{y})$$

Проблемы метода Ньютона: Если матрица Гессса $\nabla_b^2 \Phi(\vec{b})$ не является положительно-определенной, то метод будет шагать не в ту сторону (можно проверить путем перемножения с вектором антиградиента, если скалярное произведение отрецательно, то угол с антиградиентом тупой, значит шагаем мы в сторону от минимума). Кроме того, матрица Гессса может оказаться синглярной! Тогда непонятно куда шагать, есть бесконечно много вариантов.

Метод Гаусса-Ньютона:

Получается если в шаге по методу Ньютона пренебречь матрицей вторых производных, а оставить

только перекрестное произведение первых $\nabla_b^2 \Phi(\vec{b}_i) \approx J^T J$ (это может быть выгодно, так как матрица Якоби и так считается для расчета первой производной, а для шага Ньютона нужны и градиент, и Гессиан).

Данная аппроксимация матрицы Гессса часто используется на практике, так как она не требует вычисления вторых производных

ОФ: -

МЛ: $\nabla_b^2 \Phi(\vec{b}_i) \approx J^T W^{-1} J$

ЛР: $\nabla_b^2 \Phi(\vec{b}_i) = A^T W^{-1} A$

$\vec{p} = -(A^T W^{-1} A)^{-1} A^T W^{-1} (A \vec{b} - \vec{y})$ для линейной регрессии шаг Гаусса-Ньютона не отличается от шага Ньютона так как вторые производные равны нулю.

В методе Гаусса - Ньютона приближенный Гессиан всегда положительно-полуопределенный, но все равно он может оказаться сингулярным или близким к сингулярному, тогда, как мы видели на прошлом семинаре метод потимизации будет неустойчивым.

Оптимизация градиентными методами при регуляризации

Добавим в исходную невязку регуляризующий член в общем виде :

Γ - некоторая матрица, такая что $\Gamma^T \Gamma$ - положительно определенная, α - скалярный регуляризующий множитель

$$\Phi(\vec{b}) = \frac{1}{2} \vec{r}^T W^{-1} \vec{r} = \frac{1}{2} (\vec{f}_b - \vec{y})^T W^{-1} (\vec{f}_b - \vec{y}) + \frac{1}{2} \alpha \vec{b}^T \Gamma^T \Gamma \vec{b}$$

$$\nabla_b \Phi(\vec{b}) = J^T W^{-1} \vec{r} + \alpha \Gamma^T \Gamma \vec{b} \text{ - градиент регуляризованной задачи}$$

Матрица Гессса регуляризованной задачи:

$$\nabla_b^2 \Phi(\vec{b}) = J^T W^{-1} J + \sum_q H(x_q) r_q + \alpha \Gamma^T \Gamma$$

Соответственно, шаг в методе Ньютона:

$$\vec{p}_{\text{newton}} = -(J^T W^{-1} J + H + \alpha \Gamma^T \Gamma)^{-1} (J^T W^{-1} \vec{r} + \alpha \Gamma^T \Gamma \vec{b})$$

Если в этом выражении воспользоваться приближенным гессианом (метод Гаусса-Ньютона) то получим:

$$\vec{p}_{G-N} = -(J^T W^{-1} J + \alpha \Gamma^T \Gamma)^{-1} (J^T W^{-1} \vec{r} + \alpha \Gamma^T \Gamma \vec{b})$$

Как мы помним из предыдущих семинаров (J.Nocedal and S.J.Wright (2006). Numerical optimization, Springer.), чтобы направление шага было в сторону локального минимума, движение алгоритма должно производиться в направлении, вектор которого, образует острый угол с направлением антиградиента. Однако, так как регуляризация вносит дополнительную составляющую в вектор градиента, то при достаточно большом параметре регуляризации, даже метод градиентного поиска может начать идти не в "ту сторону". Это, логично, так как, мы добавили в невязку составляющую $\alpha \Gamma^T \vec{b}$, стремящуюся уменьшить длину вектора параметров, поэтому при достаточно большом параметре регуляризации эта составляющая начнет превалировать над основной $(\vec{f}_b - \vec{y})^T W^{-1} (\vec{f}_b - \vec{y})$ и алгоритм повернется в сторону нуля модуля вектора параметров $\vec{b}^T \vec{b}$.

Почему при расчете матрицы Гессса пренебрегают матрицей вторых производных?

$$\nabla_b^2 \Phi(\vec{b}) = J^T W^{-1} J + H = J^T W^{-1} J + \sum_q H(x_q) r_q, \text{ где } H_{ij}(x_q) = \frac{\partial^2 f}{\partial b_i \partial b_j}(x_q)$$

J - матрциа размером $M \times P$, где M - длина вектора невязки, а P - число переменных оптимизации, обычно $M >> P$, чтобы ее посчитать надо рассчитать P производных в M точках, пусть мы используем конечные разности, вычисление потребует $M(P+1)$ вычислений функции f . Вычисление матрицы вторых производных требует варьирования двух координат вектора параметров, то есть, оно будет стоить уже MP^2 .

Как правило, оказывается выгоднее сделать P шагов в направлении приближенного шага Ньютона, чем рассчитывать матрицу Гессса полностью.

4. Метод Левенберга-Марквардта

Метод ЛМ является достаточно распространенным методом решения задач нелинейной оптимизации, благодаря "встроенной" регуляризации данных метод является одним из наиболее часто используемых (наравне с методом сопряженных градиентов) для решени ОЗ теплопроводности. Поэтому остановимся на нем подробнее.

Идея метода довольно проста, давайте использовать метод Гаусса-Ньютона, но с параметром из метода с регуляризацией, но параметр регуляризации будем подбирать на каждом шаге свой, плюс, так как проблемы с методом Гаусса - Ньютона могут быть только из-за Гессиана, будем подбирать только его.

Шаг алгоритма и регуляризующая матрица

$$\vec{p}_{L-M}^k = -(J^T W^{-1} J + \lambda^k \Omega^k)^{-1} J^T W^{-1} (\vec{f}_b - \vec{y})$$

Где регулирующая положительно-определенная матрица (Ω^k) и параметр λ^k на каждой итерации свои. Добавление положительно-определенной матрицы $\lambda^k \Omega^k$:

1. Делает приближенный Гессиан положительно определенным, а также уменьшает **conditional number**
2. Благодаря тому, что подбор регуляризующего множителя производится внутри метода на текущий итерации производится подбор параметра λ , то есть направления поиска таким образом, делается пробный шаг, как в стратегии **line-search** и смотрится есть ли достаточное уменьшение значения функции, если шагнуть в этом направлении. Так как направление шага при этих пробных шагах меняется, то это уже не линейных поиск (line-search), подобную стратегию поиска следующего шага называют **trust region**.

При решении обратных задач теплопереноса в Л-М используют не единичную матрицу, а диагональ самого приближенного гессиана:

$$\Omega^k = \text{diag}((J^k)^T W^{-1} J^k)$$

То есть, в качестве регуляризующей матрицы используются сами диагональные элементы приближенной матрицы Гесса.

FUNTONS

```

function D = Dt(M)
% M- number of input matrix rows
D = zeros(M-1,M);
D(diagonal_inds(0,M-1,M)) = -1;
D(diagonal_inds(1,M-1,M)) = 1;
end
function D = Dtsym(M)
% M- number of input matrix rows
D = zeros(M-1,M);
D(diagonal_inds(-1,M-1,M)) = -1/2;
D(diagonal_inds(1,M-1,M)) = 1/2;
end
function D = D2t(N)
D = zeros(N,N);
D(diagonal_inds(0,M-1,M)) = -2;
D(diagonal_inds(1,M-1,M)) = 1;
D(diagonal_inds(-1,M-1,M)) = 1;
end
function indices = diagonal_inds(diagonal_index,rows_number, columns_number)
% diagonalIndicesRect returns linear indices of the k-th diagonal of an m x n
matrix
% m - number of rows
% n - number of columns
% k - diagonal number (0 = main, >0 above main, <0 below main)
%
% Linear indices are returned for the elements along the specified diagonal,
% respecting matrix dimensions.

```

```

if diagonal_index >= 0
    % Diagonal above or on main
    len = min(rows_number, columns_number-diagonal_index);
    rowIdx = 1:len;
    colIdx = (1:len) + diagonal_index;
else
    % Diagonal below main
    len = min(rows_number+diagonal_index, columns_number);
    rowIdx = (1:len) - diagonal_index;
    colIdx = 1:len;
end

% Convert row and column indices to linear indices for column-major order
indices = sub2ind([rows_number columns_number], rowIdx, colIdx);
end
function t = chebyshev_nodes(n, tmin, tmax)
    % Chebyshev-Gauss-Lobatto nodes on [-1, 1]: cos(pi * (0:n-1)/(n-1))
    k = (0:n-1)';
    u = cos(pi * k / (n - 1)); % Includes endpoints u=±1

    % Affine map to [tmin, tmax]
    t = tmin + (tmax - tmin) * (u + 1) / 2;
end
function V = cheb_vandermonde(nodes, d)
    % CHEB_VANDERMONDE: Chebyshev basis Vandermonde (well-conditioned)
    n = length(nodes); V = zeros(n, d+1);
    V(:,1) = 1; V(:,2) = nodes;
    for k = 2:d
        V(:,k+1) = 2*nodes .* V(:,k) - V(:,k-1); % T_k recurrence
    end
end
function [f,g] = rosen(x)
f = 100*(x(1) - x(2)^2)^2 + (1 - x(2))^2;
if nargout > 1
    g(1) = 200*(x(1) - x(2)^2);
    g(2) = -400*x(2)*(x(1) - x(2)^2) - 2*(1 - x(2));
end
end

```