Data Analysis and Machine Learning: Linear Regression and more Advanced Regression Analysis

Morten Hjorth-Jensen[1,2]

[1]Department of Physics, University of Oslo
[2]Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State University

May 17, 2019

Why Linear Regression (aka Ordinary Least Squares and family)

Fitting a continuous function with linear parameterization in terms of the parameters $\boldsymbol{\beta}$.

- Method of choice for fitting a continuous function!
- Gives an excellent introduction to central Machine Learning features with **understandable pedagogical** links to other methods like **Neural Networks**, **Support Vector Machines** etc
- Analytical expression for the fitting parameters $\boldsymbol{\beta}$
- Analytical expressions for statistical propertiers like mean values, variances, confidence intervals and more
- Analytical relation with probabilistic interpretations
- Easy to introduce basic concepts like bias-variance tradeoff, cross-validation, resampling and regularization techniques and many other ML topics
- Easy to code! And links well with classification problems and logistic regression and neural networks
- Allows for **easy** hands-on understanding of gradient descent methods
- and many more features

Regression analysis, overarching aims

Regression modeling deals with the description of the sampling distribution of a given random variable y and how it varies as function of another variable or a set of such variables $\boldsymbol{x} = [x_0, x_1, \dots, x_{n-1}]^T$. The first variable is called the **dependent**, the **outcome** or the **response** variable while the set of variables $\boldsymbol{x}$ is called the independent variable, or the predictor variable or the explanatory variable.

A regression model aims at finding a likelihood function $p(\boldsymbol{y}|\boldsymbol{x})$, that is the conditional distribution for $\boldsymbol{y}$ with a given $\boldsymbol{x}$. The estimation of $p(\boldsymbol{y}|\boldsymbol{x})$ is made using a data set with

- n cases $i = 0, 1, 2, \dots, n-1$
- Response (target, dependent or outcome) variable y_i with $i = 0, 1, 2, \dots, n-1$
- p so-called explanatory (independent or predictor) variables $\boldsymbol{x}_i = [x_{i0}, x_{i1}, \dots, x_{ip-1}]$ with $i = 0, 1, 2, \dots, n-1$ and explanatory variables running from 0 to $p-1$. See below for more explicit examples.

The goal of the regression analysis is to extract/exploit relationship between $\boldsymbol{y}$ and $\boldsymbol{X}$ in or to infer causal dependencies, approximations to the likelihood functions, functional relationships and to make predictions, making fits and many other things.

Regression analysis, overarching aims II

Consider an experiment in which p characteristics of n samples are measured. The data from this experiment, for various explanatory variables p are normally represented by a matrix $\mathbf{X}$.

The matrix $\mathbf{X}$ is called the *design matrix*. Additional information of the samples is available in the form of $\boldsymbol{y}$ (also as above). The variable $\boldsymbol{y}$ is generally referred to as the *response variable*. The aim of regression analysis is to explain $\boldsymbol{y}$ in terms of $\boldsymbol{X}$ through a functional relationship like $y_i = f(\mathbf{X}_{i,*})$. When no prior knowledge on the form of $f(\cdot)$ is available, it is common to assume a linear relationship between $\boldsymbol{X}$ and $\boldsymbol{y}$. This assumption gives rise to the *linear regression model* where $\boldsymbol{\beta} = [\beta_0, \dots, \beta_{p-1}]^T$ are the *regression parameters*.

Linear regression gives us a set of analytical equations for the parameters β_j.

Examples

In order to understand the relation among the predictors p, the set of data n and the target (outcome, output etc) $\boldsymbol{y}$, consider the model we discussed for describing nuclear binding energies.

There we assumed that we could parametrize the data using a polynomial approximation based on the liquid drop model. Assuming

$$BE(A) = a_0 + a_1 A + a_2 A^{2/3} + a_3 A^{-1/3} + a_4 A^{-1},$$

we have five predictors, that is the intercept, the A dependent term, the $A^{2/3}$ term and the $A^{-1/3}$ and A^{-1} terms. This gives $p = 0, 1, 2, 3, 4$. Furthermore we have n entries for each predictor. It means that our design matrix is a $p \times n$ matrix $\boldsymbol{X}$.

Here the predictors are based on a model we have made. A popular data set which is widely encountered in ML applications is the so-called credit card default data from Taiwan. The data set contains data on $n = 30000$ credit card holders with predictors like gender, marital status, age, profession, education, etc. In total there are 24 such predictors or attributes leading to a design matrix of dimensionality 24×30000

General linear models

Before we proceed let us study a case from linear algebra where we aim at fitting a set of data $\boldsymbol{y} = [y_0, y_1, \dots, y_{n-1}]$. We could think of these data as a result of an experiment or a complicated numerical experiment. These data are functions of a series of variables $\boldsymbol{x} = [x_0, x_1, \dots, x_{n-1}]$, that is $y_i = y(x_i)$ with $i = 0, 1, 2, \dots, n-1$. The variables x_i could represent physical quantities like time, temperature, position etc. We assume that $y(x)$ is a smooth function.

Since obtaining these data points may not be trivial, we want to use these data to fit a function which can allow us to make predictions for values of y which are not in the present set. The perhaps simplest approach is to assume we can parametrize our function in terms of a polynomial of degree $n-1$ with n points, that is

$$y = y(x) \rightarrow y(x_i) = \tilde{y}_i + \epsilon_i = \sum_{j=0}^{n-1} \beta_j x_i^j + \epsilon_i,$$

where ϵ_i is the error in our approximation.

Rewriting the fitting procedure as a linear algebra problem

For every set of values y_i, x_i we have thus the corresponding set of equations

$$\begin{aligned}
y_0 &= \beta_0 + \beta_1 x_0^1 + \beta_2 x_0^2 + \cdots + \beta_{n-1} x_0^{n-1} + \epsilon_0 \\
y_1 &= \beta_0 + \beta_1 x_1^1 + \beta_2 x_1^2 + \cdots + \beta_{n-1} x_1^{n-1} + \epsilon_1 \\
y_2 &= \beta_0 + \beta_1 x_2^1 + \beta_2 x_2^2 + \cdots + \beta_{n-1} x_2^{n-1} + \epsilon_2 \\
&\dots\dots \\
y_{n-1} &= \beta_0 + \beta_1 x_{n-1}^1 + \beta_2 x_{n-1}^2 + \cdots + \beta_{n-1} x_{n-1}^{n-1} + \epsilon_{n-1}.
\end{aligned}$$

Rewriting the fitting procedure as a linear algebra problem, more details

Defining the vectors

$$\boldsymbol{y} = [y_0, y_1, y_2, \dots, y_{n-1}]^T,$$

and

$$\boldsymbol{\beta} = [\beta_0, \beta_1, \beta_2, \dots, \beta_{n-1}]^T,$$

and

$$\boldsymbol{\epsilon} = [\epsilon_0, \epsilon_1, \epsilon_2, \dots, \epsilon_{n-1}]^T,$$

and the design matrix

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_0^1 & x_0^2 & \dots & \dots & x_0^{n-1} \\ 1 & x_1^1 & x_1^2 & \dots & \dots & x_1^{n-1} \\ 1 & x_2^1 & x_2^2 & \dots & \dots & x_2^{n-1} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & x_{n-1}^1 & x_{n-1}^2 & \dots & \dots & x_{n-1}^{n-1} \end{bmatrix}$$

we can rewrite our equations as

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}.$$

The above design matrix is called a Vandermonde matrix.

Generalizing the fitting procedure as a linear algebra problem

We are obviously not limited to the above polynomial expansions. We could replace the various powers of x with elements of Fourier series or instead of x_i^j we could have $\cos(jx_i)$ or $\sin(jx_i)$, or time series or other orthogonal functions. For every set of values y_i, x_i we can then generalize the equations to

$$\begin{aligned} y_0 &= \beta_0 x_{00} + \beta_1 x_{01} + \beta_2 x_{02} + \cdots + \beta_{n-1} x_{0n-1} + \epsilon_0 \\ y_1 &= \beta_0 x_{10} + \beta_1 x_{11} + \beta_2 x_{12} + \cdots + \beta_{n-1} x_{1n-1} + \epsilon_1 \\ y_2 &= \beta_0 x_{20} + \beta_1 x_{21} + \beta_2 x_{22} + \cdots + \beta_{n-1} x_{2n-1} + \epsilon_2 \\ &\dots\dots \\ y_i &= \beta_0 x_{i0} + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_{n-1} x_{in-1} + \epsilon_i \\ &\dots\dots \\ y_{n-1} &= \beta_0 x_{n-1,0} + \beta_1 x_{n-1,2} + \beta_2 x_{n-1,2} + \cdots + \beta_{n-1} x_{n-1,n-1} + \epsilon_{n-1}. \end{aligned}$$

Note that we have $p = n$ here. The matrix is symmetric. This is generally not the case!

Generalizing the fitting procedure as a linear algebra problem

We redefine in turn the matrix $\boldsymbol{X}$ as

$$\boldsymbol{X} = \begin{bmatrix} x_{00} & x_{01} & x_{02} & \dots & \dots & x_{0,n-1} \\ x_{10} & x_{11} & x_{12} & \dots & \dots & x_{1,n-1} \\ x_{20} & x_{21} & x_{22} & \dots & \dots & x_{2,n-1} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ x_{n-1,0} & x_{n-1,1} & x_{n-1,2} & \dots & \dots & x_{n-1,n-1} \end{bmatrix}$$

and without loss of generality we rewrite again our equations as

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}.$$

The left-hand side of this equation is kwown. Our error vector $\boldsymbol{\epsilon}$ and the parameter vector $\boldsymbol{\beta}$ are our unknow quantities. How can we obtain the optimal set of β_i values?

Optimizing our parameters

We have defined the matrix $\boldsymbol{X}$ via the equations

$$\begin{aligned} y_0 &= \beta_0 x_{00} + \beta_1 x_{01} + \beta_2 x_{02} + \cdots + \beta_{n-1} x_{0n-1} + \epsilon_0 \\ y_1 &= \beta_0 x_{10} + \beta_1 x_{11} + \beta_2 x_{12} + \cdots + \beta_{n-1} x_{1n-1} + \epsilon_1 \\ y_2 &= \beta_0 x_{20} + \beta_1 x_{21} + \beta_2 x_{22} + \cdots + \beta_{n-1} x_{2n-1} + \epsilon_1 \\ &\dots\dots \\ y_i &= \beta_0 x_{i0} + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_{n-1} x_{in-1} + \epsilon_1 \\ &\dots\dots \\ y_{n-1} &= \beta_0 x_{n-1,0} + \beta_1 x_{n-1,2} + \beta_2 x_{n-1,2} + \cdots + \beta_{n-1} x_{n-1,n-1} + \epsilon_{n-1}. \end{aligned}$$

As we noted above, we stayed with a system with the design matrix $\boldsymbol{X} \in \mathbb{R}^{n\times n}$, that is we have $p = n$. For reasons to come later (algorithmic arguments) we will hereafter define our matrix as $\boldsymbol{X} \in \mathbb{R}^{n\times p}$, with the predictors refering to the column numbers and the entries n being the row elements.

Our model for the nuclear binding energies

In our introductory notes we looked at the so-called liguid drop model. Let us remind ourselves about what did by looking at the code.

We restate the parts of the code we are most interested in.

With $\boldsymbol{\beta} \in \mathbb{R}^{p\times 1}$, it means that we will hereafter write our equations for the approximation as

$$\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta},$$

throughout these lectures.

Optimizing our parameters, more details

With the above we use the design matrix to define the approximation $\tilde{\boldsymbol{y}}$ via the unknown quantity $\boldsymbol{\beta}$ as

$$\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta},$$

and in order to find the optimal parameters β_i instead of solving the above linear algebra problem, we define a function which gives a measure of the spread between the values y_i (which represent hopefully the exact values) and the parametrized values $\tilde{y}_i$, namely

$$Q(\boldsymbol{\beta}) = \sum_{i=0}^{n-1} \left(y_i - \tilde{y}_i\right)^2 = \left(\boldsymbol{y} - \tilde{\boldsymbol{y}}\right)^T \left(\boldsymbol{y} - \tilde{\boldsymbol{y}}\right),$$

or using the matrix $\boldsymbol{X}$ and in a more compact matrix-vector notation as

$$Q(\boldsymbol{\beta}) = \left(\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{\beta}\right)^T \left(\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{\beta}\right).$$

Note: It is normal to have a term $1/n$ in front of this equation since it can then be linked to statistical properties like the variance of a given variable. We come back to this in our discussion of the so-called bias-variance trade-off. It is also common to define the function Q as

$$Q(\boldsymbol{\beta}) = \frac{1}{2n} \sum_{i=0}^{n-1} \left(y_i - \tilde{y}_i\right)^2,$$

since when taking the first derivative with respect to the unknown parameters β, the factor of 2 cancels out.

Interpretations and optimizing our parameters

The function

$$Q(\boldsymbol{\beta}) = \left(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\right)^T \left(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\right),$$

can be linked to the variance of the quantity y_i if we interpret the latter as the mean value. When linking below with the maximum likelihood approach below, we will indeed interpret y_i as a mean value (see exercises)

$$y_i = \langle y_i \rangle = \beta_0 x_{i,0} + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \cdots + \beta_{n-1} x_{i,n-1} + \epsilon_i,$$

where $\langle y_i \rangle$ is the mean value. Keep in mind also that till now we have treated y_i as the exact value. Normally, the response (dependent or outcome) variable y_i the outcome of a numerical experiment or another type of experiment and is thus only an approximation to the true value. It is then always accompanied by an error estimate, often limited to a statistical error estimate given by the standard deviation discussed earlier. In the discussion here we will treat y_i as our exact value for the response variable.

In order to find the parameters β_i we will then minimize the spread of $Q(\boldsymbol{\beta})$ by requiring

$$\frac{\partial Q(\boldsymbol{\beta})}{\partial \beta_j} = \frac{\partial}{\partial \beta_j}\left[\sum_{i=0}^{n-1}\left(y_i - \beta_0 x_{i,0} - \beta_1 x_{i,1} - \beta_2 x_{i,2} - \cdots - \beta_{n-1} x_{i,n-1}\right)^2\right] = 0,$$

which results in

$$\frac{\partial Q(\boldsymbol{\beta})}{\partial \beta_j} = -2\left[\sum_{i=0}^{n-1} x_{ij}\left(y_i - \beta_0 x_{i,0} - \beta_1 x_{i,1} - \beta_2 x_{i,2} - \cdots - \beta_{n-1} x_{i,n-1}\right)\right] = 0,$$

or in a matrix-vector form as

$$\frac{\partial Q(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = 0 = \boldsymbol{X}^T\left(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\right).$$

Interpretations and optimizing our parameters

We can rewrite

$$\frac{\partial Q(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = 0 = \boldsymbol{X}^T\left(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\right),$$

as

$$\boldsymbol{X}^T\boldsymbol{y} = \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta},$$

and if the matrix $\boldsymbol{X}^T\boldsymbol{X}$ is invertible we have the solution

$$\boldsymbol{\beta} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}.$$

We note also that since our design matrix is defined as $\boldsymbol{X} \in \mathbb{R}^{n\times p}$, the product $\boldsymbol{X}^T\boldsymbol{X} \in \mathbb{R}^{p\times p}$. In the above case we have that $p \ll n$, in our case $p = 5$ meaning that we end up with inverting a small 5×5 matrix. This is a rather common situation, in many cases we end up with low-dimensional matrices to invert. The methods discussed here and for many other supervised learning algorithms like classification with logistic regression or support vector machines, exhibit dimensionalities which allow for the usage of direct linear algebra methods such as **LU** decomposition or **Singular Value Decomposition** (SVD) for finding the inverse of the matrix $\boldsymbol{X}^T\boldsymbol{X}$.

Interpretations and optimizing our parameters

The residuals $\boldsymbol{\epsilon}$ are in turn given by

$$\boldsymbol{\epsilon} = \boldsymbol{y} - \tilde{\boldsymbol{y}} = \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta},$$

and with

$$\boldsymbol{X}^T\left(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\right) = 0,$$

we have

$$\boldsymbol{X}^T\boldsymbol{\epsilon} = \boldsymbol{X}^T\left(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\right) = 0,$$

meaning that the solution for $\boldsymbol{\beta}$ is the one which minimizes the residuals. Later we will link this with the maximum likelihood approach.

Let us now return to our nuclear binding energies and simply code the above equations.

Own code for Ordinary Least Squares

It is rather straightforward to implement the matrix inversion and obtain the parameters $\boldsymbol{\beta}$. After having defined the matrix $\boldsymbol{X}$ we simply need to write Alternatively, you can use the least squares functionality in **Numpy** as

And finally we plot our fit with and compare with data

Adding error analysis and training set up

We can easily test our fit by computing the $R2$ score that we discussed in connection with the functionality of $_Scikit_Learn_intheintroductoryslides.Sincewearenotusing_Scikit-Learnherewecandefineourown\mathrm{R2}functionasandwewouldbeusingitas$

We can easily add our **MSE** score as and finally the relative error as

The χ^2 function

Normally, the response (dependent or outcome) variable y_i is the outcome of a numerical experiment or another type of experiment and is thus only an approximation to the true value. It is then always accompanied by an error estimate, often limited to a statistical error estimate given by the standard deviation discussed earlier. In the discussion here we will treat y_i as our exact value for the response variable.

Introducing the standard deviation σ_i for each measurement y_i, we define now the χ^2 function (omitting the $1/n$ term) as

$$\chi^2(\boldsymbol{\beta}) = \sum_{i=0}^{n-1} \frac{\left(y_i - \tilde{y}_i\right)^2}{\sigma_i^2} = \left(\boldsymbol{y} - \tilde{\boldsymbol{y}}\right)^T \frac{1}{\boldsymbol{\Sigma^2}} \left(\boldsymbol{y} - \tilde{\boldsymbol{y}}\right),$$

where the matrix $\boldsymbol{\Sigma}$ is a diagonal matrix with σ_i as matrix elements.

The χ^2 function

In order to find the parameters β_i we will then minimize the spread of $\chi^2(\boldsymbol{\beta})$ by requiring

$$\frac{\partial \chi^2(\boldsymbol{\beta})}{\partial \beta_j} = \frac{\partial}{\partial \beta_j}\left[\sum_{i=0}^{n-1}\left(\frac{y_i - \beta_0 x_{i,0} - \beta_1 x_{i,1} - \beta_2 x_{i,2} - \cdots - \beta_{n-1} x_{i,n-1}}{\sigma_i}\right)^2\right] = 0,$$

which results in

$$\frac{\partial \chi^2(\boldsymbol{\beta})}{\partial \beta_j} = -2\left[\sum_{i=0}^{n-1} \frac{x_{ij}}{\sigma_i}\left(\frac{y_i - \beta_0 x_{i,0} - \beta_1 x_{i,1} - \beta_2 x_{i,2} - \cdots - \beta_{n-1} x_{i,n-1}}{\sigma_i}\right)\right] = 0,$$

or in a matrix-vector form as

$$\frac{\partial \chi^2(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = 0 = \boldsymbol{A}^T\left(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{\beta}\right).$$

where we have defined the matrix $\boldsymbol{A} = \boldsymbol{X}/\boldsymbol{\Sigma}$ with matrix elements $a_{ij} = x_{ij}/\sigma_i$ and the vector $\boldsymbol{b}$ with elements $b_i = y_i/\sigma_i$.

The χ^2 function

We can rewrite

$$\frac{\partial \chi^2(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = 0 = \boldsymbol{A}^T\left(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{\beta}\right),$$

as

$$\boldsymbol{A}^T\boldsymbol{b} = \boldsymbol{A}^T\boldsymbol{A}\boldsymbol{\beta},$$

and if the matrix $\boldsymbol{A}^T\boldsymbol{A}$ is invertible we have the solution

$$\boldsymbol{\beta} = \left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T\boldsymbol{b}.$$

The χ^2 function

If we then introduce the matrix

$$\boldsymbol{H} = \left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1},$$

we have then the following expression for the parameters β_j (the matrix elements of $\boldsymbol{H}$ are h_{ij})

$$\beta_j = \sum_{k=0}^{p-1} h_{jk}\sum_{i=0}^{n-1}\frac{y_i}{\sigma_i}\frac{x_{ik}}{\sigma_i} = \sum_{k=0}^{p-1} h_{jk}\sum_{i=0}^{n-1} b_i a_{ik}$$

We state without proof the expression for the uncertainty in the parameters β_j as (we leave this as an exercise)

$$\sigma^2(\beta_j) = \sum_{i=0}^{n-1}\sigma_i^2\left(\frac{\partial \beta_j}{\partial y_i}\right)^2,$$

resulting in

$$\sigma^2(\beta_j) = \left(\sum_{k=0}^{p-1} h_{jk}\sum_{i=0}^{n-1} a_{ik}\right)\left(\sum_{l=0}^{p-1} h_{jl}\sum_{m=0}^{n-1} a_{ml}\right) = h_{jj}!$$

The χ^2 function

The first step here is to approximate the function y with a first-order polynomial, that is we write

$$y = y(x) \rightarrow y(x_i) \approx \beta_0 + \beta_1 x_i.$$

By computing the derivatives of χ^2 with respect to β_0 and β_1 show that these are given by

$$\frac{\partial \chi^2(\boldsymbol{\beta})}{\partial \beta_0} = -2\left[\sum_{i=0}^{n-1}\left(\frac{y_i-\beta_0-\beta_1 x_i}{\sigma_i^2}\right)\right] = 0,$$

and

$$\frac{\partial \chi^2(\boldsymbol{\beta})}{\partial \beta_0} = -2\left[\sum_{i=0}^{n-1} x_i\left(\frac{y_i-\beta_0-\beta_1 x_i}{\sigma_i^2}\right)\right] = 0.$$

The χ^2 function

For a linear fit (a first-order polynomial) we don't need to invert a matrix!! Defining

$$\gamma = \sum_{i=0}^{n-1}\frac{1}{\sigma_i^2},$$

$$\gamma_x = \sum_{i=0}^{n-1}\frac{x_i}{\sigma_i^2},$$

$$\gamma_y = \sum_{i=0}^{n-1}\left(\frac{y_i}{\sigma_i^2}\right),$$

$$\gamma_{xx} = \sum_{i=0}^{n-1}\frac{x_i x_i}{\sigma_i^2},$$

$$\gamma_{xy} = \sum_{i=0}^{n-1}\frac{y_i x_i}{\sigma_i^2},$$

we obtain

$$\beta_0 = \frac{\gamma_{xx}\gamma_y - \gamma_x\gamma_y}{\gamma\gamma_{xx}-\gamma_x^2},$$

$$\beta_1 = \frac{\gamma_{xy}\gamma - \gamma_x\gamma_y}{\gamma\gamma_{xx}-\gamma_x^2}.$$

This approach (different linear and non-linear regression) suffers often from both being underdetermined and overdetermined in the unknown coefficients β_i. A better approach is to use the Singular Value Decomposition (SVD) method discussed below. Or using Lasso and Ridge regression. See below.

Fitting an Equation of State for Dense Nuclear Matter

Before we continue, let us introduce yet another example. We are going to fit the nuclear equation of state using results from many-body calculations. The equation of state we have made available here, as function of density, has been derived using modern nucleon-nucleon potentials with the addition of three-body forces. This time the file is presented as a standard **csv** file.

The beginning of the Python code here is similar to what you have seen before, with the same initializations and declarations. We use also **pandas** again, rather extensively in order to organize our data.

The difference now is that we use **Scikit-Learn's** regression tools instead of our own matrix inversion implementation. Furthermore, we sneak in **Ridge** regression (to be discussed below) which includes a hyperparameter λ, also to be explained below.

The code

The above simple polynomial in density ρ gives an excellent fit to the data. Can you give an interpretation of the various powers of ρ?

We note also that there is a small deviation between the standard OLS and the Ridge regression at higher densities. We discuss this in more detail below.

Splitting our Data in Training and Test data

It is normal in essentially all Machine Learning studies to split the data in a training set and a test set (sometimes also an additional validation set). **Scikit-Learn** has an own function for this. There is no explicit recipe for how much data should be included as training data and say test data. An accepted rule of thumb is to use approximately 2/3 to 4/5 of the data as training data. We will postpone a discussion of this splitting to the end of these notes and our discussion of the so-called **bias-variance** tradeoff. Here we limit ourselves to repeat the above equation of state fitting example but now splitting the data into a training set and a test set.

The singular value decomposition

The examples we have looked at so far are cases where we normally can invert the matrix $\boldsymbol{X}^T\boldsymbol{X}$. Using a polynomial expansion as we did both for the masses and the fitting of the equation of state, leads to row vectors of the design matrix which are essentially orthogonal due to the polynomial character of our model. This may however not the be case in general and a standard matrix inversion algorithm based on say LU decomposition may lead to singularities. We will see an example of this below when we try to fit the coupling constant of the widely used Ising model. There is however a way to partially circumvent this problem and also gain some insight about the ordinary least squares approach.

This is given by the **Singular Value Decomposition** algorithm, perhaps the most powerful linear algebra algorithm. Let us look at a different example

where we may have problems with the standard matrix inversion algorithm. Thereafter we dive into the math of the SVD.

The Ising model

The one-dimensional Ising model with nearest neighbor interaction, no external field and a constant coupling constant J is given by

$$H = -J \sum_{k}^{L} s_k s_{k+1}, \tag{1}$$

where $s_i \in \{-1, 1\}$ and $s_{N+1} = s_1$. The number of spins in the system is determined by L. For the one-dimensional system there is no phase transition.

We will look at a system of $L = 40$ spins with a coupling constant of $J = 1$. To get enough training data we will generate 10000 states with their respective energies.

Here we use ordinary least squares regression to predict the energy for the nearest neighbor one-dimensional Ising model on a ring, i.e., the endpoints wrap around. We will use linear regression to fit a value for the coupling constant to achieve this.

Reformulating the problem to suit regression

A more general form for the one-dimensional Ising model is

$$H = -\sum_{j}^{L} \sum_{k}^{L} s_j s_k J_{jk}. \tag{2}$$

Here we allow for interactions beyond the nearest neighbors and a state dependent coupling constant. This latter expression can be formulated as a matrix-product

$$\boldsymbol{H} = \boldsymbol{X} J, \tag{3}$$

where $X_{jk} = s_j s_k$ and J is a matrix which consists of the elements $-J_{jk}$. This form of writing the energy fits perfectly with the form utilized in linear regression, that is

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \tag{4}$$

We split the data in training and test data as discussed in the previous example