Assignment #6

Answer all the questions below. Submit a Python/R **Notebook** (Jupyter or Colab).

Practice on Jacobian & Hessian Matrices computation using Python/R.

1. Jacobian Matrix

The **Jacobian matrix** represents the first-order partial derivatives of a vector-valued function.

- Suppose you have a function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$, where $\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})]$ is a function that takes n inputs and gives m outputs.
- The **Jacobian matrix**, $\mathbf{J}(\mathbf{x})$, is an $m \times n$ matrix, where each entry J_{ij} is the partial derivative of the i-th function $f_i(\mathbf{x})$ with respect to the j-th variable x_j:

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

2. Hessian Matrix

The **Hessian matrix** is the matrix of second-order partial derivatives for a scalar-valued function.

- Suppose you have a function $f : \mathbb{R}^n \to \mathbb{R}$, which maps an n-dimensional vector to a scalar.
- The **Hessian matrix**, $\mathbf{H}(f)(\mathbf{x})$, is an $n \times n$ symmetric matrix, where each entry H_{ij} is the second-order partial derivative of the function $f(\mathbf{x})$ with respect to the variables x_i and x_j:

$$\mathbf{H}(f)(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Calculate the **Jacobian** and **Hessian** matrices in Python using libraries like **SymPy** (for **symbolic differentiation**) and **NumPy** (for **numerical evaluation**).

For a function $f : \mathbb{R}^2 \to \mathbb{R}^2$:

$$f(x, y) = \begin{bmatrix} x^2 + y \\ \sin(xy) \end{bmatrix}$$

(1) Compute/build the **Jacobian matrix** using **SymPy**. (if sympy is not installed -→ **pip install sympy**)

```
import sympy as sp
```

Perform the following steps:
1_ Define the symbolic variables
2_ Define the vector-valued function f(x, y)
3_ Calculate/create the Jacobian matrix
4_ Print the Jacobian matrix
5_ Substitute specific values of x and y (x=2 & y=3)

The output should look like this:

```
Jacobian matrix:
⎡2⋅x   1⎤
⎢        ⎥
⎣y⋅cos(x⋅y)  x⋅cos(x⋅y)⎦

Jacobian matrix at x=2, y=3:
⎡4   1 ⎤
⎢        ⎥
⎣-2⋅cos(6)  2⋅cos(6)⎦
```

(2) Compute/build the **Hessian matrix**
Perform the following steps:
1_ Define the scalar-valued function f(x, y)
2_ Calculate the Hessian matrix
3_ Print the Hessian matrix
4_ Substitute specific values of x and y (x=2 & y=3)

The output should look like this:

```
Hessian matrix:
⎡2  cos(x⋅y) - y⋅sin(x⋅y)⎤
⎢                         ⎥
⎣cos(x⋅y) - y⋅sin(x⋅y)  2 - x⋅sin(x⋅y)⎦

Hessian matrix at x=2, y=3:
⎡2  cos(6) + 3⋅sin(6)⎤
⎢                     ⎥
⎣cos(6) + 3⋅sin(6)   2 + 2⋅sin(6)⎦
```

Numerically computing the Jacobian or Hessian at given points (rather than symbolically), you can use NumPy and SciPy.

(3) Numerical Jacobian:

```
import numpy as np
from scipy.optimize import approx_fprime
```

Perform the following steps:
1_ Define the function f: R^2 -> R^2
2_ Define a point to evaluate the Jacobian (x=2 & y=3)
3_ Set epsilon (e), a small perturbation for finite difference approximation
(Choosing epsilon make use of this command: **np.finfo(float).eps**)

```
>>> print(epsilon)
1.4901161193847656e-08
```

4_ Calculate the Jacobian matrix numerically

The output should look like this:

```
Numerical Jacobian at X = [2, 3]:
 [4. 1.]
```

(4) Numerical Hessian:

A way to approximate the Hessian using **NumPy** and **numdifftools**: (**pip install numdifftools**)

```
import numpy as np
import numdifftools as nd
```

Perform the following steps:
1_ Define the scalar-valued function
2_ Create a Hessian object from numdifftools
3_ Define the point where we want to calculate the Hessian (x=2, y=3)
4_ Calculate the Hessian matrix

The output should look like this:

```
Numerical Hessian at X = [2, 3]:
[[ 2.          0.96017029]
 [ 0.96017029  2.64025543]]
```