1 Генерация тестовых данных

```
import numpy as np
import random
from scipy import stats
from sklearn import linear_model


n = 5 # number of features
N = 100 # number of observations

np.set_printoptions(precision=4)
```

Будем искать регрессию по $N = 100$ наблюдениям $n = 5$ признаков.
Создадим ковариационную матрицу `covariance`, определяющую взаимозависимость признаков.

```
covariance = np.zeros((n,n))
for i in xrange(n):
  for j in xrange(n):
    covariance[i,j] = covariance[j, i] if covariance[j, i] != 0 else 2*random.random()-1
  covariance = np.dot(covariance, np.matrix.transpose(covariance))
```

```
covariance:
[[   9.1414  -34.4653  -12.9226   -3.3109    1.0664]
 [ -34.4653  137.978    54.8444   16.43     -0.7962]
 [ -12.9226   54.8444   22.9445    7.7641    0.9377]
 [  -3.3109   16.43      7.7641    4.0312    1.3248]
 [   1.0664   -0.7962    0.9377    1.3248    2.0319]]
```

Определим операцию `random_vector(R, N=1)`, которая возвращает вектор длины N (по умолчанию равной 1), состоящий из реализаций нормально распределённого случайного вектора с ковариационной матрицей `R`.

```
def random_vector(R, N=1):
  k = len(R[0]) # get number of features from covariance matrix because relying on global variables
  x = np.empty((N, k)) # create empty array of given shape
  # fill this array with random observations of normal distribution with given covariance matrix R
  for i in xrange(N):
    s = np.random.normal(size=len(R[0]))
    x[i] = np.dot(np.linalg.cholesky(R), s)
```

Сгенерируем выборку `X` и получим значения `Y` как линейную функцию от `X` со случайным шумом.

```
# get random sample
X = random_vector(covariance, N)

beta = np.array([3,2,0,1,1]) # regression coefficients
sigma = 0.01 # noise variance

# get Y that we will try to predict by X
Y = np.add(np.dot(X, beta), np.random.normal(scale=sigma, size=N))
```

Таким образом, мы получили набор $\left\{\{x_{ji}\}_{i=1}^{p}\right\}_{j=1}^{n}$ и набор $\{y_i\}_{i=1}^{n}$, причём

$$y_i = 3 \cdot x_{1i} + 2 \cdot x_{2i} + 0 \cdot x_{3i} + 1 \cdot x_{4i} + 1 \cdot x_{5i} + \varepsilon_i, \varepsilon_i \sim N(0,1)$$

.

2 Стандартная линейная регрессия

Объект `clf` после вызова функции `fit()` будет иметь всю информацию по поводу предсказания `Y` по `X`:

```
clf = linear_model.LinearRegression()
clf.fit(X, Y)
```

`LinearRegression()` подразумевает модель регрессии вида $y = \beta_0 + \beta_1 \cdot x_1 + \ldots + \beta_p \cdot x_p$, рассматривая которую, мы должны получить коэффициенты регрессии, близкие к $\beta_0 = 0, \beta_1 = 3, \beta_2 = 2, \beta_3 = 0, \beta_4 = 1, \beta_5 = 1$.

```
estimated regression:
-0.000897018574098 [ 3.0115  2.0065 -0.0094  1.0004  1.0009]
real regression:
[3 2 0 1 1]
```

3 Подсчёт коэффициентов линейной регрессии с помощью матрицы вторых моментов

Оценим коэффициенты линейной регрессии в модели $y_i = \alpha + \beta_1(x_{1i} - \bar{x_1}) + \ldots + \beta_p(x_{pi} - \bar{x_p})$ (`L` – матрица вторых центральных моментов внутри X, `L0` – вектор вторых центральных моментов между X и Y, `mean_x` $= \bar{X}$). Научимся считать алгебраические дополнения:

```
def matrix_cofactor(matrix, row, col):
    nrows, ncols = matrix.shape
    minor = np.zeros([nrows-1, ncols-1])
    minor[:row,:col] = matrix[:row,:col]
    minor[row:,:col] = matrix[row+1:,:col]
    minor[:row,col:] = matrix[:row,col+1:]
    minor[row:,col:] = matrix[row+1:,col+1:]
    C = (-1)**(row+col) * np.linalg.det(minor)
    return C
```

Построим матрицу вторых моментов и оценки параметров по ней:

```
L = np.empty((n, n))
mean_x = np.mean(X, axis=0)
for i in xrange(n):
  for j in xrange(n):
    # map(f, X, ...) <=> sapply(X, f, ...)
    L[i][j] = np.average(
      map(lambda t:
        (X[t][i] - mean_x[i])*(X[t][j] - mean_x[j]), xrange(N)))
L0 = np.empty((n, ))
mean_y = np.mean(Y)
L0 = map(lambda j: np.average(
  map(lambda t:
    (Y[t] - mean_y)*(X[t][j] - mean_x[j]), xrange(N))), xrange(n))

LSM_expected_alpha = mean_y
```

```
LSM_expected_beta = map(lambda i:
  np.sum(map(lambda j:
    L0[j]*matrix_cofactor(L,i,j), xrange(n)
  ))/np.linalg.det(L), xrange(n))
```

Оценки вектора $(\beta_1, \ldots, \beta_5)$ должны оказаться похожими на изначально заданное `beta`:

```
regression estimates, calculated using cofactors:
[ 3.0115  2.0065 -0.0094  1.0004  1.0009]
real regression:
[3 2 0 1 1]
```

Параметр β_0, равный 0 в нашем случае, получается как $\beta_0 = \alpha - \beta_1 \bar{x_1} + \ldots + \beta_p \bar{x_p}$

```
beta_0 = LSM_expected_alpha - np.sum(map(lambda x,y: x*y, beta, mean_x))
```

```
-0.000656852795076
```