Multivariate-Outcome Linear Regression

Last modified: 2026-11-19

```
# The Python chunks import numpy and pandas; reticulate provisions them.
library(reticulate) # nolint: undesirable_function_linter
py_packages <- c("numpy", "pandas")
py_require(py_packages)
```

Source

This material was a section of the linear regression page. It extends the normal equations from one response to several.

1 The multivariate model

Definition 1.1 (Multivariate linear regression)**.** When there are multiple target variables ($q > 1$), each response vector is $\underset{\sim}{y}_i \in \mathbb{R}^q$. The linear model assumes:

$$\underset{\sim}{y} = \mathbf{B}^\top \underset{\sim}{D}(\underset{\sim}{x}) + \underset{\sim}{\varepsilon}, \qquad \underset{\sim}{D}(\underset{\sim}{x}) \in \mathbb{R}^{p^*},\ \mathbf{B} \in \mathbb{R}^{p^* \times q}$$

where noise vectors $\underset{\sim}{\varepsilon} \sim \mathcal{N}(\vec{0}, \Sigma)$ have zero expected value[a] and covariance matrix[b] $\Sigma \in \mathbb{R}^{q \times q}$.
In matrix notation over all n instances:

$$\mathbf{Y} = \mathbf{DB} + \mathbf{E}$$

where $\mathbf{Y} \in \mathbb{R}^{n \times q}$ stacks target rows, $\mathbf{D} \in \mathbb{R}^{n \times p^*}$ is the design matrix, and $\mathbf{E} \in \mathbb{R}^{n \times q}$ stacks noise rows. The objective minimizes the squared Frobenius norm[c] of the residual matrix:

$$\hat{\mathbf{B}} = \arg\min_{\mathbf{B}} \|\mathbf{Y} - \mathbf{DB}\|_F^2$$

Write $\mathbf{Y}_{*,k}$ and $\mathbf{B}_{*,k}$ for the k-th columns of $\mathbf{Y}$ and $\mathbf{B}$, respectively.

[a] https://morrison-lab.github.io/pds/expectation.html#def-expectation
[b] https://morrison-lab.github.io/pds/variance-covariance.html#def-cov-vec-x
[c] https://morrison-lab.github.io/mds/linear-algebra-special-matrices.html#def-frobenius-norm

Example 1.1 (Two targets at once)**.**
In the `Auto` data [1], let the targets be `mpg` and `weight`, so $q = 2$, and let the features be an intercept and `horsepower`, so $p^* = 2$. Following Definition 1.1, build $\mathbf{Y}$, $\mathbf{D}$, the fitted $\hat{\mathbf{B}}$ and the residual matrix[a] $\mathbf{E}$, and estimate the noise covariance Σ from the residuals:

1.0.1 Python

```python
import numpy as np
import pandas as pd

auto_mr = pd.read_csv("data/auto.csv")
Y_mr = auto_mr[["mpg", "weight"]].to_numpy(float)
Phi_mr = np.column_stack([np.ones(len(Y_mr)), auto_mr["horsepower"]])

# One column of B per target; lstsq fits all columns at once.
B_mr = np.linalg.lstsq(Phi_mr, Y_mr, rcond=None)[0]
E_mr = Y_mr - Phi_mr @ B_mr
# The q x q covariance of the noise: average outer product of the residual rows.
Sigma_mr = E_mr.T @ E_mr / len(Y_mr)
print(Y_mr.shape, Phi_mr.shape, B_mr.shape, E_mr.shape)
```

```
(392, 2) (392, 2) (2, 2) (392, 2)
```

```python
print(np.array2string(Sigma_mr, precision=1, suppress_small=True))
```

```
[[    23.9  -1053.2]
 [ -1053.2 181763.8]]
```

```python
# Bootstrap: resample cars with replacement, refit both targets,
# and recompute the off-diagonal covariance entry each time.
rng_mr = np.random.default_rng(571)
boot_cov_mr = np.empty(2000)
for b in range(2000):
    rows = rng_mr.integers(0, len(Y_mr), len(Y_mr))
    coef_b = np.linalg.lstsq(Phi_mr[rows], Y_mr[rows], rcond=None)[0]
    resid_b = Y_mr[rows] - Phi_mr[rows] @ coef_b
    boot_cov_mr[b] = (resid_b.T @ resid_b / len(rows))[0, 1]
# The 2.5th and 97.5th percentiles of the resampled entries.
cov_ci_mr = np.quantile(boot_cov_mr, [0.025, 0.975])
print(cov_ci_mr)
```

```
[-1277.54117641  -822.01445198]
```

1.0.2 R

```r
auto_mr <- read.csv("data/auto.csv")
Y_mr <- as.matrix(auto_mr[, c("mpg", "weight")])
Phi_mr <- cbind(1, auto_mr$horsepower)

# One column of B per target; lm with a matrix response fits all columns at once.
B_mr <- coef(lm(Y_mr ~ auto_mr$horsepower))
E_mr <- Y_mr - Phi_mr %*% B_mr
# The q x q covariance of the noise: average outer product of the residual rows.
Sigma_mr <- crossprod(E_mr) / nrow(Y_mr)
rbind(Y = dim(Y_mr), Phi = dim(Phi_mr), B = dim(B_mr), E = dim(E_mr))
```

```
    [,1] [,2]
Y    392    2
Phi  392    2
B      2    2
E    392    2
```

```r
round(Sigma_mr, 1)
```

```
           mpg   weight
mpg       23.9  -1053.2
weight -1053.2 181763.8
```

```r
# Bootstrap: resample cars with replacement, refit both targets,
# and recompute the off-diagonal covariance entry each time.
set.seed(571)
boot_cov_mr <- replicate(2000, {
  rows <- sample(nrow(Y_mr), replace = TRUE)
  coef_b <- coef(lm(Y_mr[rows, ] ~ auto_mr$horsepower[rows]))
  resid_b <- Y_mr[rows, ] - Phi_mr[rows, ] %*% coef_b
  crossprod(resid_b)[1, 2] / length(rows)
})
# The 2.5th and 97.5th percentiles of the resampled entries.
cov_ci_mr <- quantile(boot_cov_mr, c(0.025, 0.975))
cov_ci_mr
```

```
     2.5%     97.5%
-1270.0709  -818.2046
```

$\mathbf{Y}$ and $\mathbf{E}$ have one row per car and one column per target, $\mathbf{D}$ has one column per feature, and $\hat{\mathbf{B}}$ has one column per target, so its shape is $p^* \times q$. The diagonal of the covariance matrix holds the residual variance of each target. The off-diagonal entry, -1053.2, is negative: cars whose `mpg` is above the line tend to be lighter than the line predicts, so the errors of the two targets are correlated. Its 95% bootstrap percentile interval, from 2000 resamples of the cars, runs from -1270.1 to -818.2 (in miles per gallon times pounds), so the interval excludes zero. The interval describes which cars were drawn, not the process that produced them beyond that. This is the structure the matrix Σ in the definition allows for.

[a] https://morrison-lab.github.io/mds/linear-algebra-matrices.html#def-matrix

2 The objective by columns

Theorem 2.1 (The Frobenius objective is a sum over columns)**.** *For every $\mathbf{B} \in \mathbb{R}^{p^* \times q}$, the objective of Definition 1.1 is the sum[a] of the squared-error objectives of the q single-target regressions:*

$$\|\mathbf{Y} - \mathbf{DB}\|_F^2 = \sum_{k=1}^{q} \|\mathbf{Y}_{*,k} - \mathbf{DB}_{*,k}\|^2.$$

[a] https://morrison-lab.github.io/mds/algebra-sums.html#def-summation

Proof

Proof. The squared Frobenius norm of a matrix is the sum of the squares of its entries. Sum them column by column. Column k of $\mathbf{Y} - \mathbf{DB}$ is $\mathbf{Y}_{*,k} - \mathbf{DB}_{*,k}$, and the sum of squares of its entries is $\|\mathbf{Y}_{*,k} - \mathbf{DB}_{*,k}\|^2$. Adding over $k = 1, \ldots, q$ gives the right side. □

Example 2.1 (Two targets from horsepower)**.**
In the `Auto` data [1], predict the two targets `mpg` and `acceleration` from an intercept and `horsepower`. For the least squares $\hat{\mathbf{B}}$ and for a second matrix[a] $\mathbf{B}$ that shifts every entry of $\hat{\mathbf{B}}$ by the same amount, compute the squared Frobenius norm of the residual matrix and the sum of the squared norms of its columns:

2.0.1 Python

```python
import numpy as np
import pandas as pd

auto_mv = pd.read_csv("data/auto.csv")
Phi_mv = np.column_stack([np.ones(len(auto_mv)), auto_mv["horsepower"].to_numpy(float)])
Y_mv = auto_mv[["mpg", "acceleration"]].to_numpy(float)
B_hat_mv = np.linalg.lstsq(Phi_mv, Y_mv, rcond=None)[0]
B_other_mv = B_hat_mv + 0.5

def both_sides(B):
    resid = Y_mv - Phi_mv @ B
    return np.linalg.norm(resid, "fro") ** 2, sum(np.sum(resid[:, k] ** 2) for k in range(resid.shape

sides_hat = both_sides(B_hat_mv)
sides_other = both_sides(B_other_mv)
print(sides_hat, sides_other)
```

```
(np.float64(10948.360306521437), np.float64(10948.360306521441)) (np.float64(2480858.3603065214), np.f
```

2.0.2 R

```r
auto_mv <- read.csv("data/auto.csv")
Phi_mv <- cbind(1, auto_mv$horsepower)
Y_mv <- as.matrix(auto_mv[, c("mpg", "acceleration")])
B_hat_mv <- solve(crossprod(Phi_mv), crossprod(Phi_mv, Y_mv))
B_other_mv <- B_hat_mv + 0.5

both_sides <- function(B) {
  resid <- Y_mv - Phi_mv %*% B
  c(frobenius = norm(resid, "F")^2, columns = sum(apply(resid, 2, function(col) sum(col^2))))
}
rbind(fitted = both_sides(B_hat_mv), other = both_sides(B_other_mv))
```

```
        frobenius    columns
fitted   10948.36   10948.36
other  2480858.36 2480858.36
```

For the fitted matrix, both sides equal 10,948.4. For the shifted matrix, they equal 2,480,858 and 2,480,858. The identity[b] holds for every $\mathbf{B}$, not only the minimizer[c]. The fitted matrix has the smaller residual, as the minimizer must.

[a] https://morrison-lab.github.io/mds/linear-algebra-matrices.html#def-matrix
[b] https://morrison-lab.github.io/mds/algebra-basics.html#def-identity
[c] https://morrison-lab.github.io/mds/algebra-basics.html#def-global-minimizer

3 The normal equations

Theorem 3.1 (Multivariate normal equations)**.** *The least-squares estimator for multivariate linear regression is given by:*

$$\hat{\mathbf{B}} = (\mathbf{D}^\top \mathbf{D})^{-1} \mathbf{D}^\top \mathbf{Y}$$

Proof

Proof. By Theorem 2.1,

$$\|\mathbf{Y} - \mathbf{DB}\|_F^2 = \sum_{k=1}^{q} \|\mathbf{Y}_{*,k} - \mathbf{DB}_{*,k}\|^2,$$

so each column $\mathbf{B}_{*,k}$ of $\mathbf{B}$ can be optimized independently. By the normal equations, the minimizer for each column is

$$\hat{\mathbf{B}}_{*,k} = (\mathbf{D}^\top \mathbf{D})^{-1} \mathbf{D}^\top \mathbf{Y}_{*,k}.$$

Concatenating all q columns yields

$$\hat{\mathbf{B}} = (\mathbf{D}^\top \mathbf{D})^{-1} \mathbf{D}^\top \mathbf{Y}.$$

□

Example 3.1 (One formula, three separate fits)**.**
In the `Auto` data [1], predict `mpg`, `acceleration` and `weight` from an intercept, `horsepower` and `displacement`. Compute $\hat{\mathbf{B}} = (\mathbf{D}^\top \mathbf{D})^{-1} \mathbf{D}^\top \mathbf{Y}$ of Theorem 3.1 in one step, and compare each of its columns with a separate least-squares fit of that target:

3.0.1 Python

```
import numpy as np
import pandas as pd

auto_mn = pd.read_csv("data/auto.csv")
Y_mn = auto_mn[["mpg", "acceleration", "weight"]].to_numpy(float)
Phi_mn = np.column_stack([np.ones(len(Y_mn)), auto_mn["horsepower"], auto_mn["displacement"]])

# All three targets at once.
B_mn = np.linalg.solve(Phi_mn.T @ Phi_mn, Phi_mn.T @ Y_mn)

# One univariate fit per target column.
B_cols_mn = np.column_stack([
    np.linalg.lstsq(Phi_mn, Y_mn[:, k], rcond=None)[0] for k in range(Y_mn.shape[1])
])
print(np.array2string(B_mn, precision=4, suppress_small=True))
```

```
[[  37.4695   21.3115 1380.2336]
 [  -0.0583   -0.074     3.1021]
 [  -0.0408    0.0101    6.5494]]
```

```
print(np.allclose(B_mn, B_cols_mn))
```

```
True
```

3.0.2 R

```r
auto_mn <- read.csv("data/auto.csv")
Y_mn <- as.matrix(auto_mn[, c("mpg", "acceleration", "weight")])
Phi_mn <- cbind(1, auto_mn$horsepower, auto_mn$displacement)

# All three targets at once.
B_mn <- solve(crossprod(Phi_mn), crossprod(Phi_mn, Y_mn))

# One univariate fit per target column.
B_cols_mn <- sapply(colnames(Y_mn), function(k) coef(lm(Y_mn[, k] ~ Phi_mn[, 2] + Phi_mn[, 3])))
round(B_mn, 4)
```

```
         mpg acceleration    weight
[1,] 37.4695      21.3115 1380.2336
[2,] -0.0583      -0.0740    3.1021
[3,] -0.0408       0.0101    6.5494
```

```r
isTRUE(all.equal(unname(B_mn), unname(B_cols_mn)))
```

```
[1] TRUE
```

$\hat{\mathbf{B}}$ has one column per target, and each column matches the fit that ignores the other targets. The multivariate problem decouples this way because the squared Frobenius norm is a sum over the columns of the residual matrix.

4 Videos

Watch

Hutchinson teaches the multivariate case[a] (11 min) [2].

[a] https://facultyweb.cs.wwu.edu/~hutchib2/video_lectures/data371/#linear_regression_pt2

[1] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning with applications in R*, 2nd ed. Springer, 2021. Available: https://www.statlearning.com/

[2] B. Hutchinson, "DATA 471/571 (machine learning) and CSCI 481/581 (deep learning) video lectures." Western Washington University. Accessed: Sep. 28, 2026. [Online]. Available: https://facultyweb.cs.wwu.edu/~hutchib2/video_lectures/data371/