Python II - Practice exam 1 (solutions)

35V3A1-Computational Aspects in Econometrics

Introduction

The description on TestVision will read something as follows:

On the next page you will find the four questions of the Python II module that you will have to implement correctly for a total maximum of 7 + 7 + 19 + 17 = 50 points.

Use the following (MANDATORY) template for your answers, and upload it on the next page: python-ii-template.

Apart from correctly solving the problems, your submission is also assessed on the other usual "Good coding" criteria, such as efficiency, hard coding, DRY, single responsibility, coding style & documentation, KISS.

Packages seen in the course materials are included in the template.

```
# Import any packages needed
import numpy as np
import scipy.optimize as optimize
import scipy.stats as stats
from scipy.optimize import linprog
import matplotlib.pyplot as plt
```

Question 1 [7 pts]

Consider the linear minimization problem

$$
\begin{array}{llrcrcl}
\text{min} & z = & 30x_1 & + & 20x_2 & & \\
\text{s.t.} & & 2x_1 & + & 2x_2 & \leq & 8 \\
 & & x_1 & - & 2x_2 & \leq & 6 \\
 & & x_1 & & & \geq & 0 \\
 & & & & x_2 & \in & \mathbb{R} = (-\infty, \infty)
\end{array}
$$

Implement this problem using the `linprog` function, and print the optimal solution (x_1, x_2) found by `linprog` (which should be $[0, -3]$).

```
## Define input data

# Coefficients of the objective function
c = np.array([30, 20])  # Minimize 30x1 + 20x2

# Coefficients of the inequality constraints (Ux \leq z)
U = np.array([[2,2],[1,-2]]) # 2x1 + 2x2 <= 8, x1 - 2x2 <= 6
z = np.array([[8],[6]])

# Bounds for the variables x1 and x2 (l = 0, u = infinity)
x1_bounds = (0, None)  # x1 >= 0
x2_bounds = (None, None)  # x2 unconstrained

# Solve the linear programming problem
```

```
# (You can use \ to continue command on next line)
result = linprog(c, A_ub=U, b_ub=z, \
                    bounds=[x1_bounds, x2_bounds])

# Output the results
print('The problem is solved by', result.x, \
      'with objective function value', result.fun)
```

```
The problem is solved by [ 0. -3.] with objective function value -60.0
```

Question 2 [7 pts]

Write a function quantities() that takes as input a one-dimensional array $x \in \mathbb{R}^n$, and a two-dimensional $n \times n$ array $A \in \mathbb{R}^{n\times n}$ where the entry at position (i, j) is denoted by a_{ij} for $i, j = 0, \dots, n-1$. It should output the following two quantities:

- The elements of the matrix A ordered from smallest to largets in every row.
- The quantity $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_{ij}(x_i x_j)^2$.

Do not use for-loops, if- or while-statements.

For $x = [-1, 1, 2]$ *and* $A = [[-5, -10, 9], [1, 3, 0], [2, 1, 4]]$*, the outputs should be* $[[-10, -5, 9], [0, 1, 3], [1, 2, 4]]$ *and* 101*, respectively.*

```
def quantities(x,A):
    first = np.sort(A,axis=1)
    second = np.sum(A*(x*x[:,None])**2)
    return first, second

# Test input
x = np.array([-1,1,2])
A = np.array([[-5,-10,9],[1,3,0],[2,1,4]])

first, second = quantities(x,A)
print(first)
print(second)
```

```
[[-10  -5   9]
 [  0   1   3]
 [  1   2   4]]
101
```

Question 3 [5 + 3 + 8 + 3 = 19 pts]

For n identically and independently distributed (i.i.d.) random variables $X_0, \dots, X_{n-1}$ with common cumulative density function (cdf) $F : \mathbb{R} \to [0, 1]$ and probability density function (pdf) $f : \mathbb{R} \to \mathbb{R}_{\geq 0}$, the probability density function (pdf) of the random variable $X_{\text{max}} = \max\{X_0, \dots, X_{n-1}\}$ is given by

$$f_{\text{max}}(x) = n \cdot f(x) \cdot F(x)^{n-1}$$

a) **[5 pts]** Write a function `pdf_max` that takes as input a continuous probability distribution (a `stats.rv_continuous` object), integer $n \in \mathbb{N}$ and number $x \in \mathbb{R}$. It should output the number $f_{\text{max}}(x)$.

```
def pdf_max(x,dist,n):
    return n*dist.pdf(x)*dist.cdf(x)**(n-1)
```

b) **[3 pts]** Test your function on a normal distribution with mean 4 and standard deviation 2, $n = 5$ and $x = 6$. *The answer should be* ≈ 0.30.

```
dist = stats.norm(loc=4,scale=2)
n = 5
x = 6

print(pdf_max(x,dist,n))
```

```
0.3031089650710318
```

c) **[8 pts]** The expected value of X_{max} for a distribution supported on an interval $[a, b]$ is given by

$$\int_a^b x f_{\text{max}}(x) \mathrm{d}x$$

For a given vector $x = [x_0, \dots, x_k]$ with $x_0 = a$ and $x_k = b$, we can approximate this integral by the sum

$$\sum_{i=0}^{k-1} (x_{i+1} - x_i) \cdot x_i f_{\text{max}}(x_i).$$

Write a function `expectation_max` that takes as input a vector $x = [x_0, \dots, x_k]$, a continuous probability distribution (a `stats.rv_continuous` object) supported on $[x_0, x_k]$, and an integer $n \in \mathbb{N}$. It should output the summation above. Do not use for-loops, if- or while-statements.

```
def expectation_max(x,dist,n):
    delta = x[1:] - x[:-1]
    return np.sum(delta*x[:-1]*pdf_max(x[:-1],dist,n))
```

d) **[3 pts]** Test your function on a uniform distribution on the interval $[6, 10]$ for $n = 3$ and 500 (i.e., $k = 499$) points in the interval $[6, 10]$. *The answer should be* ≈ 8.97.

```
a = 6
b = 10
x = np.linspace(a,b,500)
dist = stats.uniform(loc=a,scale=b-a)
n = 3

print(expectation_max(x,dist,n))
```

```
8.969963976048291
```

Question 4 [11 + 1 + 5 = 17 pts]

For data points $(x_i, y_i) \in \mathbb{R}^2$, consider the nonlinear regression model $y_i = f(x_i, \beta) + \epsilon_i$ where $\beta = [\beta_0, \dots, \beta_{d-1}]$ with

$$f(x, \beta) = \sum_{j=0}^{d-1} \beta_j x^j.$$

a) **[11 pts]** Write a function `polynomial_fit` that takes as input an $m \times 2$ matrix D of m data points, with one point (x_i, y_i) on every row, and an integer $d \in \mathbb{N}$. It should output the vector β that best fits the data using `least_squares`, i.e., the vector that solves $\min_\beta \sum_{i=0}^{m-1}(y_i - f(x_i, \beta))^2$ (this formula is only mentioned to recall what `least_squares` does). As initial guess for `least_squares` you should take the all-ones vector. You may use a for-loop in your solution, but points will be deducted for this.

```
def f(x,beta):
    d = np.size(beta)
    exponents = np.arange(d)
    return (x[:,None]**exponents) @ beta

def model(beta,x,y):
    return y - f(x,beta)

def polynomial_fit(D,d):
    x = D[:,0]
    y = D[:,1]

    result = optimize.least_squares(model,x0=np.ones(d),args=(x,y))
    return result.x
```

b) **[1 pts]** Test your function on the input data in the template, and $d = 4$, that prints the optimal vector β that was found. *The solution should be* $\approx [2.77, 5.08, 1.02, -1.01]$.

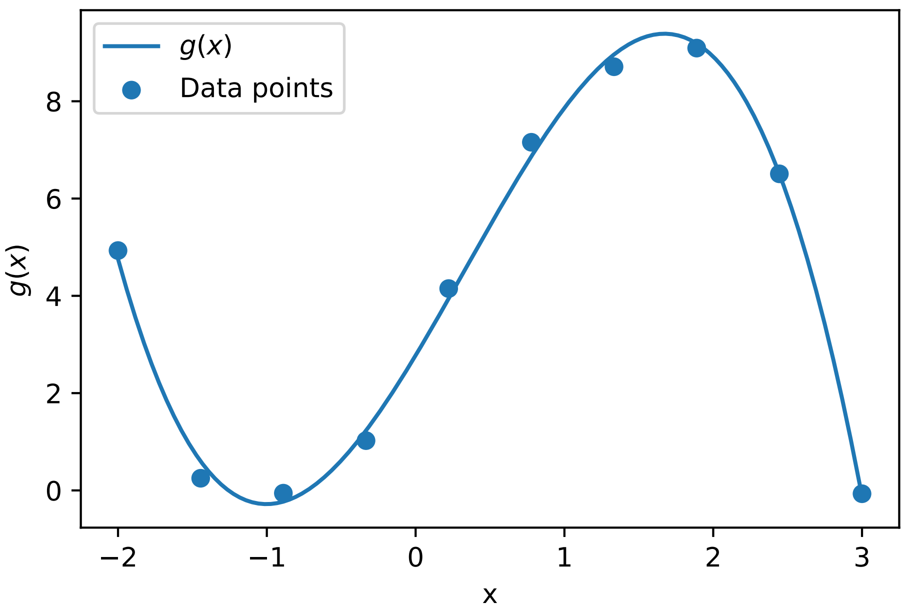
```
from scipy import optimize

D = np.array([
 [-2,          4.93111585],
 [-1.44444444,  0.24990997],
 [-0.88888889, -0.05682958],
 [-0.33333333,  1.02338187],
 [ 0.22222222,  4.14945194],
 [ 0.77777778,  7.15688363],
 [ 1.33333333,  8.70971831],
 [ 1.88888889,  9.09293372],
 [ 2.44444444,  6.50802039],
 [ 3,          -0.06842749]])

d = 4
```

```
beta_fit = polynomial_fit(D,d)
print(beta_fit)
```

```
[ 2.77456629  5.08294531  1.01577513 -1.01074597]
```

c) **[5 pts]** Plot the data points and the polynomial $g(x) = f(x, \beta) = \sum_{j=0}^{d-1} \beta_j x^j$ on the interval $[-2, 3]$ where β is the vector found in part b). *Take* $\beta = [3, 5, -1, 1]$ *if the testing in part b) did not work. Your figure should look like the one below.*

```
x_data = D[:,0]
y_data = D[:,1]

x = np.linspace(-2,3,600)
y = f(x,beta_fit)

#Create the figure
plt.figure()

# Create the plot
plt.plot(x, y, label='$g(x)$')
plt.scatter(x_data, y_data, label='Data points')

# Create labels for axes
plt.xlabel('x')
plt.ylabel('$g(x)$')

# Create the legend with the specified labels
plt.legend()

# Show the plot
plt.show()
```


g(x)
Data points
8
6
4
2
0
g(x)
−2
−1
0
1
2
3
x