Statistical Data Analysis
Discussion notes – week 8

- Problem sheet 5
- Example of Least Squares for averaging
- Comments on multiple regression
- Least Squares with constraints

Problem sheet 5

Exercise 1: For this exercise you will do a simple multivariate analysis either with the python `scikit-learn` package or using C++ with TMVA from ROOT. The input data consists of events of two types: signal and background. Three quantities $\mathbf{x} = (x_1, x_2, x_3)$ are are measured for each event. The marginal distributions of each of the three components are shown in Fig. 1.

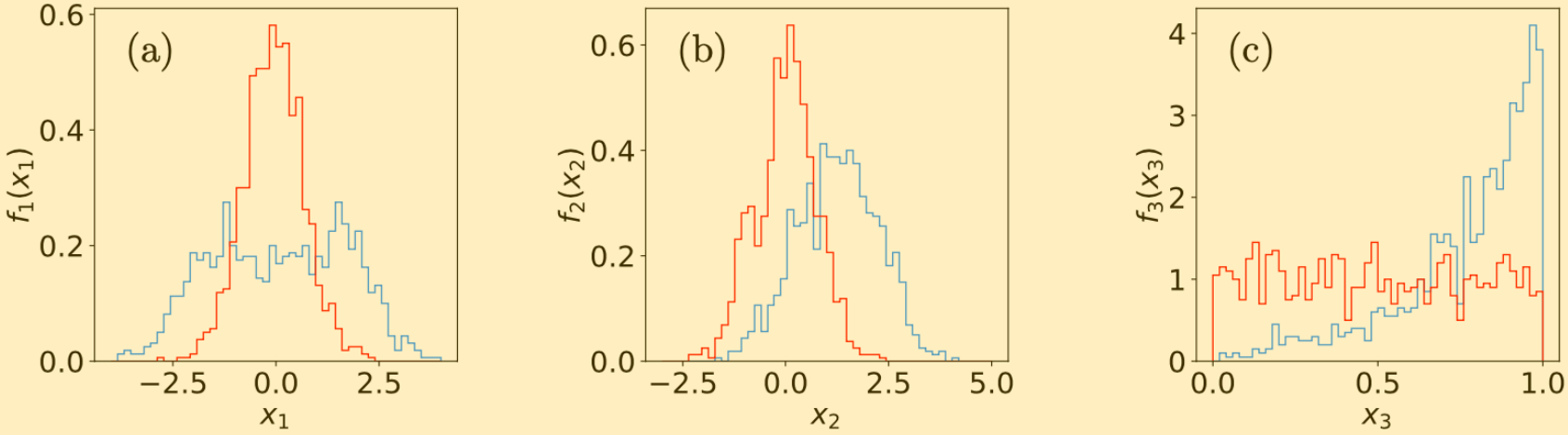
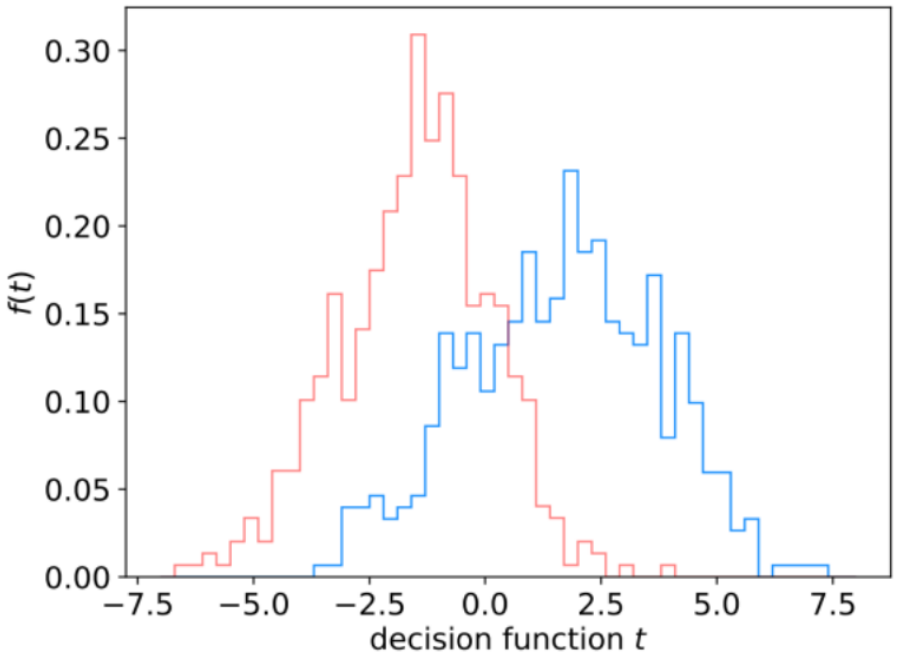


Figure 1: Marginal distributions of the three components of the feature vector $\mathbf{x} = (x_1, x_2, x_3)$ for events of the two classes: signal ($y = 1$, blue) and background ($y = 0$, red).

Problem sheet 5 / 1

Exercise 1: The data files supplied contain events of the two classes, signal and background. Each event is characterized by three quantities, $\mathbf{x} = (x_1, x_2, x_3)$.

The program provided already contains a Fisher discriminant and allows one to evaluate for each event the test statistic $t(\mathbf{x})$. Using this we want to select a sample of events enriched in signal by requiring $t > t_c$ with $t_c = 0$. That is, we test for each event the hypothesis that it is of the background type, and we reject that hypothesis (and thus select as signal) if $t > t_c$.



Problem sheet 5 / 1(a)

1(a) [5 marks] The code you are given calculates already the signal efficiency (the power of the test), i.e., $\varepsilon_{\rm s} = P(t > t_{\rm c}|{\rm s})$. Add the necessary code to find the background efficiency (the size of the test) $\varepsilon_{\rm b} = P(t > t_{\rm c}|{\rm b})$.

For each event we test the hypothesis H_0 : event is of type b

Critical region of test: $t > t_c$

metrics.accuracy_score(y_test, y_pred) takes as input an array of y values (true class labels) y_test (from the test sample) and predicted values y_pred, and returns the fraction of times the prediction is correct, i.e.,

metrics.accuracy_score(y_bkg_test, y_bkg_pred)

= P(reject as b | b) = 1 – P(classify as b | b) = 1 - P(t > t_c | b)

So add the code:

size = 1. - metrics.accuracy_score(y_bkg_test, y_bkg_pred)

For the LDA this gives: size of test of background = 0.23387

Problem sheet 5 / 1(b)

1(b) [3 marks] What is the signal purity of the selected sample, i.e., $P(\mathrm{s}|t > t_\mathrm{c})$? Assume that the two event classes have equal prior probabilities.

$$\text{signal purity} = P(s|t > t_\mathrm{c}) = \frac{P(t > t_\mathrm{c}|s)\pi_s}{P(t > t_\mathrm{c}|s)\pi_s + P(t > t_\mathrm{c}|b)\pi_b}$$

with prior probabilities $\pi_\mathrm{s} = \pi_\mathrm{b} = 0.5$. To get this, add to the code

purity = effSig*piSig / (effSig*piSig + effBkg*piBkg)

where $\texttt{effSig} = \varepsilon_\mathrm{s} = P(t > t_\mathrm{c}|s)$, $\texttt{effBkg} = \varepsilon_\mathrm{s} = P(t > t_\mathrm{c}|b)$.

For the LDA this gives:

power of test with respect to signal = 0.7916

purity of signal sample = 0.7719

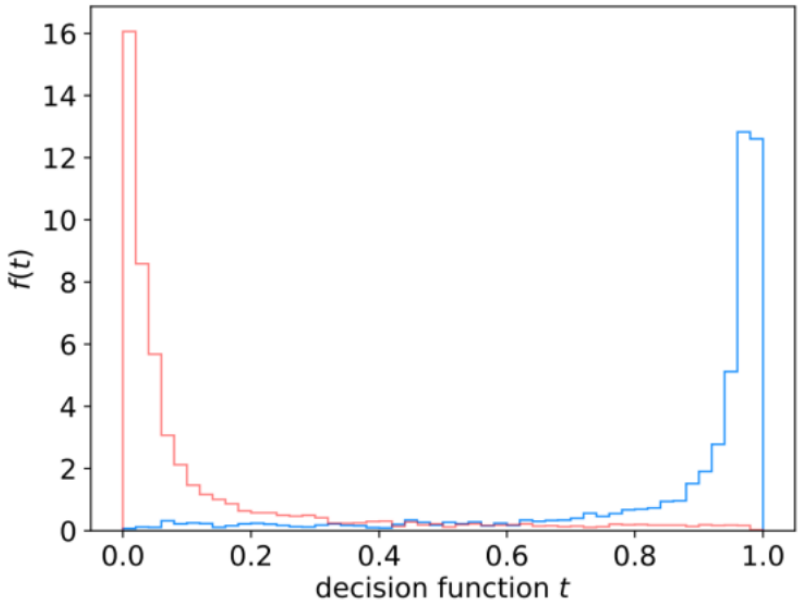
Problem sheet 5 / 1(c)

1(c) [**8 marks**] Modify your program to include a multilayer perceptron with one hidden layer containing 3 nodes. Using the test data samples, make a histogram of the neural network's decision function for both event types. (With scikit-learn, you will need to use instead the output of the function `predict_proba`, which is monotonically related to the MLP output.) Compare to the corresponding histogram from the Fisher discriminant.

Include the code:

```
clf = MLPClassifier(hidden_layer_sizes=(3,), activation='tanh',
max_iter=2000, random_state=0)
```

Histogram of the MLP output (as given by predict_proba):



Problem sheet 5 / 1(d)

1(d) [4 marks] Select signal events by requiring $t_{\rm MLP} > t_{\rm c}$ with $t_{\rm c} = 0.5$. What are the signal and background efficiencies? What is the signal purity assuming equal prior probabilities for the two event types?

To select events with the MLP, the function predict_proba is used instead of the decision function. As in (b), to find the purity, use the efficiencies in Bayes' theorem:

Signal efficiency = $\varepsilon_s = P(t>t_c|s)$
Background efficiency = $\varepsilon b = P(t>t_c|b)$
Signal purity = $P(s|t>t_c) = \varepsilon_s \pi_s /(\varepsilon_s \pi_s + \varepsilon_b \pi_b)$

By defining the critical region with a minimum threshold on its value of 0.5, the program gives:

power of test with respect to signal = 0.8769
size of test of background = 0.1129
purity of signal sample = 0.8859

Problem sheet 5 / 2 (bonus question)

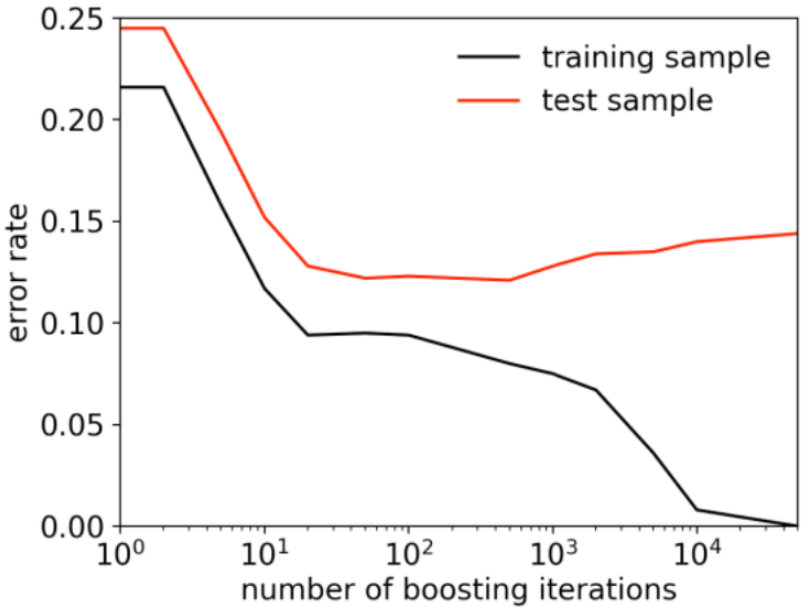
2(a) Modify your program to include a boosted decision tree with 200 boosting iterations.

2(b) Now repeat this for different numbers of boosting iterations, say, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 10,000, 50,000.

For each classifier, compute the total error rate using a boundary value of $t_c = 0$. That is, compute the fraction of events (signal and background) that are found on the wrong side of the boundary. Plot this as a function of the number of boosting iterations for both the training sample and the statistically independent test sample. Determine roughly the optimal number of boosting iterations.

The total error rate of a BDT (using AdaBoost) as a function of the number of boosting iterations.

The error rate from the training sample goes to zero after around 10^4 iterations. For the test sample, a minimum error rate of 12.1% is found at 500 iterations, with a shallow minimum around 12% from 10^2 to 10^3 iterations. For more than 10^3 iterations the error rate increases slowly.



Problem on Least-Squares Averaging

$y_i \sim \text{Gauss}(\lambda, \sigma_i)$ $\quad i = 1, 2$

↳ indep. $\quad\}$ param to be measured.

a) Likelihood function

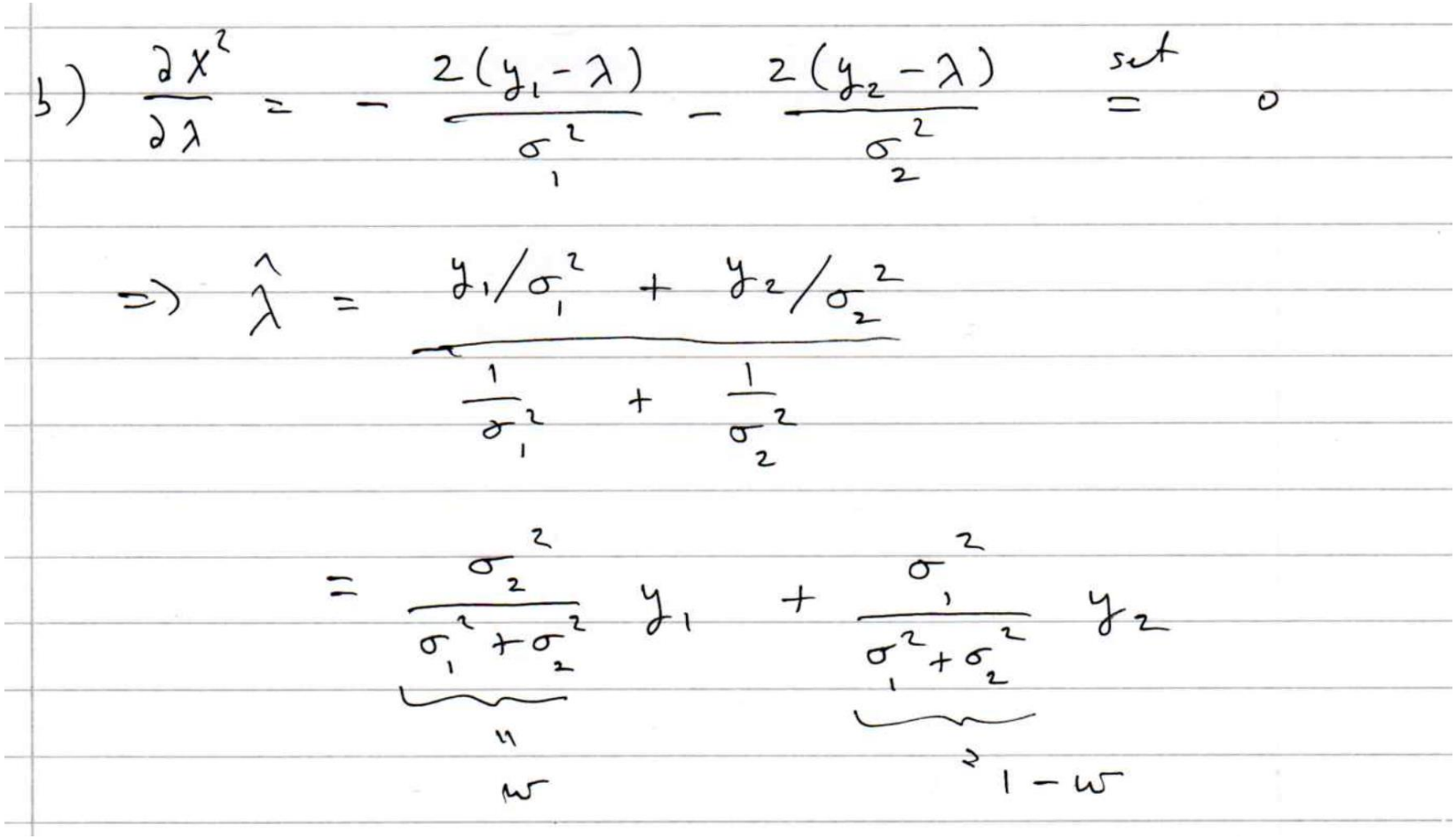
$$L(\lambda) = \prod_{i=1}^{2} \frac{1}{\sqrt{2\pi}\,\sigma_i} e^{-(y_i - \lambda)^2 / 2\sigma_i^2}$$

$$\Rightarrow \ln L(\lambda) = -\frac{1}{2} \sum_{i=1}^{2} \frac{(y_i - \lambda)^2}{\sigma_i^2} + C$$

$$\Rightarrow \text{minimize} \quad \chi^2(\lambda) = \sum_{i=1}^{2} \frac{(y_i - \lambda)^2}{\sigma_i^2}$$

LS averaging (cont.)

Find LS estimator for λ:



LS averaging (cont.)

Find variance of LS estimator:

c) $$E[\hat{\lambda}] = \frac{\frac{E[y_1]}{\sigma_1^2} + \frac{E[y_2]}{\sigma_2^2}}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}}$$, use $E[y_i] = \lambda$, $i = 1, 2$

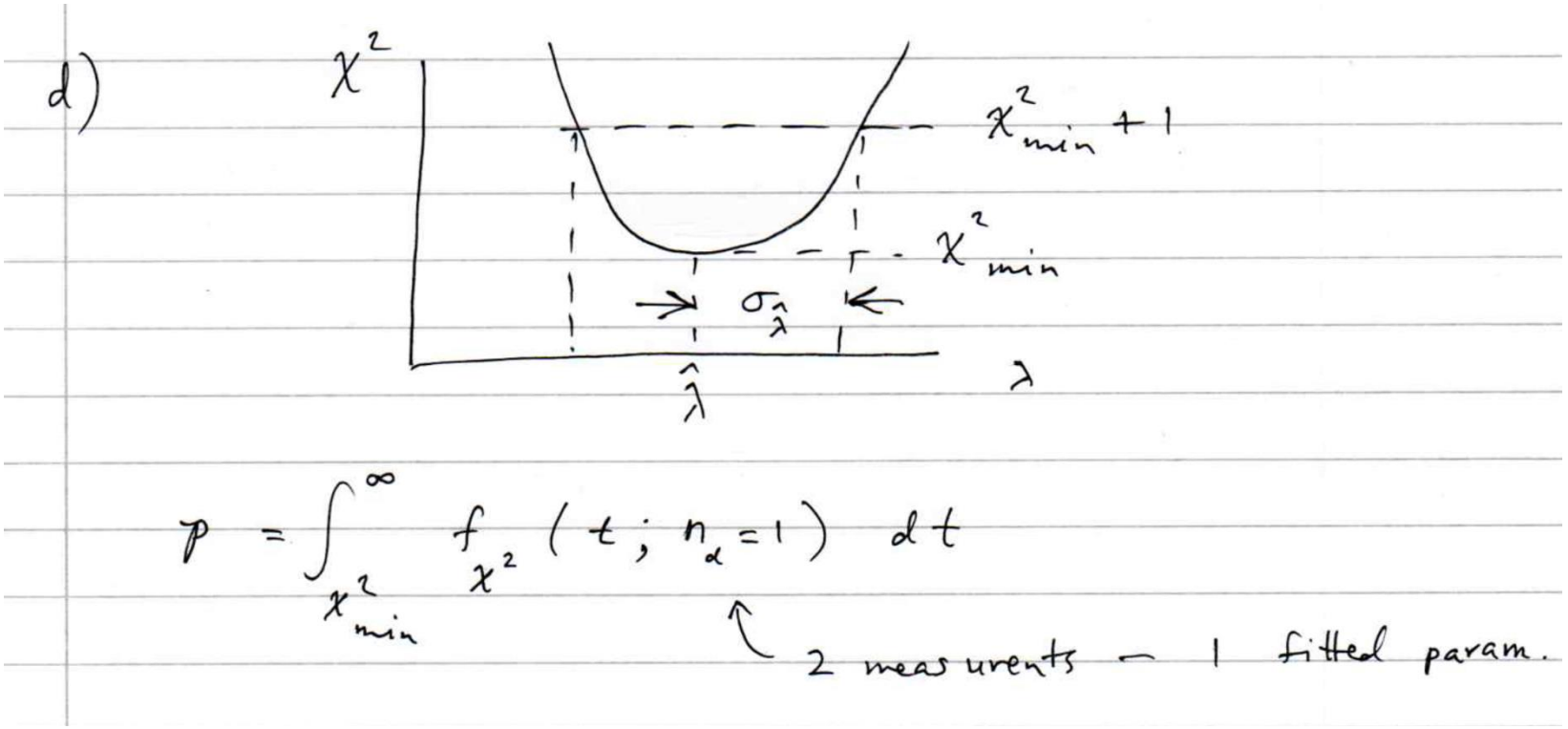
$$= \lambda \Rightarrow \hat{\lambda} \text{ is unbiased.}$$

$$V[\hat{\lambda}] = \frac{1}{\left(\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}\right)^2}\left(\frac{V[y_1]}{\sigma_1^4} + \frac{V[y_2]}{\sigma_2^4}\right) = \frac{1}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}}$$

($V[y_1] \to \sigma_1^2$, $V[y_2] \to \sigma_2^2$)

LS averaging (cont.)

Illustrate graphical method for variance of LS estimator; explain how to find p-value for goodness-of-fit.



LS averaging (cont.)

Find "BLUE" estimator:

e) Suppose $\hat{\lambda} = w y_1 + (1-w) y_2$ for some w

$=$ most general linear unbiased λ

$E[\hat{\lambda}] = w E[y_1] + (1-w) E[y_2] = \lambda \Rightarrow$ unbiased ✓

($E[y_1] \uparrow \lambda$, $E[y_2] \uparrow \lambda$)

$V[\hat{\lambda}] = w^2 \sigma_1^2 + (1-w)^2 \sigma_2^2$ ← adjust w to minimize

$\frac{\partial V[\hat{\lambda}]}{\partial w} = 2w\sigma_1^2 - 2(1-w)\sigma_2^2 \overset{\text{set}}{=} 0$

$\Rightarrow w = \frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2} \Rightarrow$ same as LS!

$\hat{\lambda}$ = "BLUE" (Best Linear Unbiased Estimator)

Simple Least Squares fits

A simple way to do least squares curve fitting is with the python routine **curve_fit**.

For an introduction to this see the the materials from RHUL's year-3 introduction to statistics.

This includes a short program simpleFit.py for doing least-squares fits; also a root/C++ version simpleFit.C.

Fitting the parameters with Python

The routine routine **curve_fit** from **scipy.optimize** can find LS estimators numerically. To use it you need:

```
import numpy as np
from scipy.optimize import curve_fit
```

We need to define the fit function $\mu(x; \theta)$, e.g., a straight line:

```
def func(x, *theta):
    theta0, theta1 = theta
    return theta0 + theta1*x
```

Fitting the parameters with Python (2)

The data values (x_i, y_i, σ_i) need to be in the form of NumPy arrays, e.g,

```
x   = np.array([1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0])
y   = np.array([2.7, 3.9, 5.5, 5.8, 6.5, 6.3, 7.7, 8.5, 8.7])
sig = np.array([0.3, 0.5, 0.7, 0.6, 0.4, 0.3, 0.7, 0.8, 0.5])
```

Start values of the parameters can be specified:

```
p0 = np.array([1.0, 1.0])
```

To find the parameter values that minimize $\chi^2(\theta)$, call **curve_fit**:

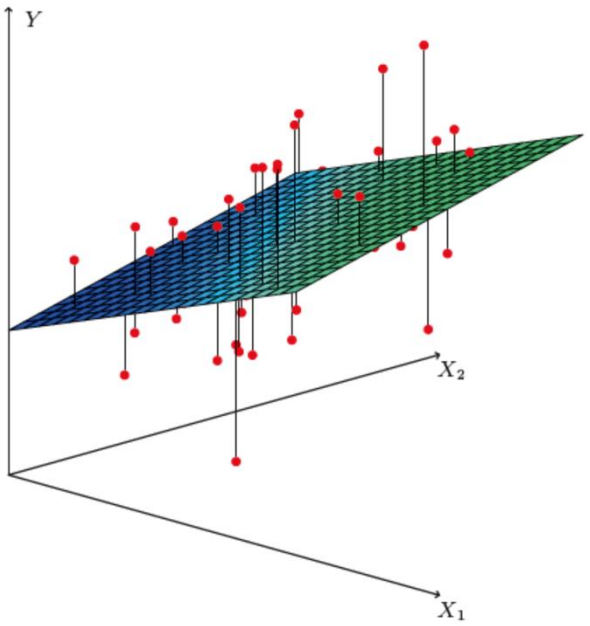
```
thetaHat, cov = curve_fit(func, x, y, p0, sig, absolute_sigma=True)
```

Returns estimators and covariance matrix as NumPy arrays.

Need **absolute_sigma=True** for the fit errors (cov. matrix) to have desired interpretation.

Brief intro to multiple regression

Multiple regression* can be seen as an extension of curve fitting to the case where the variable x is replaced by a multi-dimensional $\boldsymbol{x} = (x_1,...,x_n)$, e.g., fitting a surface. Here suppose the data are points $(\boldsymbol{x}_i, y_i)$, $i = 1,...,N$ (no error bars) and $\boldsymbol{x}$ is usually a random variable, often called the explanatory or predictor variable.



Equivalently, we can view it as an extension to classification with the discrete class label $y = 0, 1$ replaced by a continuous target y (and in this context $\boldsymbol{x}$ can also be called the feature vector).

*Note the term ”multivariate” regression refers to a vector target variable y; here we treat only scalar y.

Target (fit) function and loss function

As in the case of curve fitting, we assume some parametric function of $\boldsymbol{x}$ that represents the mean of the target variable

$$E[y] = f(\mathbf{x}; \mathbf{w})$$

where $\boldsymbol{w}$ is a vector of adjustable parameters ("weights").

Suppose we have training data consisting of $(\boldsymbol{x}_i, y_i)$, $i = 1,\ldots,N$.

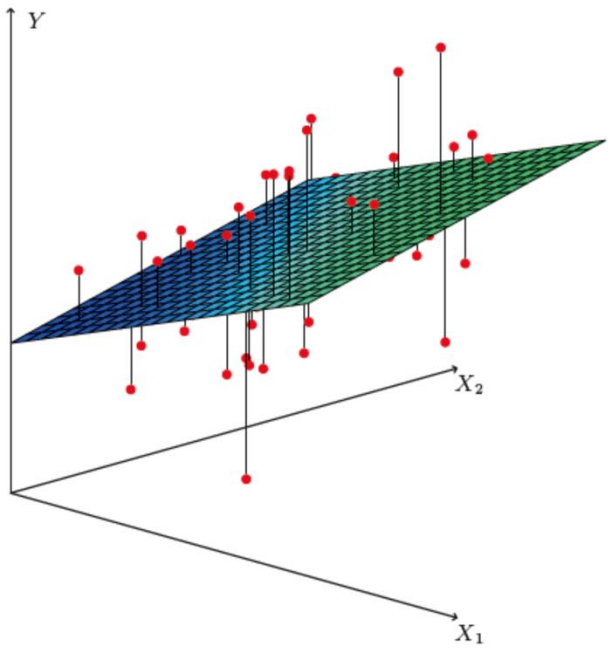
Use these to determine the weights by minimizing a loss function (analogous to the χ^2), e.g.,

$$L(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n} |y_i - f(\mathbf{x}_i; \mathbf{w})|^2$$

Linear regression

In linear regression, the fit function is of the form

$$f(\mathbf{x};\mathbf{w}) = w_0 + \sum_{i=1}^{n} w_i x_i$$



i.e. the problem is equivalent to an unweighted least-squares fit of a (hyper-)plane:

Can be generalized to a nonlinear surface in $\boldsymbol{x}$-space by transforming $\boldsymbol{x}$ to a set of basis functions $\varphi_1(\boldsymbol{x}),\ldots,\varphi_m(\boldsymbol{x})$

$$f(\mathbf{x};\mathbf{w}) = w_0 + \sum_{i=1}^{m} w_i \varphi_i(\mathbf{x})$$

(still linear in the weights)

Nonlinear regression

Examples of nonlinear regression include:

MLP (multilayer perceptron) regression

Boosted decision tree regression

Support vector regression

For MLP regression, as with classification, regard the feature vector as the layer $k = 0$; i.e., $\varphi_i^{(0)} = x_i$.

The ith node of hidden layer k is

$$\varphi_i^{(k)} = h\left(w_{i0}^{(k)} + \sum_{j=1}^{n} w_{ij}^{(k)} \varphi_j^{(k-1)} \right)$$

where h is the activation function (tanh, relu, sigmoid,...).

MLP Regression (cont.)

For the final layer (k=K), in MLP regression (in contrast to classification), one omits the activation function, i.e.,

$$f(\mathbf{x};\mathbf{w}) = w_0^{(K)} + \sum_{j=1}^{n} w_j^{(K)} \varphi_j^{(K-1)}$$

where $\varphi_j^{(K-1)}$ = are the nodes of the last hidden layer ($k = K-1$).

For info on other types of multiple regression see, e.g.,

Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani, *An Introduction to Statistical Learning with Applications in R*, Springer, 2013;

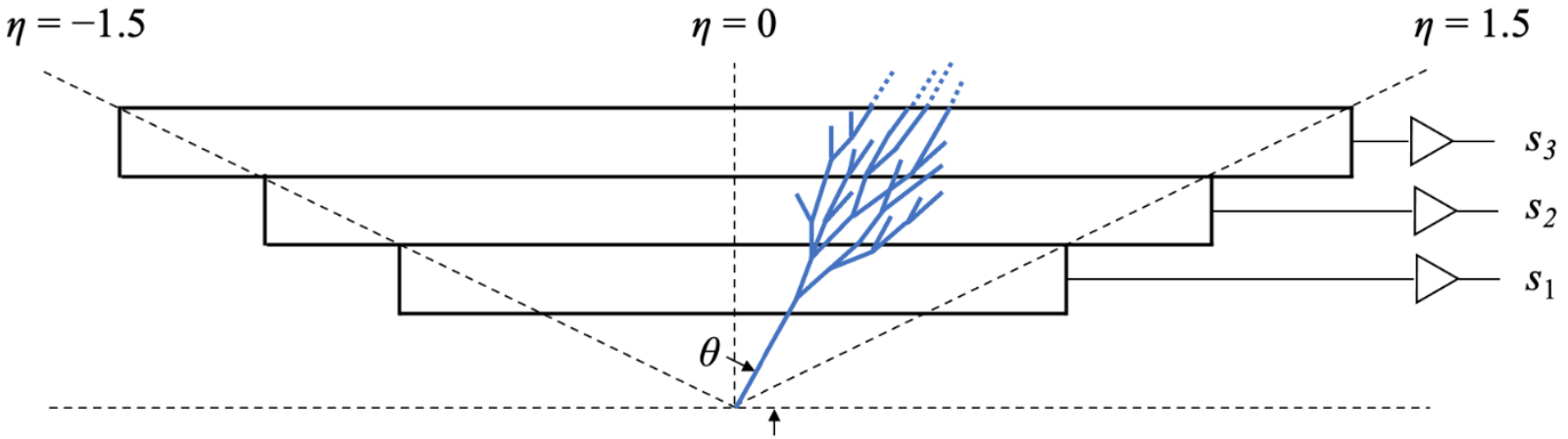
https://www.statlearning.com/

and the scikit-learn documentation.

Multiple regression example

Suppose particles with different energies E and angles θ (or equivalently $\eta = -\ln\tan(\theta/2)$) enter a calorimeter and create a particle showers that gives signals in three layers, s_1, s_2 and s_3, as well as an estimate of η.

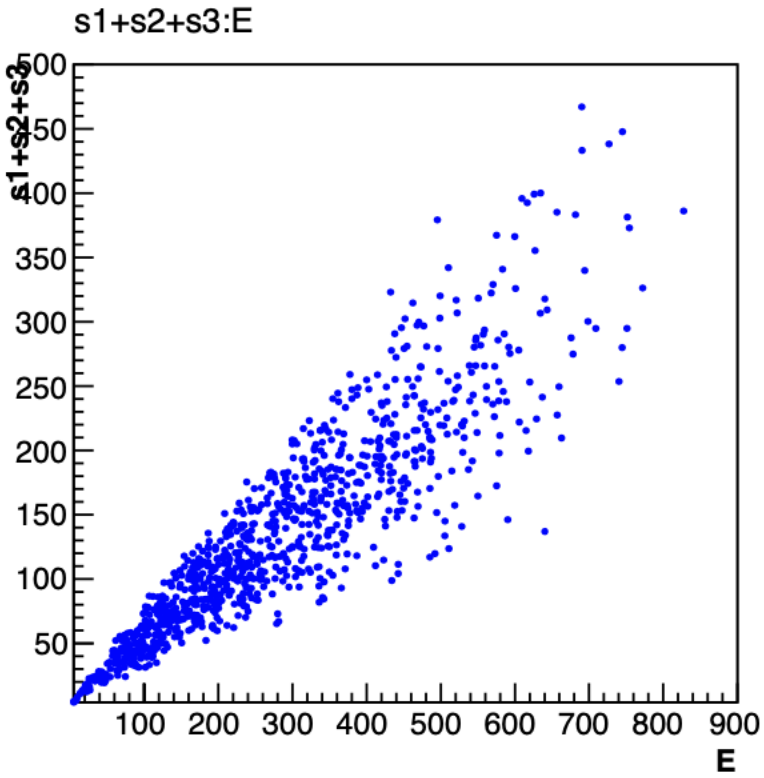
Some of the energy leaks through, with increased leakage for higher energy and more oblique angles (higher η).



The goal is to estimate the target $y_i = E_i$ given feature vectors $\boldsymbol{x}_i = (\eta, s_1, s_2, s_3)_i$ for $i = 1,...,N$ training events.

Energy estimate from sum of signals

Naively, one could try just summing the signals: $\hat{E} = s_1 + s_2 + s_3$

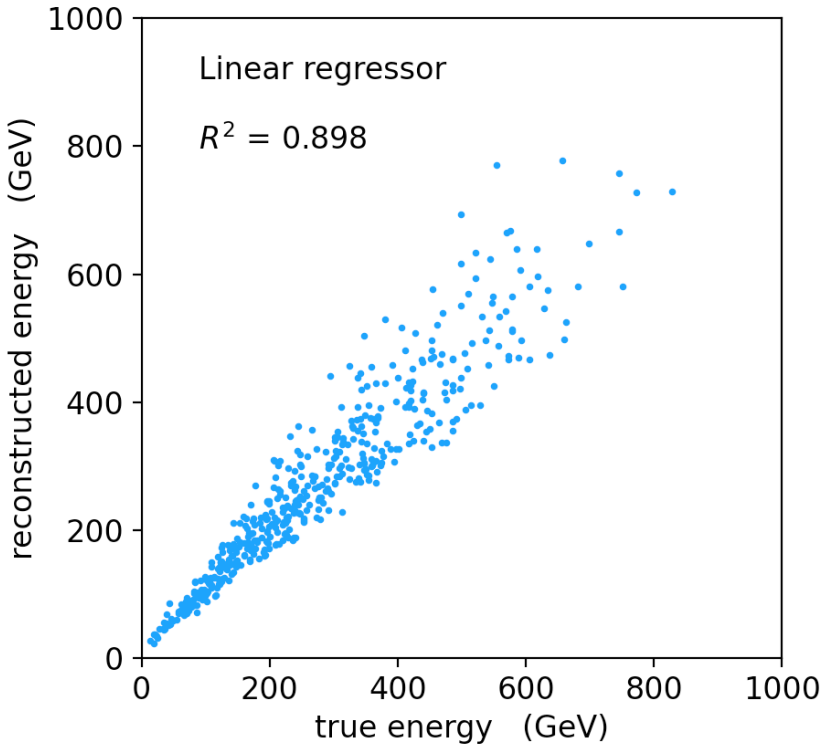


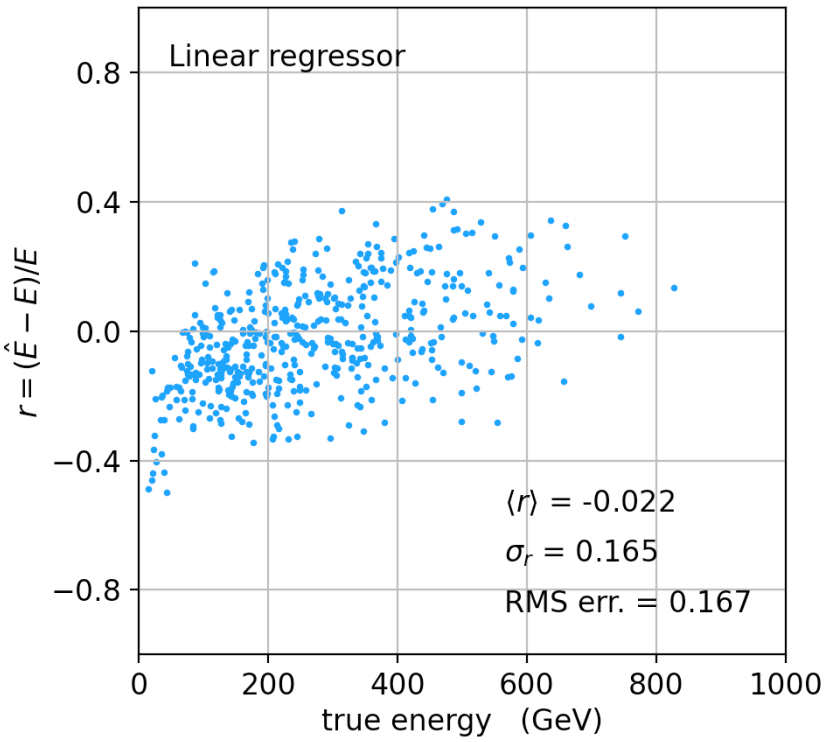
Gives very poor resolution because the particles have a distribution of energies and angles and hence differing amounts of the energy leak through undetected.

Linear regression

See MVRegressor.py, here using

```
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)
```

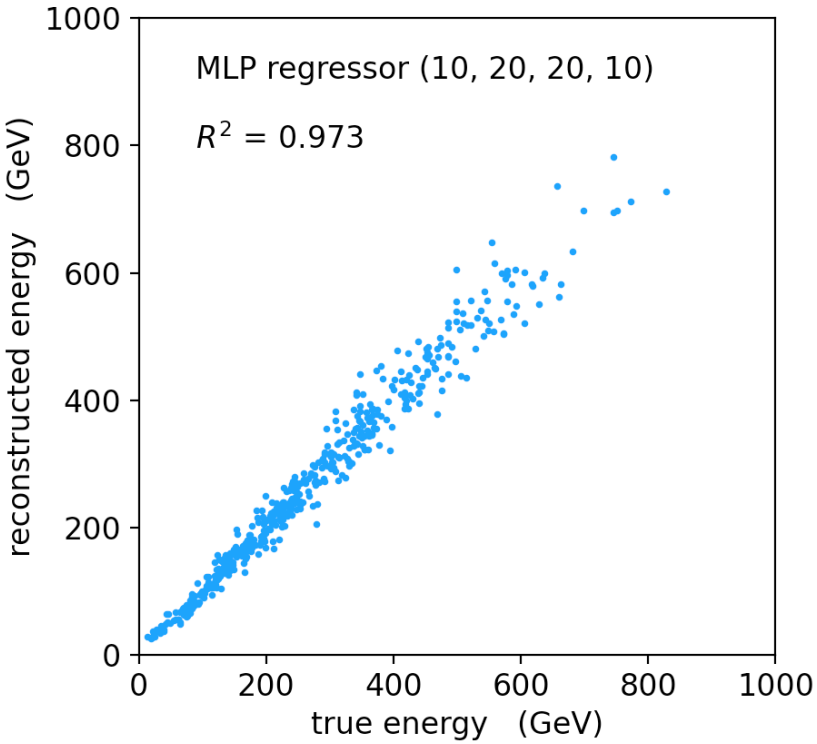


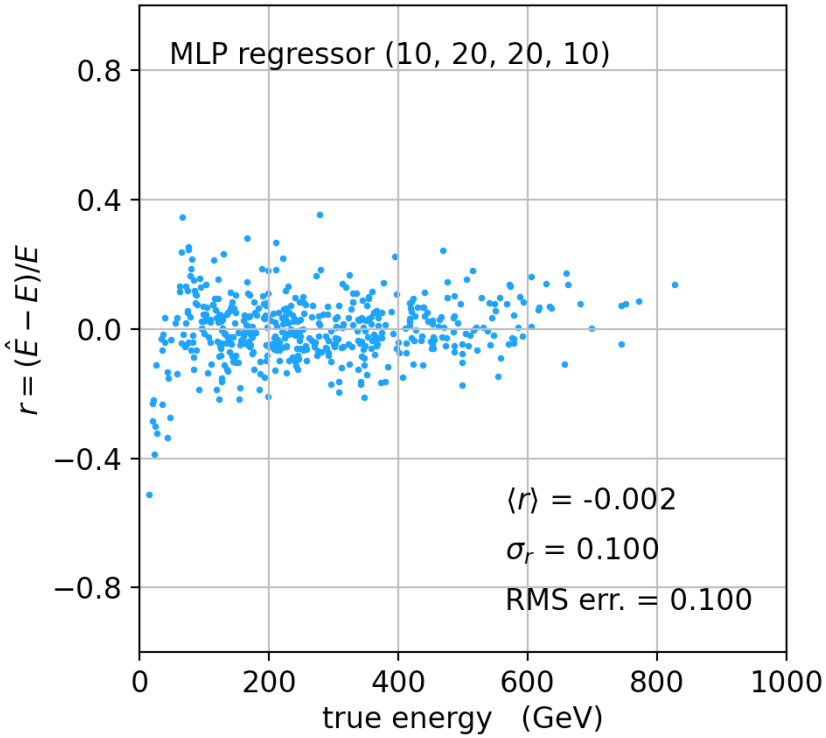


Average relative resolution 16.7%.

MLP Regression

```
regr = MLPRegressor(hidden_layer_sizes=(10,20,20,10), activation='relu'
regr.fit(X_train, y_train)
```





Better resolution (10%), here significant bias at low energies.

Refinements for multiple regression

One can try many improvements:

Scaling of predictor and target variables, e.g., standardize to zero mean and unit variance.

Use cross-validation to assess accuracy (and hence use entire sample of events for training.

Try different loss functions.

Try different regression algorithms (ridge regression, lasso, decision tree, support vector regression,...).

Some simple code using scikit-learn and a short project description can be found here:

https://www.pp.rhul.ac.uk/~cowan/ph3010/ml/regression/

Parameter estimation with constraints

When estimating parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_M)$ one may have additional information available in the form of K constraints

$$c_k(\boldsymbol{\theta}) = 0\,, \quad k = 1, \ldots, K$$

In some problems it may be possible to define $L = M - K$ new parameters $\eta_1, \ldots, \eta_L$ such that every point in $\boldsymbol{\eta}$-space satisfies the constraints. If so, estimate $\boldsymbol{\eta}$ e.g. with Maximum Likelihood or Least Squares and then transform back to $\boldsymbol{\theta}$. But it may be difficult to find new parameters with the required properties.

Suppose the estimators are found by minimizing $\chi^2(\boldsymbol{\theta})$. One can implement the constraints by minimizing instead the Lagrange function

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}, \mathbf{y}) = \chi^2(\boldsymbol{\theta}, \mathbf{y}) + \sum_{k=1}^{K} \lambda_k c_k(\boldsymbol{\theta})$$

with respect to $\boldsymbol{\theta}$ and the Lagrange multipliers $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_K)$.

Finding constrained estimators

Define a $K+M$ dimensional vector to contain the parameters and Lagrange multipliers

$$\boldsymbol{\gamma} = (\theta_1, \ldots, \theta_M, \lambda_1, \ldots, \lambda_K)$$

The estimators for $\boldsymbol{\gamma}$ are found from the solutions to

$$F_i(\boldsymbol{\gamma}, \mathbf{y}) \equiv \frac{\partial \mathcal{L}}{\partial \gamma_i} = 0 \,, \quad i = 1, \ldots, M + K$$

This gives the parameter values that minimize $\chi^2(\boldsymbol{\theta})$ subject to the constraints.

Covariance matrix of estimators

To find the covariance matrix of the estimators, find the solutions $\tilde{\boldsymbol{\gamma}}$ to the equations above when the data $\boldsymbol{y}$ are equal to their expected values $\langle \boldsymbol{y} \rangle$ (in practice estimate with the observed values). This gives estimators

$$\hat{\boldsymbol{\gamma}}(\mathbf{y}) \approx \tilde{\boldsymbol{\gamma}} + C(\mathbf{y} - \langle \mathbf{y} \rangle) \quad \text{where} \quad C = -A^{-1}B$$

and where $$A_{ij} = \left[\frac{\partial F_i}{\partial \gamma_j}\right]_{\tilde{\boldsymbol{\gamma}}, \langle \mathbf{y} \rangle} \quad \text{and} \quad B_{ij} = \left[\frac{\partial F_i}{\partial y_j}\right]_{\tilde{\boldsymbol{\gamma}}, \langle \mathbf{y} \rangle} .$$

Using this approximation for $\hat{\boldsymbol{\gamma}}(\mathbf{y})$, find the covariance matrix $U_{ij} = \mathrm{cov}[\hat{\gamma}_i, \hat{\gamma}_j]$ using error propagation, i.e.,

$$U = CVC^T \quad \text{where} \quad V_{ij} = \mathrm{cov}[y_i, y_j]$$

Derivation of formula for covariance

Starting from the equations $F_i(\boldsymbol{\gamma}, \mathbf{y}) = 0$, $i = 1, \ldots, K + M$, consider two solutions: $\hat{\boldsymbol{\gamma}}$ corresponding to data $\mathbf{y}$ and $\tilde{\boldsymbol{\gamma}}$ corresponding to $\langle \mathbf{y} \rangle$. Expanding $F_i(\hat{\boldsymbol{\gamma}}, \mathbf{y})$ to first order in $\hat{\boldsymbol{\gamma}}$ and $\mathbf{y}$ about $\tilde{\boldsymbol{\gamma}}$ and $\langle \mathbf{y} \rangle$ gives

$$F_i(\mathbf{y}, \hat{\boldsymbol{\gamma}}) \approx F_i(\langle \mathbf{y} \rangle, \tilde{\boldsymbol{\gamma}}) + \sum_{j=1}^{M+K} \left[\frac{\partial F_i}{\partial \gamma_j} \right]_{\langle \mathbf{y} \rangle, \tilde{\boldsymbol{\gamma}}} (\hat{\gamma}_j - \tilde{\gamma}_j) + \sum_{j=1}^{N} \left[\frac{\partial F_i}{\partial y_j} \right]_{\langle \mathbf{y} \rangle, \tilde{\boldsymbol{\gamma}}} (y_j - \langle y_j \rangle) \, .$$

The terms $F_i(\mathbf{y}, \hat{\boldsymbol{\gamma}})$ and $F_i(\langle \mathbf{y} \rangle, \tilde{\boldsymbol{\gamma}})$ are both zero because both pairs of arguments are assumed to be solutions to $F_i = 0$. Dropping these terms, the equation can be rewritten in matrix form $\hat{\boldsymbol{\gamma}} \approx \tilde{\boldsymbol{\gamma}} + C(\mathbf{y} - \langle \mathbf{y} \rangle)$, where $C = -A^{-1}B$.

For more details see the PDG review on statistics Sec. 40.2.4 at **pdg.lbl.gov** or the note:

https://www.pp.rhul.ac.uk/~cowan/stat/notes/lscon.pdf

Example of constrained estimators

Suppose we have measurements y_1, y_2 and y_3 of the three angles θ_1, θ_2, θ_3 of a triangle.

Model as independent and Gaussian: $y_i \sim \text{Gauss}(\theta_i, \sigma)$.

To find the estimators, one could replace $\theta_3 = \pi - \theta_1 - \theta_2$ and minimize $\chi^2(\theta_1,\theta_2)$.

Alternatively, minimize $\mathcal{L}(\boldsymbol{\theta}, \lambda) = \sum_{i=1}^{3} \frac{(y_i - \theta_i)^2}{\sigma^2} + \lambda(\theta_1 + \theta_2 + \theta_3 - \pi)$

$\rightarrow$

$$\hat{\theta}_1 = \frac{1}{3}(2y_1 - y_2 - y_3 + \pi) \qquad \hat{\theta}_2 = \frac{1}{3}(-y_1 + 2y_2 - y_3 + \pi)$$

$$\hat{\theta}_3 = \frac{1}{3}(-y_1 - y_2 + 2y_3 + \pi) \qquad \hat{\lambda} = \frac{2}{3\sigma^2}(y_1 + y_2 + y_3 - \pi)$$

Variances of estimates reduced by constraint: $V[\hat{\theta}_i] = \frac{2}{3}\sigma^2$, $i = 1, 2, 3$