Data Analysis and Machine Learning: Linear Regression

Morten Hjorth-Jensen[1,2]

[1]Department of Physics and Center for Computing in Science Education, University of Oslo, Norway
partment of Physics and Astronomy and Facility for Rare Ion Beams and National Superconducting Cyclotron Laboratory, Michigan State University, U

Jul 18, 2021

Linear Regression, basic overview

The aim of this set of lectures is to introduce basic aspects of linear regression, a widely applied set of methods used to fit continuous functions.

We will also use these widely popular methods to introduce resampling techniques like bootstrapping and cross-validation.

We will in particular focus on

- Ordinary linear regression
- Ridge regression
- Lasso regression
- Resampling techniques
- Bias-variance tradeoff

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. These methods are at the heart of all essentially all Machine Learning methods.
- and many more features

Additional Reading

For more discussions of Ridge and Lasso regression, Wessel van Wieringen's article is highly recommended. Similarly, Mehta et al's article is also recommended. The textbook by Hastie, Tibshirani, and Friedman on The Elements of Statistical Learning Data Mining is highly recommended.

Regression Analysis, Definitions and Aims

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. These variables are also called features or predictors.

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 $\boldsymbol{\beta} = [\beta_0, \ldots, \beta_{p-1}]^T$ 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 an $n \times p$ 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. This is however a classification problem and we will come back to it when we discuss Logistic Regression.

General linear models

Before we proceed let us study a case from linear algebra where we aim at fitting a set of data $\boldsymbol{y} = [y_0, y_1, \ldots, y_{n-1}]$. 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 $\boldsymbol{x} = [x_0, x_1, \ldots, x_{n-1}]$, that is $y_i = y(x_i)$ with

$i = 0, 1, 2, \ldots, n-1$. 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

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

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

Defining the vectors

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

and

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

and

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

and the design matrix

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

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

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

Note that we have used $p = n$ here. The matrix is thus quadratic (it may be 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

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

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 unknown 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

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

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 liquid drop model. Let us remind ourselves about what we did by looking at the code.

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

```
# Common imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import display
import os

# Where to save the figures and data files
PROJECT_ROOT_DIR = "Results"
FIGURE_ID = "Results/FigureFiles"
DATA_ID = "DataFiles/"

if not os.path.exists(PROJECT_ROOT_DIR):
    os.mkdir(PROJECT_ROOT_DIR)

if not os.path.exists(FIGURE_ID):
    os.makedirs(FIGURE_ID)

if not os.path.exists(DATA_ID):
    os.makedirs(DATA_ID)

def image_path(fig_id):
    return os.path.join(FIGURE_ID, fig_id)

def data_path(dat_id):
    return os.path.join(DATA_ID, dat_id)

def save_fig(fig_id):
    plt.savefig(image_path(fig_id) + ".png", format='png')
```

```
infile = open(data_path("MassEval2016.dat"),'r')


# Read the experimental data with Pandas
Masses = pd.read_fwf(infile, usecols=(2,3,4,6,11),
              names=('N', 'Z', 'A', 'Element', 'Ebinding'),
              widths=(1,3,5,5,5,1,3,4,1,13,11,11,9,1,2,11,9,1,3,1,12,11,1),
              header=39,
              index_col=False)

# Extrapolated values are indicated by '#' in place of the decimal place, so
# the Ebinding column won't be numeric. Coerce to float and drop these entries.
Masses['Ebinding'] = pd.to_numeric(Masses['Ebinding'], errors='coerce')
Masses = Masses.dropna()
# Convert from keV to MeV.
Masses['Ebinding'] /= 1000

# Group the DataFrame by nucleon number, A.
Masses = Masses.groupby('A')
# Find the rows of the grouped DataFrame with the maximum binding energy.
Masses = Masses.apply(lambda t: t[t.Ebinding==t.Ebinding.max()])
A = Masses['A']
Z = Masses['Z']
N = Masses['N']
Element = Masses['Element']
Energies = Masses['Ebinding']

# Now we set up the design matrix X
X = np.zeros((len(A),5))
X[:,0] = 1
X[:,1] = A
X[:,2] = A**(2.0/3.0)
X[:,3] = A**(-1.0/3.0)
X[:,4] = A**(-1.0)
# Then nice printout using pandas
DesignMatrix = pd.DataFrame(X)
DesignMatrix.index = A
DesignMatrix.columns = ['1', 'A', 'A^(2/3)', 'A^(-1/3)', '1/A']
display(DesignMatrix)
```

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

parameterized values $\tilde{y}_i$, namely

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

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

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

This function is one possible way to define the so-called cost function.

It is also common to define the function C as

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

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

$$C(\boldsymbol{\beta}) = \frac{1}{n}\left\{\left(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}\right)^T\left(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}\right)\right\},$$

can be linked to the variance of the quantity y_i if we interpret the latter as the mean value. Below we will show that

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

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 $C(\boldsymbol{\beta})$, that is we are going to solve the problem

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^p}\frac{1}{n}\left\{\left(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}\right)^T\left(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}\right)\right\}.$$

In practical terms it means we will require

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

which results in

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

or in a matrix-vector form as

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

Interpretations and optimizing our parameters

We can rewrite

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

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 $\boldsymbol{X}^T\boldsymbol{X}$.

Small question: Do you think the example we have at hand here (the nuclear binding energies) can lead to problems in inverting the matrix $\boldsymbol{X}^T\boldsymbol{X}$? What kind of problems can we expect?

Some useful matrix and vector expressions

The following matrix and vector relation will be useful here and for the rest of the course. Vectors are always written as boldfaced lower case letters and matrices as upper case boldfaced letters.

$$\frac{\partial(\boldsymbol{b}^T\boldsymbol{a})}{\partial \boldsymbol{a}} = \boldsymbol{b},$$

$$\frac{\partial(\boldsymbol{a}^T\boldsymbol{A}\boldsymbol{a})}{\partial \boldsymbol{a}} = (\boldsymbol{A} + \boldsymbol{A}^T)\boldsymbol{a},$$

$$\frac{\partial tr(\boldsymbol{B}\boldsymbol{A})}{\partial \boldsymbol{A}} = \boldsymbol{B}^T,$$

$$\frac{\partial \log|\boldsymbol{A}|}{\partial \boldsymbol{A}} = (\boldsymbol{A}^{-1})^T.$$

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

```
# matrix inversion to find beta
beta = np.linalg.inv(X.T @ X) @ X.T @ Energies
# or in a more old-fashioned way
# beta = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(Energies)
# and then make the prediction
ytilde = X @ beta
```

Alternatively, you can use the least squares functionality in **Numpy** as

```
fit = np.linalg.lstsq(X, Energies, rcond =None)[0]
ytildenp = np.dot(fit,X.T)
```

And finally we plot our fit with and compare with data

```
Masses['Eapprox']  = ytilde
# Generate a plot comparing the experimental with the fitted values values.
fig, ax = plt.subplots()
ax.set_xlabel(r'$A = N + Z$')
ax.set_ylabel(r'$E_\mathrm{bind}\,/\mathrm{MeV}$')
ax.plot(Masses['A'], Masses['Ebinding'], alpha=0.7, lw=2,
            label='Ame2016')
ax.plot(Masses['A'], Masses['Eapprox'], alpha=0.7, lw=2, c='m',
            label='Fit')
ax.legend()
save_fig("Masses2016OLS")
plt.show()
```

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 $_Scikit_Learn_intheintroductoryslides.Sincewearenotusing_Scikit-Learnherewecandefineourown\mathrm{R2}functionas$

```
\PYG{k}{def} \PYG{n+nf}{R2}\PYG{p}{(}\PYG{n}{y\PYGZus{}data}\PYG{p}{,} \PYG{n}{y\PYGZus{}mod
    \PYG{k}{return} \PYG{l+m+mi}{1} \PYG{o}{\PYGZhy{}} \PYG{n}{np}\PYG{o}{.}\PYG{n}{sum}\PYG
```

```
   \relax
   \providecommand\hyper@newdestlabel[2]{}
   \providecommand\zref@newlabel[2]{}
   \providecommand\HyperFirstAtBeginDocument{\AtBeginDocument}
   \HyperFirstAtBeginDocument{\ifx\hyper@anchor\@undefined
   \global\let\oldcontentsline\contentsline
   \gdef\contentsline#1#2#3#4{\oldcontentsline{#1}{#2}{#3}}
   \global\let\oldnewlabel\newlabel
   \gdef\newlabel#1#2{\newlabelxx{#1}#2}
   \gdef\newlabelxx#1#2#3#4#5#6{\oldnewlabel{#1}{{#2}{#3}}}
   \AtEndDocument{\ifx\hyper@anchor\@undefined
   \let\contentsline\oldcontentsline
   \let\newlabel\oldnewlabel
   \fi}
   \fi}
   \global\let\hyper@last\relax
   \gdef\HyperFirstAtBeginDocument#1{#1}
   \providecommand\HyEField@AuxAddToFields[1]{}
   \providecommand\HyEField@AuxAddToCoFields[2]{}
   \@writefile{toc}{\contentsline {paragraph}{}{2}{section*.6}\protected@file@percent
}
   \@writefile{toc}{\contentsline {paragraph}{}{3}{section*.8}\protected@file@percent
}
   \@writefile{toc}{\contentsline {paragraph}{}{3}{section*.10}\protected@file@percent
}
   \@writefile{toc}{\contentsline {paragraph}{}{3}{section*.12}\protected@file@percent
}
   \@writefile{toc}{\contentsline {paragraph}{}{4}{section*.14}\protected@file@percent
}
   \@writefile{toc}{\contentsline {paragraph}{}{4}{section*.16}\protected@file@percent
}
   \@writefile{toc}{\contentsline {paragraph}{}{5}{section*.18}\protected@file@percent
}
   \@writefile{toc}{\contentsline {paragraph}{}{5}{section*.20}\protected@file@percent
}
   \@writefile{toc}{\contentsline {paragraph}{}{6}{section*.22}\protected@file@percent
}
   \@writefile{toc}{\contentsline {paragraph}{}{7}{section*.25}\protected@file@percent
}
   \@writefile{toc}{\contentsline {paragraph}{}{8}{section*.27}\protected@file@percent
}
   \@writefile{toc}{\contentsline {paragraph}{}{9}{section*.29}\protected@file@percent
}
   \@writefile{toc}{\contentsline {paragraph}{}{9}{section*.30}\protected@file@percent
}
   \gdef\minted@oldcachelist{,
```