Python II - Practice exam 2 (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 5 + 7 + 13 + 25 = 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
import scipy.special as special
import matplotlib.pyplot as plt
```

Question 1 [5 pts]

The function f is defined by

$$f(x) = \begin{cases} -x+1 & \text{if } x < 0 \\ x^2+x+1 & \text{if } 0 \leq x < 1 \\ 3 & \text{if } x \geq 1 \end{cases}$$

Write a function `f()` that takes as input a one-dimensional array $x = [x_0, \dots, x_{n-1}] \in \mathbb{R}^n$ and outputs the one-dimensional array $[f(x_0), \dots, f(x_{n-1})]$. Do not use for-loops, if- or while-statements. *For* $x = [-5, -0.5, 0.5, 4, 9]$ *the output should be* $[6, 1.5, 1.75, 3, 3]$.

```
def f(x):
    return (x < 0)*(-x + 1) + ((x >=0) & (x <= 1))*(x**2 + x + 1) + (x >= 1)*3

# Test input
x = np.array([-5,-0.5,0.5,4,9])
print(f(x))
```

```
[6.   1.5  1.75 3.   3.  ]
```

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 denotes by a_{ij} for $i, j = 0, \dots, n-1$. It should output the following two quantities:

- The row products $\prod_{j=0}^{n-1} a_{ij}$ for $i = 0, \dots, n-1$.

- The quantity $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} a_{ij} e^{x_i x_j}$ where $e \approx 2.71$ is Euler's number.

Do not use for-loops, if- or while-statements. *For* $x = [-1, 1, 2]$ *and* $A = [[-5, 4, 9], [1, 1, 1], [2, 1, 4]]$, *the output should be* $[-180, 1, 8]$ *and* ≈ 225.63.

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

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

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

```
[-180    1    8]
225.625670338062
```

Question 3 [8 + 5 pts]

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

$$F_{\text{max}}(x) = F(x)^n$$

a) **[8 pts]** Write a function `max_median()` that takes as input a continuous probability distribution (a `stats.rv_continuous` object) and integer $n \in \mathbb{N}$. It should output the median (as a scalar number, not a list/array) of the cdf of the random variable X_{max}, which is the solution to the equation $F_{\text{max}}(x) = 0.5$. Use `fsolve()` in your solution with as initial guess the median of the inputted continuous probability distribution. *Hint: Define an auxiliary function that models the equation to be solved.*

```
def equation(x,dist,n):
    return dist.cdf(x)**n - 0.5

def max_median(dist,n):
    result = optimize.fsolve(equation,x0=dist.median(),args=(dist,n))[0]
    return result
```

b) **[5 pts]** Test your function `max_median()` by printing its output for the following inputs:

- Normal distribution with mean $\mu = 5$, standard deviation $\sigma = 2$, and $n = 3$.
- Uniform distribution on the interval $[4, 9]$ with $n = 10$.
- Gamma distribution with shape parameter $a = 3$, scale parameter 4, location parameter 0, and $n = 2$. *If your function works correctly, the outputs should be* $\approx 6.64, 8.67$ *and* 14.63.

```
dist = [stats.norm(loc=5,scale=2), stats.uniform(4,9-4), stats.gamma(a=3, scale=4)]
n = [3, 10, 2]

for i in range(len(n)):
    print(max_median(dist[i],n[i]))
```

```
6.6386572396672205
8.665164957684022
14.62542588068095
```

Question 4 [8 + 8 + 9 pts]

The Maclaurin approximation of order n of the sine function at a given $x \in \mathbb{R}$ is given by

$$M(x,n) = \sum_{k=0}^{n-1} \frac{(-1)^k}{(2k+1)!} x^{2k+1}$$

a) **[8 pts]** Write a function `M()` that takes as input a scalar $x \in \mathbb{R}$ and integer $n \in \mathbb{N}$. It should output the number $M(x,n)$. You can compute the factorial $r! = r(r-1)\cdots 1$ of an integer r with `special.factorial(r)` (assuming you uncommented `import scipy.special as special` in the template). Do not use for-loops in your solution. *For $x = 0.5$ and $n = 5$, the function should output* ≈ 0.48.

```
def M(x,n):
    a = np.arange(0,n)
    sign = (-1)**a
    fact = special.factorial(2*a + 1)
    return np.sum(np.divide(sign,fact)*x**(2*a + 1))

x = 0.5
n = 5
print(M(x,n))
```

```
0.4794255386164159
```

For given $x \in \mathbb{R}$ and $n \in \mathbb{N}$, we define the approximation error with respect to the exponential function as

$$\text{Error}(x,n) = |M(x,n) - \sin(x)|.$$

b) **[8 pts]** Write a function `max_error` that takes as input scalars $a, b \in \mathbb{R}$, with $a < b$, and integer n, and returns the value of the maximization problem

$$E(a,b,n) = \max_{x \in [a,b]} \text{Error}(x,n).$$

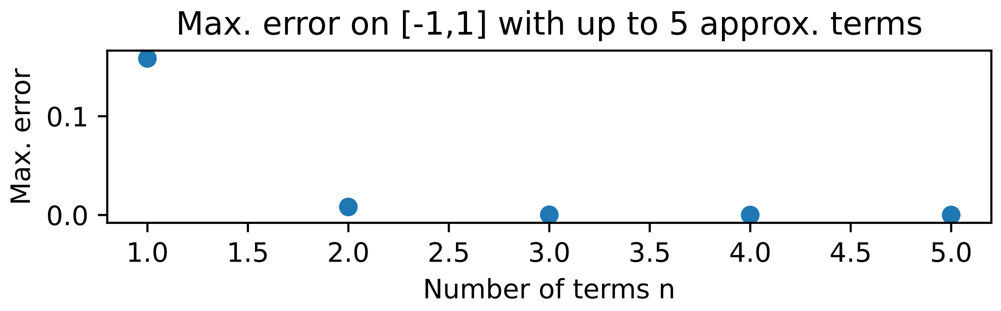
That is, it returns the maximum error over the interval $[a,b]$. Use `minimize_scalar()` with the `bounded`-method in your solution. *For $a = -4, b = 2$ and $n = 5$, your function should output* ≈ 0.095.

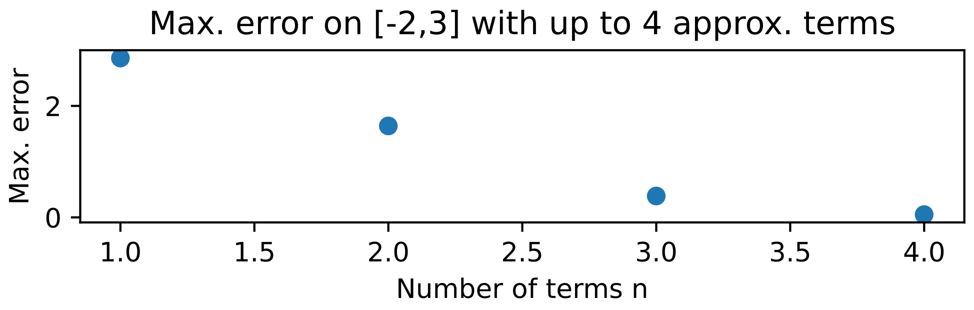
```
def objective(x,n):
    return -np.abs(np.sin(x)-M(x,n))

def max_error(a,b,n):
    result = optimize.minimize_scalar(objective,bounds=[a,b],args=(n),method='bounded')
    return -result.fun



a = -4
b = 2
n = 5
print(max_error(a,b,n))
```

```
0.09507280323290379
```

c) **[9 pts]** Create a figure that for both parameter combinations $(a, b, N) \in \{(-1, 1, 5), (-2, 3, 4)\}$ creates a subplot in the figure containing a scatter plot of the values $E(a, b, n)$ for $n = 1, \dots, N$ with subplot title "Max. error on $[a, b]$ with up to N approx. terms" (where a, b, N should be replaced by their respective values). You are allowed to use for-loops. Your figure should look like this:

```
r = 2
fig = plt.figure()


combi = [(-1,1,5),(-2,3,4)]
for p in range(r):
    a, b, N = combi[p][0], combi[p][1], combi[p][2]
    result = np.zeros(N)
    for i in range(1,N+1):
        result[i-1] = max_error(a,b,i)
    x = np.arange(1,N+1)

    # Add subplot
    apx = fig.add_subplot(r,1,p+1)
    apx.scatter(x,result)
    apx.set_xlabel('Number of terms n')
    apx.set_ylabel('Max. error')
    apx.set_title('Max. error on [%.i,%.i] with up to %.i approx. terms' % (a,b,N))

# Tighten layout
plt.tight_layout()

# Show plot
plt.show()
```


Max. error on [-1,1] with up to 5 approx. terms
Max. error
0.1
0.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Number of terms n


Max. error on [-2,3] with up to 4 approx. terms
Max. error
2
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Number of terms n