PDE_1

May 21, 2023

1 PDE first exercise

In this first exercise, we will focus our work on to finite differences and plotting tool.

First we import our mandatory module

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

We will use the following equation as our reference function and solution:

$$f = sin(20x) + exp(-100x) + 100 * x^2$$

In this exercise you have to: 1. make a function to compute f 2. make a function which computes the first derivative of f with respect to x 3. plot the function and its derivatives for $0 \leq x \leq 1$ in a subplot (i.e one plot for f and one plot for its derivative)

Using a simple `for` loop:

4. make a function to compute the first derivative of f using a backward difference
5. make a function to compute the first derivative of f using a forward difference
6. make a function to compute the first derivative of f using a centered difference
7. compute the error commited by the different the finite differences and plot them in a subplot

Finally, replicate the question 4. without using a `for` loop

In the following cell we define a mathematical expression for the function f

[ ]:
```
def func(x):
    return np.sin(20.0*x)+np.exp(-100.0*x)+100.0*x**2.0
```

We define next a function for the first derivative of f with respect to x

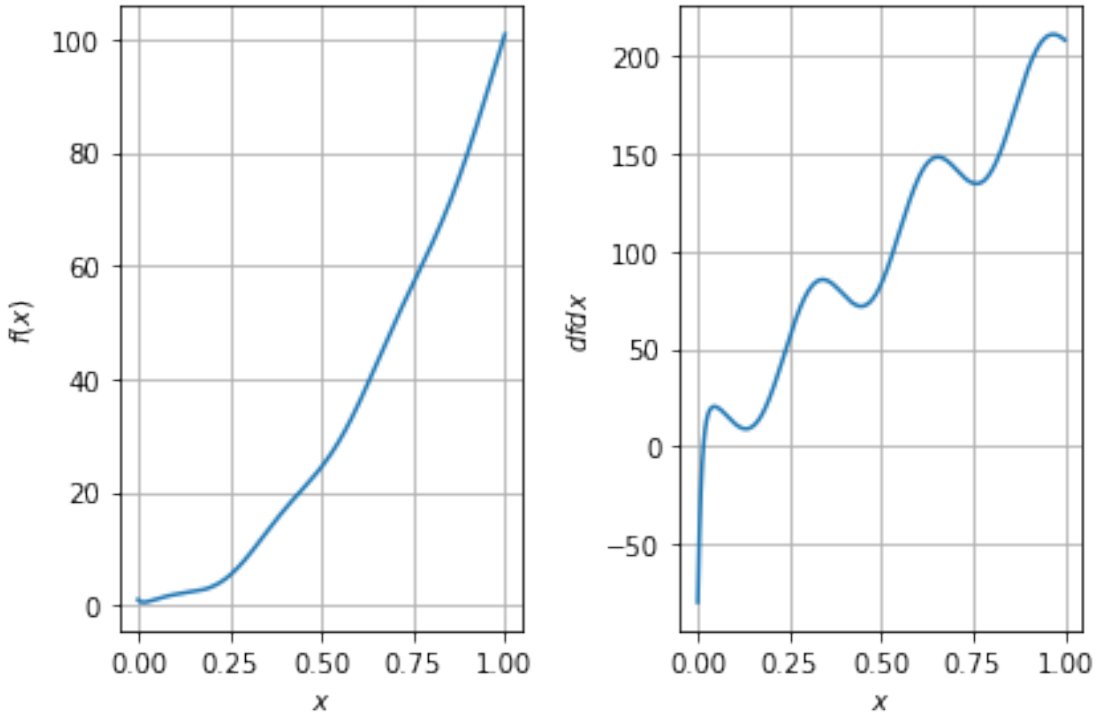
[ ]:
```
def der_1_func(x):
    return 20.0*np.cos(20.0*x)-100.0*np.exp(-100.0*x)+200.0*x
```

We then plot the function and its derivative in a subplot.

[ ]:
```
x = np.linspace(0.0,1.0,1000)
fig, axs = plt.subplots(1,2)
axs[0].plot(x,func(x))
axs[0].grid()
```

```
axs[1].plot(x,der_1_func(x))
axs[1].grid()
axs[1].set_xlabel(r'$x$')
axs[0].set_ylabel(r'$f(x)$')

axs[0].set_xlabel(r'$x$')
axs[1].set_ylabel(r'$dfdx$')
fig.tight_layout()
```



Next, we define three cells which compute the backward, forward and central differences approximation of the first order derivative of f. Since the finite difference schemes access either the previous or/and next state (i.e i-1 or i+1), we must take special care at the borders of the domain. This is done by simply skipping the first or last row of the derivate and setting it to zero.

[ ]:
```
def get_der_B(x,f):
    der = f*0.0
    for i in range(1,len(x)):
        der[i] = (f[i]-f[i-1])/(x[i]-x[i-1])
    return der
```

[ ]:
```
def get_der_F(x,f):
    der = f*0.0
    for i in range(0,len(x)-1):
        der[i] = (f[i+1]-f[i])/(x[i+1]-x[i])
```

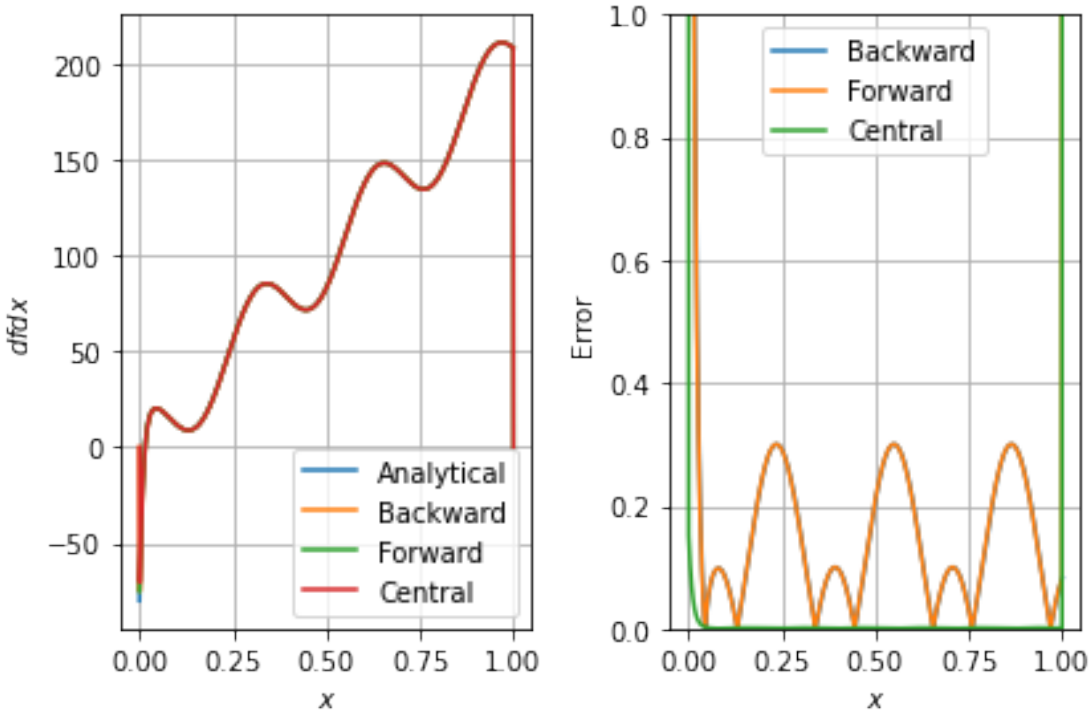
```
    return der
```

[ ]:
```
def get_der_C(x,f):
    der = f*0.0
    for i in range(1,len(x)-1):
        der[i] = (f[i+1]-f[i-1])/(x[i+1]-x[i-1])
    return der
```

We then plot the numerical derivatives compared to the analytical one as well as a simple error measurement.

[ ]:
```
x = np.linspace(0.0,1.0,1000)
fig, axs = plt.subplots(1,2)
#
axs[0].plot(x,der_1_func(x),label='Analytical')
axs[0].plot(x,get_der_B(x,func(x)),label='Backward')
axs[0].plot(x,get_der_F(x,func(x)),label='Forward')
axs[0].plot(x,get_der_C(x,func(x)),label='Central')
axs[0].grid()
axs[0].legend()
axs[0].set_xlabel(r'$x$')
axs[0].set_ylabel(r'$dfdx$')
#
axs[1].plot(x,np.sqrt((der_1_func(x)-get_der_B(x,func(x)))**2.
 ↪0),label='Backward')
axs[1].plot(x,np.sqrt((der_1_func(x)-get_der_F(x,func(x)))**2.
 ↪0),label='Forward')
axs[1].plot(x,np.sqrt((der_1_func(x)-get_der_C(x,func(x)))**2.
 ↪0),label='Central')
axs[1].set_ylim([0,1])
axs[1].grid()
axs[1].legend()
axs[1].set_xlabel(r'$x$')
axs[1].set_ylabel('Error')

fig.tight_layout()
```



We can see that the derivates computed by the different techniques are almost identical except for the starting and end points. The right-hand side graph shows the error commited by the numerical schemes. The backward and forward schemes commit the same error while the centered difference scheme predicts a lower error.

These differences are linked to the order of approximation which is a direct result from the Taylor serie expansion and will be discussed further during the lectures.

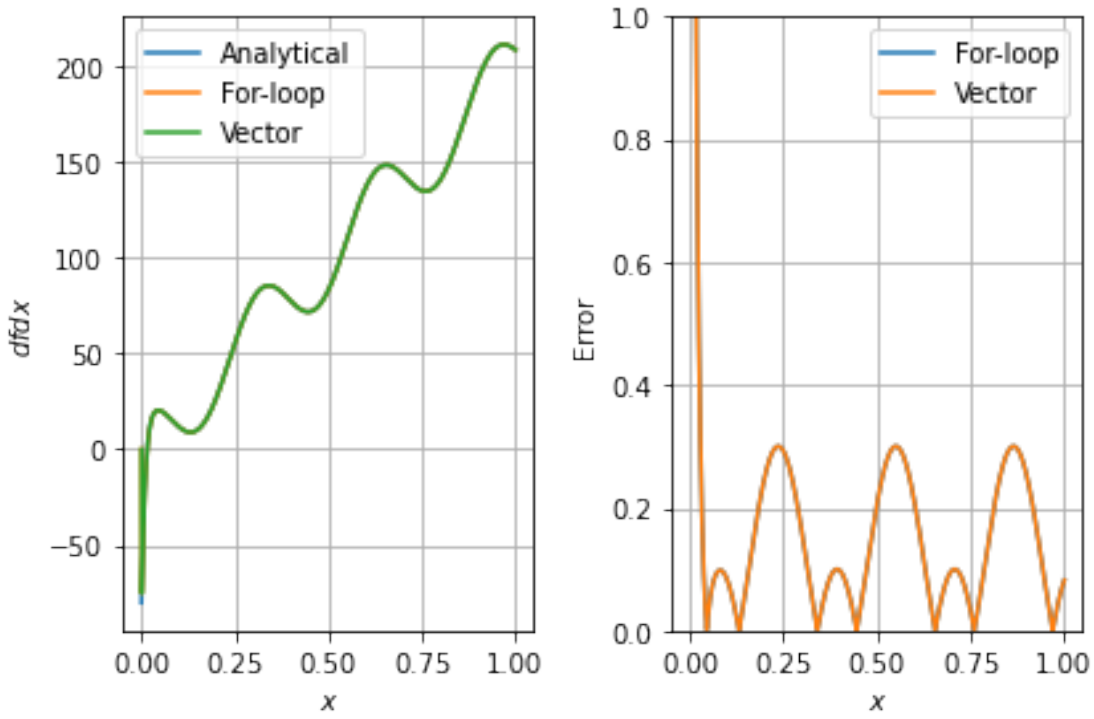
The last question is linked to the computation of a backward difference without using a for loop. This operation is carried out using the so-called "array slicing". We simply "translate" the vector by one row to get the "i-1" index.

[ ]:
```
def get_der_B_vector(x,f):
    der = f*0.0
    der[1:] = (f[1:]-f[:-1])/(x[1:]-x[:-1])
    return der
```

The next cell plots the derivatives obtained by either the analytical, the for-loop based backward and the vector based backward differences. As expected the two backward differences yield the exact same results.

[ ]:
```
x = np.linspace(0.0,1.0,1000)
fig, axs = plt.subplots(1,2)
axs[0].plot(x,der_1_func(x),label='Analytical')
axs[0].plot(x,get_der_B(x,func(x)),label='For-loop')
```

```
axs[0].plot(x,get_der_B_vector(x,func(x)),label='Vector')
axs[0].grid()
axs[0].set_xlabel(r'$x$')
axs[0].set_ylabel(r'$dfdx$')
axs[0].legend()
#
axs[1].plot(x[1:],np.sqrt((der_1_func(x)-get_der_B(x,func(x)))**2.0)[1:
 ↪],label='For-loop')
axs[1].plot(x[1:],np.sqrt((der_1_func(x)-get_der_B_vector(x,func(x)))**2.0)[1:
 ↪],label='Vector')
axs[1].grid()
axs[1].set_xlabel(r'$x$')
axs[1].set_ylabel('Error')
axs[1].legend()
axs[1].set_ylim([0,1])
fig.tight_layout()
```



[ ]: