CL603: Optimization
Gradient Computation

Course Instructor: Prof. Mani Bhushan (mbhushan@iitb.ac.in)
Course TAs: Om Prakash (prakash.om@iitb.ac.in),
Varun (varun.p@iitb.ac.in)

January 24, 2020

Date Due: 31 Jan 2020 (Between 12.30-1.30 PM; You have to upload it on moodle from the department computer room)

This tutorial will get you started with the Python or MATLAB programming relevant for this course.
Specific Aim: To understand and be able to write a function in Python or MATLAB for computing the gradient of any function.

Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a scalar function. The gradient $\mathbf{g}$ of f is a $n \times 1$ vector, defined as follows:

$$\mathbf{g} = \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \tag{1}$$

Let $\mathbf{x}^*$ be the value of $\mathbf{x}$ at which the gradient needs to be computed. The partial derivatives in the gradient can be computed numerically using central difference scheme as:

$$\left.\frac{\partial f}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}^*} \approx \frac{f(\mathbf{x}^* + h\mathbf{e_j}) - f(\mathbf{x}^* - h\mathbf{e_j})}{2h} \tag{2}$$

$$\text{where,} \qquad \mathbf{e_j} = \begin{bmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}^T \tag{3}$$

In Eq.2, the derivative is approximated by perturbing the j^{th} variable, hence vector $\mathbf{e_j}$ in Eq.3 has element 1 at j^{th} position and 0s elsewhere. The perturbation h in Eq. 2 is a small number.

The given Python and MATLAB codes (screen shots have been included) do the following:

1. *user_func* evaluates the function whose gradient is to be computed. In this illustration, the function to be evaluated is as follows:

$$f = x_1^2 + x_2 \tag{4}$$

2. *gradient_compute* evaluates the approximate derivative of the function defined in *user_func* at a point $\mathbf{x}^*$ using the central difference scheme.

3. *main_gradient* declares the point $\mathbf{x}^*$ and obtains the required gradient using the previously mentioned functions.

Do the following in Python or MATLAB:

1. Understand and write the given code.
2. Modify the given code such as to evaluate the gradient of the following function:

$$f = 2x_2x_3\sin(x_1) + 3x_1\sin(x_3)\cos(x_2) \tag{5}$$

at a point $\mathbf{x}^* = [-1 \;\; 1 \;\; 1]^T$.

3. Modify the given code to evaluate the gradient using the **forward difference scheme** given as:

$$\left.\frac{\partial f}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}^*} \approx \frac{f(\mathbf{x}^* + h\mathbf{e_j}) - f(\mathbf{x}^*)}{h} \tag{6}$$

$$\text{where,} \qquad \mathbf{e_j} = \begin{bmatrix}0 & \cdots & 0 & 1 & 0 & \cdots & 0\end{bmatrix}^T \tag{7}$$

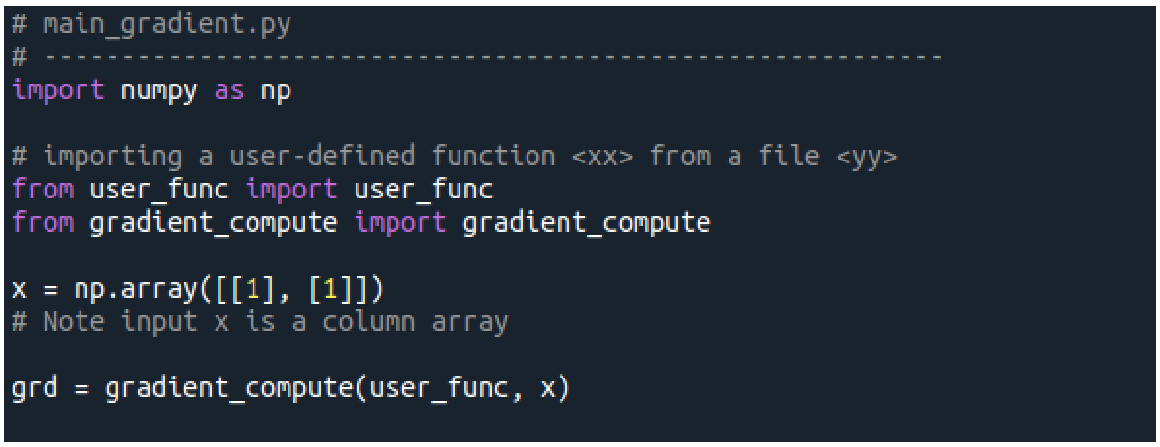
4. Create a function '*analyt_grad*' which takes $\mathbf{x}^*$ as input and returns the analytically computed gradient for the specific function given in Eq. 5 (does not need to be for any generic function). Note that you can hard-code the analytical gradient in this function and this can be used to check the results you obtain from your numerical computation.

Following will be the learning at the end of tutorial:

- Calling functions
- Creating your own function
- Numerical gradient computation

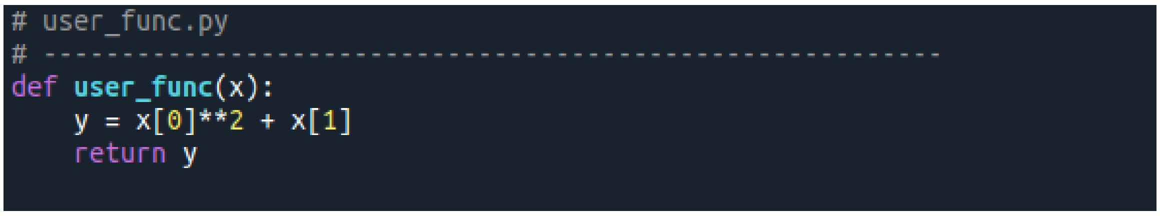
Given Codes

Python codes:



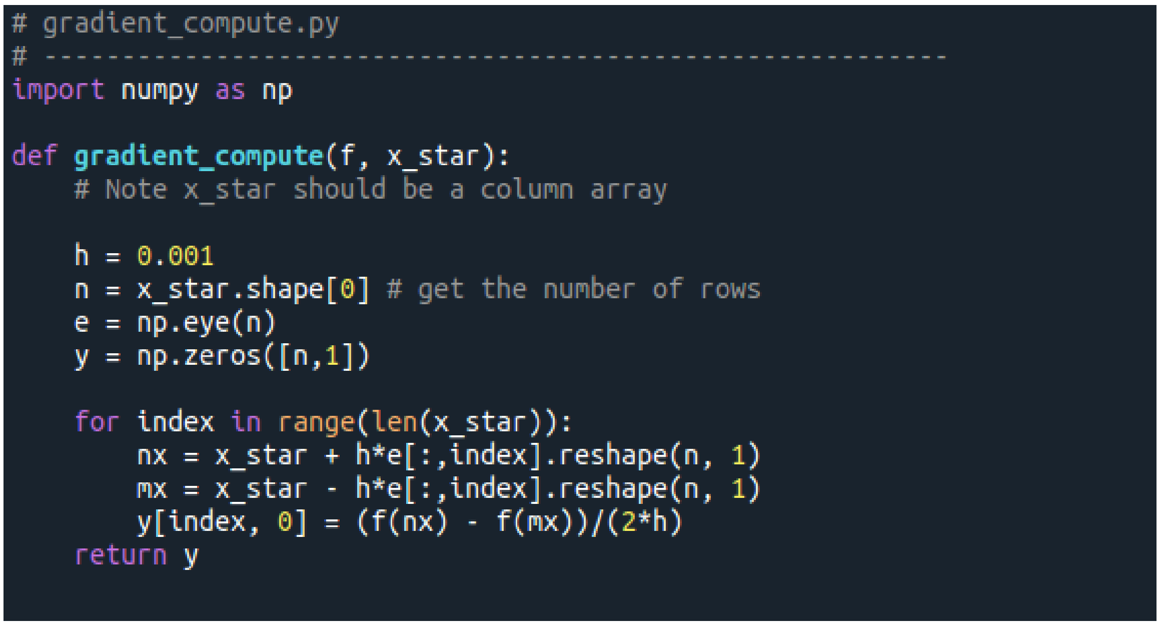
```python
# main_gradient.py
# ------------------------------------------------------------
import numpy as np

# importing a user-defined function <xx> from a file <yy>
from user_func import user_func
from gradient_compute import gradient_compute

x = np.array([[1], [1]])
# Note input x is a column array

grd = gradient_compute(user_func, x)
```

Figure 1: Python code: main_gradient.py



```python
# user_func.py
# ----------------------------------------------------------
def user_func(x):
    y = x[0]**2 + x[1]
    return y
```

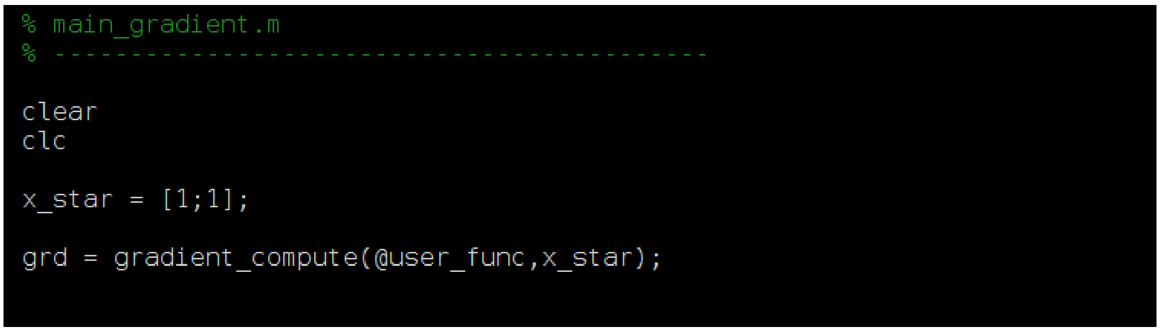
Figure 2: Python code: user_func.py



```python
# gradient_compute.py
# ----------------------------------------------------------
import numpy as np

def gradient_compute(f, x_star):
    # Note x_star should be a column array

    h = 0.001
    n = x_star.shape[0] # get the number of rows
    e = np.eye(n)
    y = np.zeros([n,1])

    for index in range(len(x_star)):
        nx = x_star + h*e[:,index].reshape(n, 1)
        mx = x_star - h*e[:,index].reshape(n, 1)
        y[index, 0] = (f(nx) - f(mx))/(2*h)
    return y
```

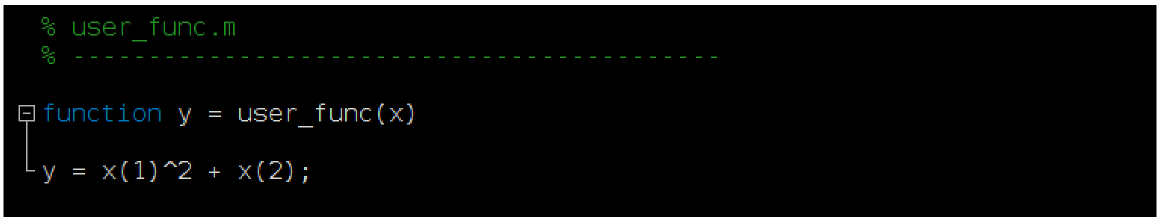
Figure 3: Python code: gradient_compute.py

MATLAB codes:



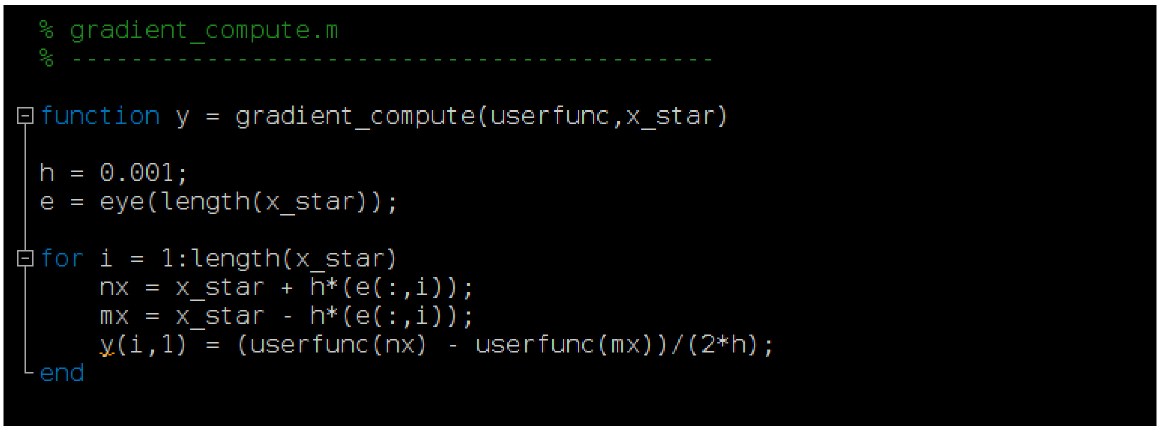
```matlab
% main_gradient.m
% -------------------------------------------

clear
clc

x_star = [1;1];

grd = gradient_compute(@user_func,x_star);
```

Figure 4: MATLAB code: main_gradient.m



```
% user_func.m
% -------------------------------------------

function y = user_func(x)

y = x(1)^2 + x(2);
```

Figure 5: MATLAB code: user_func.m



```
% gradient_compute.m
% -------------------------------------------

function y = gradient_compute(userfunc,x_star)

h = 0.001;
e = eye(length(x_star));

for i = 1:length(x_star)
    nx = x_star + h*(e(:,i));
    mx = x_star - h*(e(:,i));
    y(i,1) = (userfunc(nx) - userfunc(mx))/(2*h);
end
```

Figure 6: MATLAB code: gradient_compute.m

Learning is fun. Best of Luck!