```python
# Assessed exercises 4
# As before, each question has an associated function, with input argu
# matching those specified in the question. Your functions will be tes
# range of different input values, against a model solution, to see if
# produce the same answers.
import numpy as np
import numpy.random as npr
import numpy.linalg as npl

# At the end of lecture 4 we simulated some 2 dimensional data from a
# regression model. In this assignment we're going to try generalise t
# to higher dimensions.

# The first thing we'll need to to do is simulate the variables xi fro
# uniform distribution

# Q1 Write a function that takes n, a1, a2 and s as inputs, and return
# of length n, drawn from a uniform distribution U(a1,a2). The seed sh
# set to s.
def exercise1(n,a1,a2,s):
    npr.seed(s)
    x_unif = npr.uniform(a1,a2,size = n)
    return x_unif

# A multiple linear regression model is defined as
# y =  b0 + b1*x1 + b2*x2 + b3*x3 + b4*x4 + ... + bpxp + epsilon
# where p is the dimension and {x1,x2,...,xp} are the variables

# To fit a linear regression model to a dataset we use the standard eq
# b = (X^T X)^{-1} X^T y, to estimate the coefficients b = [b0, b1, ..
# Here, y is the dataset (1D array) and X is a matrix where the first
# filled entirely with 1s and the subsequent columns are x1, x2, etc.

# Q2 Write a function that takes p and a list S as inputs, and returns
# matrix X. Use your function from exercise one to create the x1, x2,
# variables, with n = 1000, a1 = 0 and a2 =10. The input S = (s0, s1,
# where si corresponds to the seed that should be used to create the v
# Hint: Python treats all 1D arrays as row vectors. Instead Create the
# of X and return its tranpose ((X^T)^T=X). Also, the function vstack
# in useful here.

def exercise2(p,S):
    n = 1000
    a1 = 0
    a2 = 10
    x_list = ([1]*1000)
    for i in range(len(S)):
        temp = exercise1(1000,0,10,S[i])
        x_list = np.vstack((x_list,temp))
    return x_list.T

# Q3 Write a function that takes the matirx X and vector y as input, a
# performs a multiple linear regression, using the standard equation
```

```
# b = (X^T X)^{-1} X^T y, by calculating the inverse of (X^T X) and mu
# the result by (X^T y). The function should return the vector b, whic
# the fits for the intercept and slope parameters (b0, b1, b2, b3, b4)


def exercise3(X,y):
    b = np.dot(npl.inv(np.dot(X.T,X)),np.dot(X.T,y))
    return b

# Q4 Write another function, with the same inputs and outputs, which u
# solve function rather than finding the inverse and then multiplying.
def exercise4(X,y):
    temp_a = np.dot(X.T,X)
    temp_y = np.dot(X.T,y)
    b = npl.solve(temp_a,temp_y)
    return b


# Try testing you functions for different models, e.g.
# y = 3 + 2*x1 - x2 + 0.5*x3 - 0.1*x4 + npr.normal(0,1,n)
# where x1, x2, x3, x4 should be comupted using the function exercise
# different seeds s1, s2, s3, s4. These seeds should be given as a lis
# into exercise2 to create the matrix X. Running exercise3 and exercis
# give the same result, a vector (1D array) of length p+1, with entrie
# equal to the coefficients defined in your multiple linear model,
# e.g. [3,2,-1,0.5,-0.1] for the above example. You can use %timeit to
# whether exercise3 or exercise4 is quicker for fitting the regressio
'''
This part is a test of the functions exercise3 and exercise4.
'''
s_test = (19,99,80)
x_test = exercise2(3,s_test)
test_y = npr.randint(10,100,size = 1000)

# using exercise3 to generate matrix B
test_b1 = exercise3(x_test,test_y)
%timeit exercise3(x_test,test_y)

#using exercise4 to get matrix B
test_b2 = exercise4(x_test,test_y)
%timeit exercise4(x_test,test_y)

# the result shows 2 functions get the same matrix B and
#exercise4 needs less time than exercise3 to get matrix B
```