1_differentiation

November 27, 2023

1 Numerical Differentiation

We did already have a look at the numerical differentiation in Lecture 3. We therefore don't have to extend that to much here. Yet we want to formalize the numerical differentiation a bit.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
%config InlineBackend.figure_format = 'retina'

plt.rcParams.update({'font.size': 12,
                     'axes.titlesize': 18,
                     'axes.labelsize': 16,
                     'axes.labelpad': 14,
                     'lines.linewidth': 1,
                     'lines.markersize': 10,
                     'xtick.labelsize' : 16,
                     'ytick.labelsize' : 16,
                     'xtick.top' : True,
                     'xtick.direction' : 'in',
                     'ytick.right' : True,
                     'ytick.direction' : 'in',})
```

1.1 First order derivative

Our previous way of finding the derivative was based on its definition

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + x_0) - f(x)}{\Delta x} \tag{1}$$

such that, if we don't take the limit we can approximate the derivative by

$$f'_i \approx \frac{f_{i+1} - f_i}{\Delta x} \tag{2}$$

Here we look to the right of the current posiiton i and devide by the interval Δx.

It is not difficult to see that the resulting local error δ at each step is given by

$$= f_{i+1} - f_i = \frac{1}{2}\Delta x^2 f''$$

Some better expression may be found by the Taylor expansion:

$$f(x) = f(x_0) + (x - x_0)f'(x) + \frac{(x - x_0)^2}{2!}f''(x) + \frac{(x - x_0)^3}{3!}f^{(3)}(x) + ... \tag{3}$$

which gives in discrete notation

$$f_{i+1} = f_i + \Delta x f_i' + \frac{\Delta x^2}{2!}f_i'' + \frac{\Delta x^3}{3!}f_i^{(3)} + ... \tag{4}$$

The same can be done to obtain the function value at $i - 1$

$$f_{i-1} = f_i - \Delta x f_i' + \frac{\Delta x^2}{2!}f_i'' - \frac{\Delta x^3}{3!}f_i^{(3)} + ... \tag{5}$$

which when neglecting the last term on the right side yields

$$f_i' \approx \frac{f_{i+1} - f_{i-1}}{2\Delta x} \tag{6}$$

This is similar to the formula we obtained from the definition of the derivative. Here, however, we use the function value to the left and the right of the position i and twice the interval, which actually improves accuracy.

[23]:
```
def D(f, x, h=1.e-12, *params):
    return (f(x+h, *params)-f(x-h, *params))/(2*h)
```

One may continue that type of derivation now to obtain higher order approximation of the first derivative with better accuracy. For that purpose you may calculate now $f_{i\pm2}$ and combining that with $f_{i+1} - f_{i-1}$ will lead to

$$f_i' = \frac{1}{12\Delta x}(f_{i-2} - 8f_{i-1} + 8f_{i+1} - f_{i+2}) \tag{7}$$

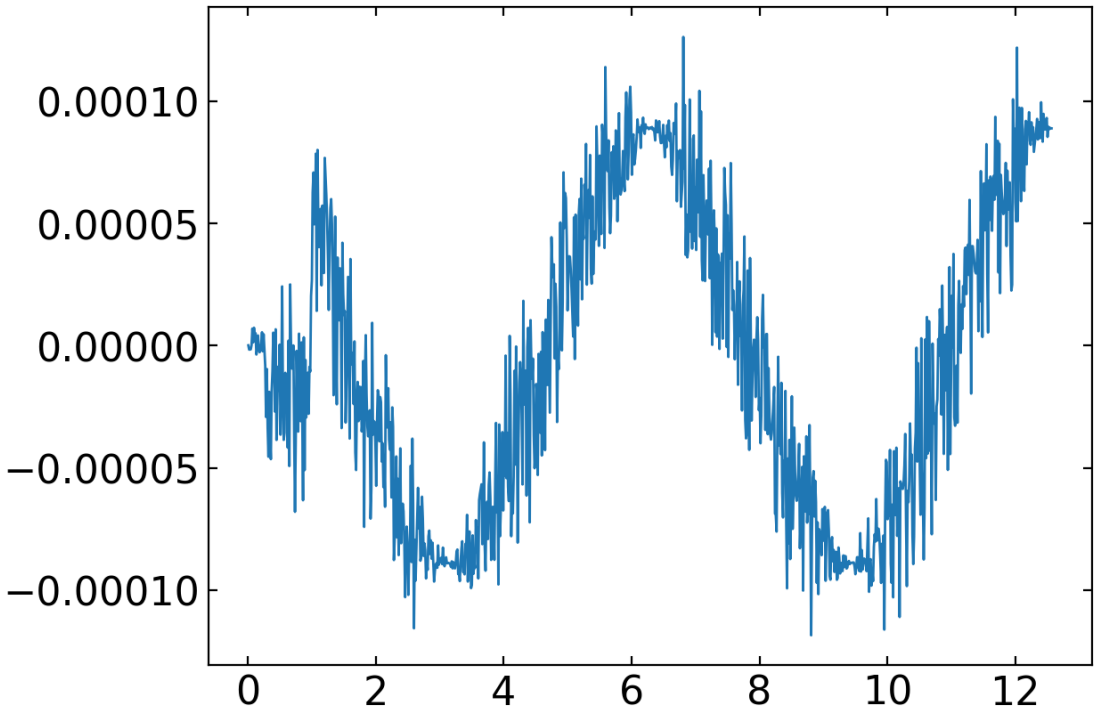
This can be used to give better values for the first derivative.

[27]:
```
def f(x):
    return(np.sin(x))
```

[28]:
```
x=np.linspace(0.01,np.pi*4,1000)
```

[29]:
```
#plt.plot(x,f(x))
plt.plot(x,D(f,x)-np.cos(x))
#plt.ylim(-2000,2000)
```

[29]:
```
[<matplotlib.lines.Line2D at 0x7ff9890b18e0>]
```



1.1.1 Matrix version of the first derivative

If we supply to the above function an array of positions x_i at which we would like to calculate the derivative, then we obtain and array of derivative values. We can write this procedure also in a different way, which will be helpful for solving differential equation later.

If we consider the above finite difference formulas for a set of positions x_i, we can represent the first derivative at these positions by matrix operation as well:

$$f' = \frac{1}{\Delta x} \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{bmatrix} = \begin{bmatrix} (f_2 - f_1)/\Delta x \\ (f_3 - f_2)/\Delta x \\ (f_4 - f_3)/\Delta x \\ (f_5 - f_4)/\Delta x \\ (f_6 - f_5)/\Delta x \\ (0 - f_6)/\Delta x \end{bmatrix}$$

Note that we took here the derivative only to the right side! Each row of the matrix multiplied by the vector containing the positions is then containing the derivative of the function f at the position x_i and the resulting vector represents the deravitave in a certain position region.

We will demonstrate how to generate such a mtrix with the `SciPy` module below.

1.2 Second order derivative

Higher order derivatives are also available from the same process. By adding f_{i+2} and f_{i-1} we arrive at

$$f_i'' \approx \frac{f_{i-1} - 2f_i + f_{i+1}}{\Delta x^2} \tag{8}$$

gives the basic equation for calculating the second order derivative and the next order may be obtained from

$$f_i'' \approx \frac{1}{12\Delta x^2}(-f_{i-2} + 16f_{i-1} - 30f_i + 16f_{i+1} - f_{i+2}) \tag{9}$$

which is again better than our previous formula

$$f_i'' \approx \frac{f_{i+2} - 2f_{i+1} + f_i}{\Delta x^2} \tag{10}$$

1.3 SciPy Module

Of course, we are not the first to define some functions for calculating the derivative of functions numerically. This is already implemented in different modules. One module is the above mentioned `SciPy` module.

The `SciPy` module provides the method `derivative`, which we can call with

`derivative(f,x,dx=1.0,n=1):`

This will calculate the n*th* derivative of the function f at the position x with a intervall $dx = 1.0$ (default value).

[30]:
```
## the derivative method is hidden in the `misc` sub-module of `SciPy`.
from scipy.misc import derivative
```

[31]:
```
derivative(np.sin,np.pi,dx=0.000001,n=2,order=15)
```

```
/tmp/ipykernel_2153207/3594364828.py:1: DeprecationWarning:
scipy.misc.derivative is deprecated in SciPy v1.10.0; and will be completely
removed in SciPy v1.12.0. You may consider using findiff:
https://github.com/maroba/findiff or numdifftools:
https://github.com/pbrod/numdifftools
  derivative(np.sin,np.pi,dx=0.000001,n=2,order=15)
```

[31]:
```
-1.7089939196257686e-09
```

1.3.1 Matrix version

The `SciPy` module also allows us to construct the matrices as mentioned above. We will need the `diags` method from the `SciPy` module for that purpose.

[32]:
```
from scipy.sparse import diags
```

Lets assume, we want to calculate the derivative of the `sin` function at certain positions.

[38]:
```
N=100
x=np.linspace(-5,5,N)
y=np.sin(x)
```

The `diags` function uses a set of numbers, that shoud be distributed along the diagonal of the matrix. If you supply a list like in the example below, the numbers are distributed using the offsets as defined in the second list. The `shape` keyword defines the shape of the matrix. Try the example in the over next cell with the `.todense()` suffix. This converts the otherwise unreadable sparse output to a readable matrix form.

[46]:
```
diags([-1, 1], [0, 1],shape=(10,10)).todense()
```

[46]:
```
matrix([[-1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
        [ 0., -1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
        [ 0.,  0., -1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.],
        [ 0.,  0.,  0., -1.,  1.,  0.,  0.,  0.,  0.,  0.],
        [ 0.,  0.,  0.,  0., -1.,  1.,  0.,  0.,  0.,  0.],
        [ 0.,  0.,  0.,  0.,  0., -1.,  1.,  0.,  0.,  0.],
        [ 0.,  0.,  0.,  0.,  0.,  0., -1.,  1.,  0.,  0.],
        [ 0.,  0.,  0.,  0.,  0.,  0.,  0., -1.,  1.,  0.],
        [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., -1.,  1.],
        [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., -1.]])
```

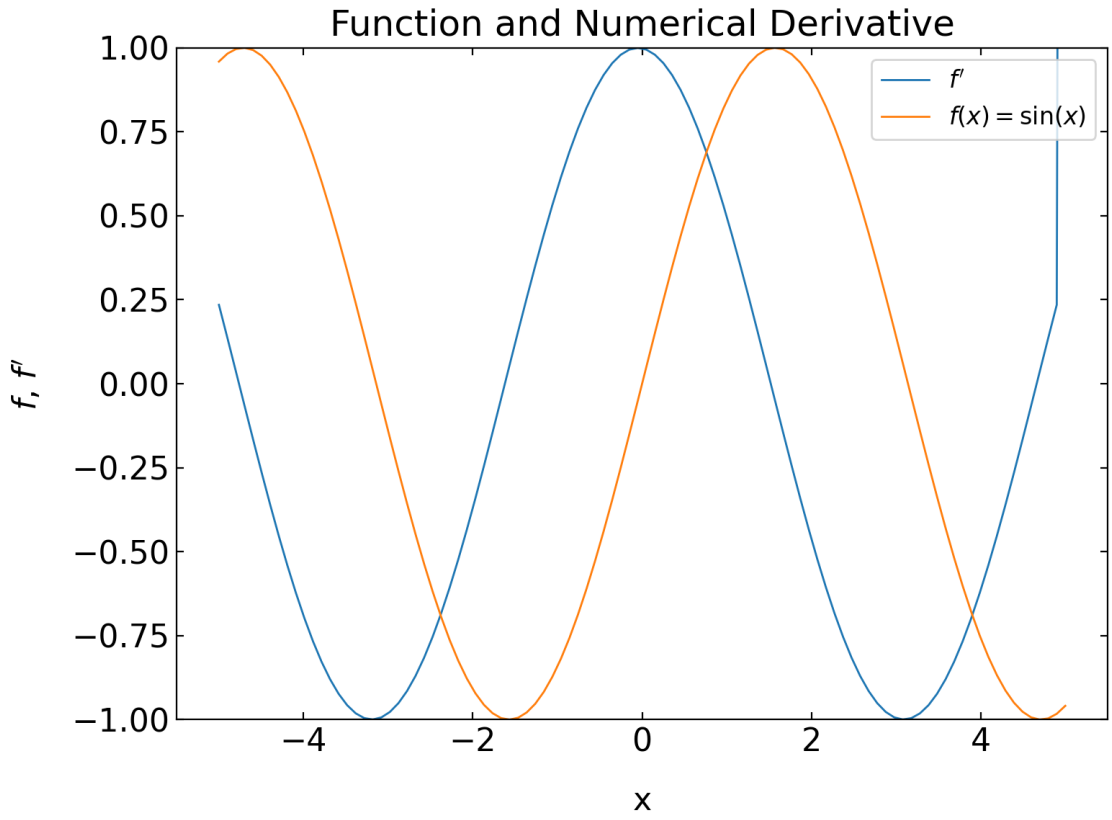
To comply with out previous definition of N=100 data point and the intervall dx, we define

[39]:
```
m=diags([-1., 0., 1.], [0,-1, 1],shape=(N,N))/(x[1]-x[0])
```

The derivative is then only a simple elementwise matrix multiplication.

[47]:
```
diff=m*y
```

[42]:
```
plt.figure(figsize=(8,6))
plt.plot(x[:],diff[:],label='$f^{\prime}$')
plt.plot(x,y,label='$f(x)=\sin(x)$')
plt.xlabel('x')
plt.ylabel('$f$, $f^{\prime}$')
plt.legend()
plt.ylim(-1,1)
plt.title('Function and Numerical Derivative')
plt.tight_layout()
plt.show()
```



Check yourself, that the following line of code will calculate the second derivative.

```
m=diags([-2., 1., 1.], [0,-1, 1], shape=(100, 100))
```