1 Генерация тестовых данных

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:
[[ 329.3752  -24.1551 -124.544    28.5415  -11.7998]
 [ -24.1551   42.7581    6.5132   -8.643     1.1639]
 [-124.544     6.5132   48.9312   -9.8425    4.563 ]
 [  28.5415   -8.643    -9.8425    3.7425   -0.7976]
 [ -11.7998    1.1639    4.563    -0.7976    2.0182]]
```

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 is
  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))
```

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)
```

```
estimated regression:
[  2.9994e+00   2.0008e+00  -7.6634e-04   1.0047e+00   9.9961e-01]
real regression:
[3 2 0 1 1]
```

Посчитаем коэффициенты линейной регрессии с помощью матрицы вторых моментов (`L` – матрица вторых центральных моментов внутри X, `L0` – вектор вторых центральных моментов между X и Y, `mean_x` $= \bar{X}$):

```
L = np.empty((n, n))
mean_x = np.mean(X, axis=1)
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))
```

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
```

И посчитаем коэффициенты линейной регрессии:

```
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))
```

```
regression estimates, calculated using cofactors:
[ 3.6057  1.5062  0.8864 -2.3301  1.5837]
2.09049325225
```