Python 10

Elias Wachmann

2024

Content

1. Differentiation in Python
2. Integration in Python
3. Numerical Solution of Initial Value Problems
4. File I/O (BONUS)
5. ASCII Encoding (BONUS)
6. pickle (BONUS)

Differentiation in Python

Numpy `numpy.gradient`

The `numpy.gradient` function calculates the gradient of an array using finite differences. `numpy.gradient`

$$\frac{\partial f}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h} \tag{1}$$

where h is a small step size. For a one-dimensional array y with spacing dx, the gradient is calculated as:

$$\text{gradient[i]} = \frac{y[\text{i}+1] - y[\text{i}-1]}{2 \cdot dx} \tag{2}$$

Numpy `numpy.gradient` Example

```python
import numpy as np

x = np.array([1, 2, 4, 7, 11])
gradient = np.gradient(x)
print("Gradient:", gradient)
```

Scipy `scipy.misc.derivative`

The `scipy.misc.derivative` function calculates the numerical derivative of a function.
`scipy.misc.derivative` This function uses the central difference formula: see Equation 1 or Finite difference

```python
from scipy.misc import derivative

def f(x):
    return x**3 + x**2

deriv = derivative(f, 1.0, dx=1e-6)
print("Derivative:", deriv)
```

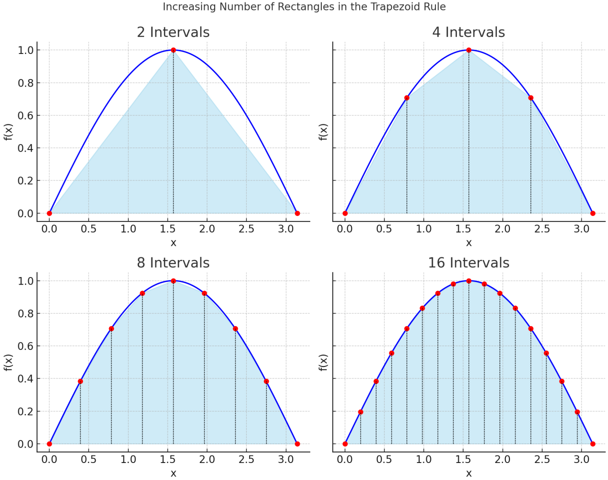
Integration in Python

Scipy `scipy.integrate.trapezoid`

The `scipy.integrate.trapezoid` function computes the trapezoidal rule to approximate the integral. `scipy.trapezoid` It uses the following approximate formula:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}\left(f(a)+f(b)\right)$$

Intuition behind the Trapezoid Rule



Scipy scipy.integrate.trapezoid

```python
import numpy as np
from scipy.integrate import trapezoid

x = np.linspace(0, np.pi, 100)
y = np.sin(x)
integral = trapezoid(y, x)
print("Integral using scipy.trapezoid:",
    integral)
```

Self-written Trapezoid Rule

An implementation of the trapezoid rule from scratch. You have to do that in one of the exercises.

Translate the intuition / formula into maybe a loop or do it entierly vectorized.

Scipy `scipy.integrate.simpson`

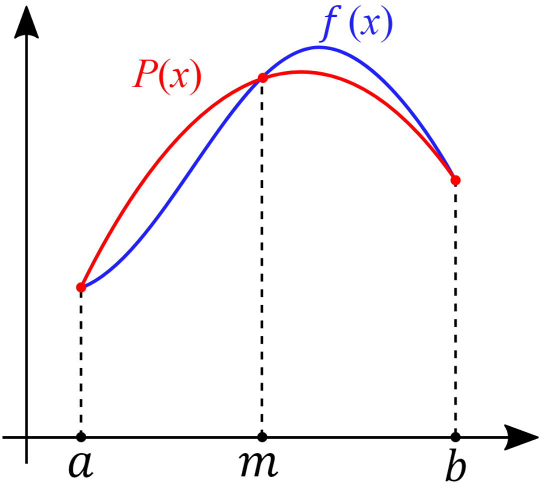
The `scipy.integrate.simpson` function computes the Simpson's rule to approximate the integral.
`scipy.simpson` Another way to approximate the

integral is using Simpson's rule:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

Quadratic interpolation is used to approximate the function between the points a and b.

Simpson rule intuition



Scipy `scipy.integrate.simpson`

```python
import numpy as np
from scipy.integrate import simpson

x = np.linspace(0, np.pi, 100)
y = np.sin(x)
integral = simpson(y, x)
print("Integral using scipy.simpson:",
    integral)
```

Numerical Solution of Initial Value Problems

Mathematical Introduction

For initial value problems, we discretize the time using $t_n = t_0 + n\Delta t$ and introduce the notation $y_n = y(t_n)$. Thus, we write

$$\dot{y}_n = f(t_n, y_n)$$

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} dt' f(t', y(t'))$$

which is still exact. We can now approximate the integral in various ways.

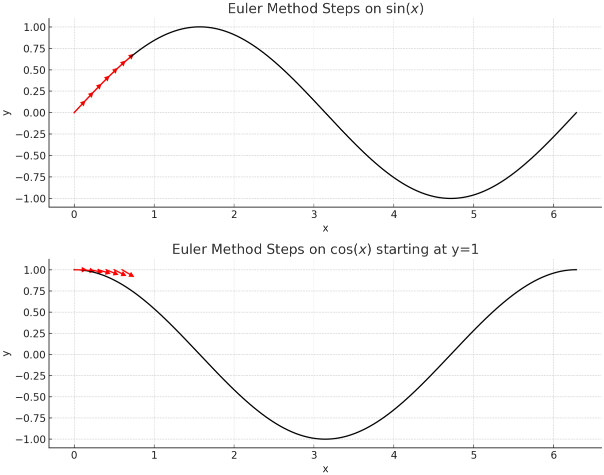
Explicit Euler Method

The explicit Euler method is the simplest method for numerically solving initial value problems. It uses a simple forward difference to approximate the next solution:

$$y_{n+1} = y_n + \Delta t \cdot f(t_n, y_n)$$

- **Intuition:** We take a small step Δt along the tangent to the solution curve.
- **Accuracy:** This method is only accurate for small Δt, as it does not account for the curvature of the solution curve.

Explicit Euler Method intuition



Problems with the Explicit Euler Method

As can be seen in the figure on the last slide the explicit Euler method can lead to large errors.

Given a start at an extreme point of the function, the values will stay put due to the tangent being parallel to the x-axis:

$$y_{n+1} = y_n + \Delta t \cdot f(t_n, y_n) = y_n$$

because the derivative $f(t_n, y_n) = 0$.

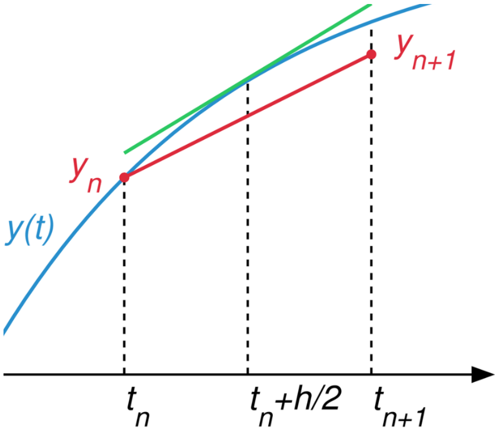
Midpoint Method

The midpoint method improves accuracy by using the derivative at the midpoint of the interval:

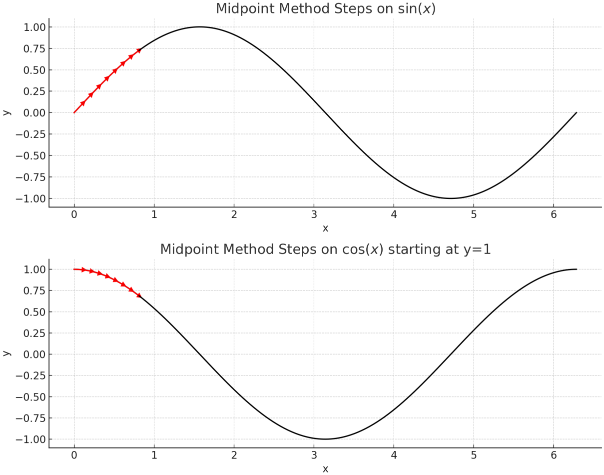
$$y_{n+1} = y_n + \Delta t \cdot f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2} \cdot f(t_n, y_n)\right)$$

- **Intuition:** We first compute a preliminary value at the midpoint of the interval and then use this value to calculate the next step.
- **Accuracy:** Higher than the explicit Euler method, as it better accounts for the curvature.

Midpoint Method intuition



Midpoint Method intuition



Improvement of the Midpoint Method over the Explicit Euler Method

Even if the derivative is zero: $f(t_n, y_n) = 0$, the midpoint method will still take a step.

$$
\begin{aligned}
y_{n+1} &= y_n + \Delta t \cdot f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2} \cdot f(t_n, y_n)\right) \\
&= y_n + \Delta t \cdot f\left(t_n + \frac{\Delta t}{2}, y_n\right) \\
&= y_n + \Delta t \cdot f\left(t_n + \frac{\Delta t}{2}, y_n\right) \neq y_n
\end{aligned}
$$

File I/O (BONUS)

File I/O – Basics

Python uses with statements to open and close files.

```
with open("slides/08/examples/testfile1.txt", "w") as f:
    f.write("Hello, world!\n")
    f.write("This is a test file.\n")

with open("slides/08/examples/testfile1.txt", "r") as f:
    for line in f:
        print(line, end="")
```

File I/O – file modes

The `open` function can be used to open files in different modes (multiple are possible):

File Mode	Description
`r`	Read mode
`w`	Write mode
`x`	Exclusive creation mode
`a`	Append mode
`b`	Binary mode
`t`	Text mode
`+`	Read and write mode

File I/O – Reading Files

To read the contents of a file, you can use the `read()` method of the file object. This method reads the entire contents of the file and returns it as a string.

```
with open('slides/08/examples/input.txt', 'r') as f:
    text = f.read()
    print(text)
```

In this example, we open the file `input.txt` in read mode (`r`), read its contents using the `read()` method, and print the contents to the console.

File I/O – Writing Files

To write to a file, you can use the write() method of the file object. This method writes the specified string to the file.

```
with open('slides/08/examples/input.txt', 'a') as f:
    f.write("This is a test file.\n")
    f.writelines(["Boring text 1.\n", "Boring text 2.\n"])
```

In this example, we open the file `example.txt` in write mode (`w`), write some strings to the file using the write() method.

File I/O – Reading multiple lines

To read multiple lines from a file, you can use the `readlines()` method of the file object. This method reads the entire contents of the file, and returns each line as an item in a list.

```
with open('slides/08/examples/input.txt', 'r') as f:
    for line in f.readlines():
        print(line, end="")
```

You do not need to close the file when using the `with` statement. It is automatically closed when the `with` block is exited.

File I/O – Binary files

To read or write binary files, you can use the `rb` and `wb` modes, respectively:

```
with open('data.bin', 'wb') as f:
    f.write(b'\x00\x01\x02\x03')

with open('data.bin', 'rb') as f:
    data = f.read()
    print(f"Original data: {data}")
```

But how do I know what is in there . . . ?

File I/O – Encoding

There are many encodings for text files.

- ASCII: 7-bit encoding, 128 characters
- Unicode: 16-bit encoding, 65536 characters
- UTF-8: 8-bit encoding, variable length, ASCII compatible
- ISO-8859-1: 8-bit encoding, 256 characters (Latin-1)

To interpret a file in the right way, you have to specify the encoding.

File I/O – Encoding

Specify the encoding when opening the file using the `encoding` argument:

```python
with open("slides/08/examples/testfile2.txt", "w", encoding="UTF-8") as f:
    f.write("Hello, Österreich!\n")

with open("slides/08/examples/testfile2.txt", "r", encoding="ASCII") as f:
    data = f.read()
    print(f"Original data: {data}")
```

This will fail! Why?

File I/O – Encoding

The specified encoding has to match the encoding of the file!

The file was written using `UTF-8` encoding and later read using `ASCII` encoding which is doomed to fail, since an ‚Ö'is in the textfile.

```
Traceback (most recent call last):
  File "/home/etschgi1/Desktop/REPOS/exercises-python/slides/08/examples/fileio5.py", line 5, in <modul
e>
    data = f.read()
  File "/usr/lib/python3.10/encodings/ascii.py", line 26, in decode
    return codecs.ascii_decode(input, self.errors)[0]
UnicodeDecodeError: 'ascii' codec can't decode byte 0xc3 in position 7: ordinal not in range(128)
```

ASCII Encoding (BONUS)

ASCII - Table

ASCII (American Standard Code for Information Interchange) is a character encoding standard that assigns unique numeric codes to each character in the English alphabet, as well as various punctuation marks and other symbols.

ASCII - Table

				b_7 b_6 b_5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Bits b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Column → Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	´	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	—	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

https://de.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange#/media/Datei:USASCII_code_chart.png

ASCII - Table

This way everything is up to interpretation: You can read the file as ASCII if there are no special characters in it.:

```
with open('slides/08/examples/ascii.txt', 'wb') as f:
    f.write(b'\x41\x42\x43')

with open('slides/08/examples/ascii.txt', 'r', encoding='ascii') as f:
    print(f.read()) # ABC
```

Using other encodings

With another encoding you can read special characters such as ‚Ö‘:

```
with open('slides/08/examples/ascii.txt', 'wb') as f:
    f.write(b'\x41\x42\x43\xD6')

with open('slides/08/examples/ascii.txt', 'r', encoding='ISO-8859-1') as f:
    print(f.read()) # ABCÖ
```

pickle (BONUS)

pickle

`pickle` is a module that allows you to store almost any Python object (lists, dictionaries, . . .) in a file.

`pickle` is a binary format, so you have to open the file in binary mode (`wb` or `rb`).

Use the `dump()` method to write an object to a file, and the `load()` method to read an object from a file.

pickle – Example

```python
import pickle

# Create a dictionary
dic_ = {'a': 1, 'b': 2, 'c': 3}
with open('slides/08/examples/dic_.pkl', 'wb') as f:
    pickle.dump(dic_, f)
with open('slides/08/examples/dic_.pkl', 'rb') as f:
    data = pickle.load(f)
    print(f"Original data: {data}")
```