Numerical Analysis

Assoc. Prof. Dr. Bora Canbula

https://github.com/canbula/NumericalAnalysis/

Variables

Variables are symbols for memory addresses.

Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

Built-in Functions

A abs() aiter() all() anext() any() ascii()	**E** enumerate() eval() exec() **F** filter()	**L** len() list() locals() **M** map()	**R** range() repr() reversed() round() **S**

hex(*x*)

Convert an integer number to a lowercase hexadecimal string prefixed with "0x". If *x* is not a Python `int` object, it has to define an `__index__()` method that returns an integer. Some examples:

```
>>> hex(255)
'0xff'
>>> hex(-42)
'-0x2a'
```

classmethod() compile() complex() **D**	help() $\boxed{\text{hex()}}$ **I** $\boxed{\text{id()}}$	ord() **P** pow() print()	type() **V** vars()

id(*object*)

Return the "identity" of an object. This is an integer which is guaranteed to be unique and constant for this object during its lifetime. Two objects with non-overlapping lifetimes may have the same `id()` value.

https://docs.python.org/3/library/functions.html

Identifier Names

For variables, functions, classes etc. we use identifier names. We must obey some rules and we should follow some naming conventions.

Rules

- Names are case sensitive.
- Names can be a combination of letters, digits, and underscore.
- Names can only start with a letter or underscore, can not start with a digit.
- Keywords can not be used as a name.

`keyword` — Testing for Python keywords

Source code: Lib/keyword.py

This module allows a Python program to determine if a string is a keyword or soft keyword.

`keyword.`**`iskeyword`**`(`*s*`)`

Return `True` if *s* is a Python keyword.

`keyword.`**`kwlist`**

Sequence containing all the keywords defined for the interpreter. If any keywords are defined to only be active when particular `__future__` statements are in effect, these will be included as well.

`keyword.`**`issoftkeyword`**`(`*s*`)`

Return `True` if *s* is a Python soft keyword.

New in version 3.9.

`keyword.`**`softkwlist`**

Sequence containing all the soft keywords defined for the interpreter. If any soft keywords are defined to only be active when particular `__future__` statements are in effect, these will be included as well.

New in version 3.9.

Identifier Names

For variables, functions, classes etc. we use identifier names. We must obey some rules and we should follow some naming conventions.

Rules

- Names are case sensitive.
- Names can be a combination of letters, digits, and underscore.
- Names can only start with a letter or underscore, can not start with a digit.
- Keywords can not be used as a name.

https://peps.python.org/

Python Enhancement Proposals Python » PEP Index » PEP 8

PEP 8 – Style Guide for Python Code

Author: Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Nick Coghlan <ncoghlan at gmail.com>

Status: Active

Type: Process

Created: 05-Jul-2001

Post-History: 05-Jul-2001, 01-Aug-2013

Identifier Names

For variables, functions, classes etc. we use identifier names. We <u>must</u> obey some <u>rules</u> and we <u>should</u> follow some naming <u>conventions</u>.

Conventions

- <u>Names to Avoid</u>
  Never use the characters ‘l’ (lowercase letter el), ‘O’ (uppercase letter oh), or ‘I’ (uppercase letter eye) as single character variable names.
- <u>Packages</u>
  Short, all-lowercase names without underscores
- <u>Modules</u>
  Short, all-lowercase names, can have underscores
- <u>Classes</u>
  CapWords (upper camel case) convention
- <u>Functions</u>
  snake_case convention
- <u>Variables</u>
  snake_case convention
- <u>Constants</u>
  ALL_UPPERCASE, words separated by underscores

Leading and Trailing Underscores

- `_single_leading_underscore`
  Weak “internal use” indicator.
  `from M import *` does not import objects whose names start with an underscore.
- `single_trailing_underscore_`
  Used by convention to avoid conflicts with keyword.
- `__double_leading_underscore`
  When naming a class attribute, invokes name mangling (inside class `FooBar`, `__boo` becomes `_FooBar__boo`)
- `__double_leading_and_trailing_underscore__`
  “magic” objects or attributes that live in user-controlled namespaces (`__init__`, `__import__`, etc.). Never invent such names; only use them as documented.

Variable Types

Python is dynamically typed. Python does not have primitive types. Everything is an object in Python, therefore, a variable is purely a reference to an object with the specified value.

Numeric Types

- Integer
- Float
- Complex
- Boolean

Formatted Output

- `print(“static text = ”, variable)`
- `print(“static text = %d” % (variable))`
- `print(“static text = {0}”.format(variable))`
- `print(f”static text = {variable}”)`
- `print(f”static text = {variable:5d}”)`

Variable Types

Python is dynamically typed. Python does not have primitive types. Everything is an object in Python, therefore, a variable is purely a reference to an object with the specified value.

Numeric Types

- Integer
- Float
- Complex
- Boolean

Sequences

- Strings
- List
- Tuple
- Set
- Dictionary

Week02/IntroductoryPythonDataStructures.pdf

Introductory Python : Data Structures in Python

Assoc. Prof. Dr. Bora Canbula
Manisa Celal Bayar University

Lists in Python:

Ordered and mutable sequence of values indexed by integers

Initializing

```
a_list = [] ## empty
a_list = list() ## empty
a_list = [3, 4, 5, 6, 7] ## filled
```

Finding the index of an item

```
a_list.index(5) ## 2 (the first occurence)
```

Accessing the items

```
a_list[0] ## 3
a_list[1] ## 4
a_list[-1] ## 7
a_list[-2] ## 6
a_list[2:] ## [5, 6, 7]
a_list[:2] ## [3, 4]
a_list[1:4] ## [4, 5, 6]
a_list[0:4:2] ## [3, 5]
a_list[4:1:-1] ## [7, 6, 5]
```

Adding a new item

```
a_list.append(9) ## [3, 4, 5, 6, 7, 9]
a_list.insert(2, 8) ## [3, 4, 8, 5, 6, 7, 9]
```

Update an item

```
a_list[2] = 1 ## [3, 4, 1, 5, 6, 7, 9]
```

Remove the list or just an item

```
a_list.pop() ## last item
a_list.pop(2) ## with index
del a_list[2] ## with index
a_list.remove(5) ## first occurence of 5
a_list.clear() ## returns an empty list
del a_list ## removes the list completely
```

Extend a list with another list

```
list_1 = [4, 2]
list_2 = [1, 3]
list_1.extend(list_2) ## [4, 2, 1, 3]
```

Reversing and sorting

```
list_1.reverse() ## [3, 1, 2, 4]
list_1.sort() ## [1, 2, 3, 4]
```

Counting the items

```
list_1.count(4) ## 1
list_1.count(5) ## 0
```

Copying a list

```
list_1 = [3, 4, 5, 6, 7]
list_2 = list_1
list_3 = list_1.copy()
list_1.append(1)
list_2 ## [3, 4, 5, 6, 7, 1]
list_3 ## [3, 4, 5, 6, 7]
```

Sets in Python:

Unordered and mutable collection of values with no duplicate elements. They support mathematical operations like union, intersection, difference and symmetric difference

Initializing

```
a_set = set() ## empty
a_set = {3, 4, 5, 6, 7} ## filled
```

No duplicate values

```
a_set = {3, 3, 3, 4, 4} ## {3, 4}
```

Adding and updating the items

```
a_set.add(5) ## {3, 4, 5}
set_1 = {1, 3, 5}
set_2 = {5, 7, 9}
set_1.update(set_2) ## {1, 3, 5, 7, 9}
```

Removing the items

```
a_set.pop() ## removes an item and returns it
a_set.remove(3) ## removes the item
a_set.discard(3) ## removes the item
```

If item does not exist in set,
remove() raises an error, discard() does not

```
a_set.clear() ## returns an empty set
del a_set ## removes the set completely
```

Mathematical operations

```
set_1 = {1, 2, 3, 5}
set_2 = {1, 2, 4, 6}
```

Union of two sets

```
set_1.union(set_2) ## {1, 2, 3, 4, 5, 6}
set_1 | set_2 ## {1, 2, 3, 4, 5, 6}
```

Intersection of two sets

```
set_1.intersection(set_2) ## {1, 2}
set_1 & set_2 ## {1, 2}
```

Difference between two sets

```
set_1.difference(set_2) ## {3, 5}
set_2.difference(set_1) ## {4, 6}
set_1 - set_2 ## {3, 5}
set_2 - set_1 ## {4, 6}
```

Symmetric difference between two sets

```
set_1.symmetric_difference(set_2) ## {3,4,5,6}
set_1 ^ set_2 ## {3, 4, 5, 6}
```

Update sets with mathematical operations

```
set_1.intersection_update(set_2) ## {1, 2}
set_1.difference_update(set_2) ## {3, 5}
set_1.symmetric_difference_update(set_2)
## {3, 4, 5, 6}
```

Copying a set

Same as lists

Dictionaries in Python:

Unordered and mutable set of key-value pairs

Initializing

```
a_dict = {} ## empty
a_dict = dict() ## empty
a_dict = {"name":"Bora"} ## filled
```

Accessing the items

```
a_dict["name"] ## "Bora"
a_dict.get("name") ## "Bora"
```

If the key does not exist in dictionary,
index notation raises an error, get() method does not

Accessing the items with views

```
other_dict = {"a": 3, "b": 5, "c": 7}
other_dict.keys() ## ['a', 'b', 'c']
other_dict.values() ## [3, 5, 7]
other_dict.items()
## [('a', 3), ('b', 5), ('c', 7)]
```

Adding a new item

```
a_dict["city"] = "Manisa"
a_dict["age"] = 37
## {"name":"Bora", "city":"Manisa", "age":37}
```

Update an item

```
a_dict["age"] = 38
## {"name":"Bora", "city":"Manisa", "age":38}
other_dict = {"age":39}
a_dict.update(other_dict)
## {"name":"Bora", "city":"Manisa", "age":39}
```

Removing the items

```
a_dict.popitem() ## last inserted item
a_dict.pop("city") ## with a key
a_dict.clear() ## returns an empty dictionary
del a_dict ## removes the dict completely
```

Initialize a dictionary with fromkeys

```
a_list = ['a', 'b', 'c']
a_dict = dict.fromkeys(a_list)
## {'a': None, 'b': None, 'c': None}
a_dict = dict.fromkeys(a_list, 0)
## {'a': 0, 'b': 0, 'c': 0}
a_tuple = (3, 'name', 7)
a_dict = dict.fromkeys(a_tuple, True)
## {3: True, 'name': True, 7: True}
a_set = {0, 1, 2}
a_dict = dict.fromkeys(a_set, False)
## {0: False, 1: False, 2: False}
```

Tuples in Python:

Ordered and immutable sequence of values indexed by integers

Initializing

```
a_tuple = () ## empty
a_tuple = tuple() ## empty
a_tuple = (3, 4, 5, 6, 7) ## filled
```

Finding the index of an item

```
a_tuple.index(5) ## 2 (the first occurence)
```

Accessing the items

Same index and slicing notation as lists

Adding, updating, and removing the items

Not allowed because tuples are immutable

Sorting

Tuples have no sort() method since they are immutable

```
sorted(a_tuple) ## returns a sorted list
```

Counting the items

```
a_tuple.count(7) ## 1
a_tuple.count(9) ## 0
```

Some Iteration Examples:

```
a_list = [3, 5, 7]
a_tuple = (4, 6, 8)
a_set = {1, 4, 7}
a_dict = {"a":1, "b":2, "c":3}
```

For ordered sequences

```
for i in range(len(a_list)):
    print(a_list[i])
for i, x in enumerate(a_tuple):
    print(i, x)
```

For ordered or unordered sequences

```
for a in a_set:
    print(a)
```

Only for dictionaries

```
for k in a_dict.keys():
    print(k)
for v in a_dict.values():
    print(v)
for k,v in zip(a_dict.keys(),a_dict.values()):
    print(k, v)
for k, v in a_dict.items():
    print(k, v)
```

Problem Set

1. What is the correct writing of the programming language that we used in this course?
( ) Phyton
( ) Pyhton
( ) Pthyon
( ) Python

2. What is the output of the code below?
```
my_name = "Bora Canbula"
print(my_name[2::-1])
```
( ) alu
( ) ula
( ) roB
( ) Bor

3. Which one is not a valid variable name?
( ) for_
( ) Manisa_Celal_Bayar_University
( ) IF
( ) not

4. What is the output of the code below?
```
for i in range(1, 5):
    print(f"{i:2d}{(i/2):4.2f}", end='')
```
( ) 010.50021.00031.50042.00
( ) 10.50 21.00 31.50 42.00
( ) 1 0.5 2 1.0 3 1.5 4 2.0
( ) 100.5 201.0 301.5 402.0

5. Which one is the correct way to print Bora's age?
```
profs = [
    {"name": "Yener", "age": 25},
    {"name": "Bora", "age": 37},
    {"name": "Ali", "age": 42}
]
```
( ) profs["Bora"]["age"]
( ) profs[1][1]
( ) profs[1]["age"]
( ) profs.age[name="Bora"]

6. What is the output of the code below?
```
x = set([int(i/2) for i in range(8)])
print(x)
```
( ) {0, 1, 2, 3, 4, 5, 6, 7}
( ) {0, 1, 2, 3}
( ) {0, 0, 1, 1, 2, 2, 3, 3}
( ) {0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4}

7. What is the output of the code below?
```
x = set(i for i in range(0, 4, 2))
y = set(i for i in range(1, 5, 2))
print(x^y)
```
( ) {0, 1, 2, 3}
( ) {}
( ) {0, 8}
( ) SyntaxError: invalid syntax

8. Which of the following sequences is immutable?
( ) List
( ) Set
( ) Dictionary
( ) String

9. What is the output of the code below?
```
print(int(2_999_999.999))
```
( ) 2
( ) 3000000
( ) ValueError: invalid literal
( ) 2999999

10. What is the output of the code below?
```
x = (1, 5, 1)
print(x, type(x))
```
( ) [1, 2, 3, 4] <class 'list'>
( ) (1, 5, 1) <class 'range'>
( ) (1, 5, 1) <class 'tuple'>
( ) (1, 2, 3, 4) <class 'set'>

Iterables - Sequences - Iterators

An **iterable** is any object that can be looped over. It represents a collection of elements that can be accessed one by one.

An object is considered iterable if:

- It implements the `__iter__()` method which returns an iterator, or
- It defines the `__getitem__()` method that can fetch items using integer indices starting from zero.

A **sequence** is a subtype of iterables. It's an ordered collection of elements that can be indexed by numbers.

- **Ordered:** Elements in a sequence have a specific order.
- **Indexable:** You can get any item using an index `my_sequence[5]`.
- **Slicable:** Supports slicing to get some of items `my_sequence[2:5]`.

An **iterator** is an object that produces items (one at a time) from its associated iterable.

- **Stateful:** An iterator remembers its state between calls. Once an element is consumed, it can't be accessed again without reinitializing the iterator.
- **Lazy Evaluation:** Items are not produced from the source iterable until the iterator's `__next__()` method is called.
- Iterators raise a StopIteration exception when there are no more items to return.
- An iterator's `__iter__()` method returns the iterator object itself.
- While all iterables must be able to produce an iterator (with `__iter__()` method), not all iterators are directly iterable without using a loop.

Numpy Arrays

Numerical Python (**NumPy**)is a powerful library for numerical computing. Its key feature is multi dimensional arrays (**ndarrays**).

Traditional Python Lists

- **Dynamically Typed:** Lists can store elements of mixed types in a single list.
- **Resizable:** Lists can be resized by appending or removing elements.
- **General-purpose:** Lists are general-purpose containers for items of any type.
- **Memory:** Lists have a larger memory overhead because of their general-purpose nature and dynamic typing.
- **Performance:** Basic operations on lists may not be as fast as those on NumPy arrays because they aren't optimized for numerical operations.

NumPy Arrays

- **Typed:** All elements in a NumPy array are of the same type.
- **Size:** The size of a NumPy array is fixed upon creation. However, one can create a new array with a different size, but resizing in-place (like appending in lists) isn't directly supported.
- **Efficiency:** NumPy arrays are memory-efficient as they store elements in contiguous blocks of memory.
- **Performance:** Operations on NumPy arrays are typically faster than lists, especially for numerical tasks, due to optimized C and Fortran extensions.
- **Vectorized Operations:** Supports operations that apply to the entire array without the need for explicit loops (e.g., adding two arrays element-wise).
- **Broadcasting:** Advanced feature allowing operations on arrays of different shapes.
- **Extensive Functionality:** Beyond just array storage, NumPy provides a vast range of mathematical, logical, shape manipulation, and other operations.
- **Interoperability:** Can interface with C, C++, and Fortran code.

Homework

Week04/arrays_firstname_lastname.py

Submit your work to GitHub

Function Description

replace_center_with_minus_one(d, n, m)

This function creates an `n-by-n` numpy array populated with random integers that have up to `d` digits. It then replaces the central `m-by-m` part of this array with `-1`.

Parameters

- **d**: Number of digits for the random integers.
- **n**: Size of the main array.
- **m**: Size of the central array that will be replaced with `-1`.

Returns

- A modified numpy array with its center replaced with `-1`.

Exceptions

- **ValueError**: This exception is raised in the following scenarios:
  - If `m > n`
  - If `d <= 0`
  - If `n < 0`
  - If `m < 0`

Problem Set

1. What is the correct way to create a NumPy array?
( ) np.list([1, 2, 3])
( ) np([1, 2, 3])
( ) np.array([1, 2, 3])
( ) np(array([1, 2, 3]))

2. Which of the following arrays is a 2-D array?
( ) [3, 5]
( ) [[3], [5]]
( ) [{1, 3}, {5, 7}]
( ) [2]

3. What is the correct way to print 5 from the array given below?
a = np.array([[1, 2], [3, 4], [5, 6]])
( ) print(a[3, 1])
( ) print(a[2, 0])
( ) print(a[1, 2])
( ) print(a[1, 3])

4. What is the correct way to print every other item from the array given below?
a = np.arange(5)
( ) print(a[1:3:5])
( ) print(a[::2])
( ) print(a[1:5])
( ) print(a[0:2:4]

5. What does the shape mean of a NumPy array?
( ) Number of columns
( ) Total number of items
( ) Number of items in each dimension
( ) Number of rows

6. What is the output of the code below?
```
n_1 = np.array([1, 2, 3])
n_2 = np.array([4, 5, 6])
n_3 = np.array([7, 8, 9])
print(np.array([n_1, n_2, n_3]).ndim)
```

Your answer:

7. What is the output of the code below?
```
n_1 = np.array([1, 2, 3])
n_2 = np.array([4, 5, 6])
n_3 = np.array([7, 8, 9])
print(np.array([n_1 + n_2 + n_3]).shape)
```

Your answer:

8. Which of the following is created with the code given below?
np.array([[1, 2, 3], [4, 5, 6]])
( ) 1-d array of shape 6 x 1
( ) 2-d array of shape 2 x 3
( ) 3-d array of shape 3 x 2
( ) 3-d array of shape 2 x 3

9. What is the output of the code below?
```
print(np.arange(10).reshape(2, -1))
```

10. What is the output of the code below?
```
Print(np.array([0.5, 1.5, 2.5]).dtype)
```

Binary Representation of Floating-Point Numbers

$$375.814$$

$$10^2 \quad 10^1 \quad 10^0 \qquad 10^{-1} \quad 10^{-2} \quad 10^{-3}$$

$$375.814 = (3 \cdot 10^2) + (7 \cdot 10^1) + (5 \cdot 10^0) \\ +(8 \cdot 10^{-1}) + (1 \cdot 10^{-2}) + (4 \cdot 10^{-3})$$

Integer Part

$$\begin{aligned} 375 \,\%\, 2 &= 1 \\ 187 \,\%\, 2 &= 1 \\ 93 \,\%\, 2 &= 1 \\ 46 \,\%\, 2 &= 0 \\ 23 \,\%\, 2 &= 1 \\ 11 \,\%\, 2 &= 1 \\ 5 \,\%\, 2 &= 1 \\ 2 \,\%\, 2 &= 0 \\ & 1 \end{aligned}$$

Decimal Part

$$\begin{aligned} 0.814 \cdot 2 &= 1.628 \rightarrow 1 \\ 0.628 \cdot 2 &= 1.256 \rightarrow 1 \\ 0.256 \cdot 2 &= 0.512 \rightarrow 0 \\ 0.512 \cdot 2 &= 1.024 \rightarrow 1 \\ 0.024 \cdot 2 &= 0.048 \rightarrow 0 \\ 0.048 \cdot 2 &= 0.096 \rightarrow 0 \\ 0.096 \cdot 2 &= 0.192 \rightarrow 0 \\ 0.192 \cdot 2 &= 0.384 \rightarrow 0 \\ 0.384 \cdot 2 &= 0.768 \rightarrow 0 \\ 0.768 \cdot 2 &= 1.536 \rightarrow 1 \\ 0.036 \cdot 2 &= 0.072 \rightarrow 0 \\ & \vdots \end{aligned}$$

$$(375.814)_{10} = (101110111.1101000010\ldots)_2$$

Error

Homework

Week05/bra_firstname_lastname.py

Submit your work to GitHub

Binary Representation API

Your task is to create a Python application that exposes an API endpoint to convert floating-point numbers into their binary representation. This application will be a Flask web service that can accept GET requests with a floating-point number and respond with its binary representation. The binary representation should separate the integer and fractional parts with a dot.

Objectives

- Implement a BinaryRepresentation class that can take a float and provide methods to convert both the integer and decimal parts into binary.
- Develop a Flask API with a route that accepts a floating-point number as a query parameter and returns its binary representation.
- Ensure that your code passes a series of tests that will be run against it.
- All the details that you can't understand from this assignment should be extracted from the tests: **Week05/test_bra.py**

Requirements

1. **BinaryRepresentation Class**

- The class should be initialized with one float parameter.
- Implement a method **integer2binary** that converts the integer part of the float to binary.
- Implement a method **decimal2binary** that converts the decimal part of the float to binary up to 10 places.
- Implement the **__str__** method to return the binary representation as a string, separating the integer and decimal parts with a dot.

2. **Flask API**

- Set up a Flask application.
- Define a route that listens to **GET** requests and expects a number query parameter.
- Validate that the number parameter is a **float** and return an appropriate response if not.
- Utilize the **BinaryRepresentation** class to return the binary representation of the number.

3. **Error Handling**

- Ensure that the application correctly handles cases where the number is not provided, or is not a valid float.

Problem Set

1. In binary system, which of the following digits are used to represent a number?

( ) 1 and 2
( ) 0 and 1
( ) 0, 1 and 2
( ) A and B

2. Which of the following codes gives a binary representation of 97?

( ) binary(97)
( ) (97).binary()
( ) f"{97:b}"
( ) to_binary(97)

3. What is the name of the NumPy method which converts a number to binary system?

( ) np.binary()
( ) np.bin()
( ) np.binary_representation()
( ) np.binary_repr()

4. The code given below produces this output:

```
> 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 = 97
```

Complete the code with appropriate statements for the lines given with (1) and (2).

```
n = 16;r = 97;r_0 = r;b = [0]*n
for i in range(n-1, -1, -1):
    x = 2**i
    if r >= x:
        (1)
        (2)
b = b[::-1]
print(*b, end='')
print(f" = {r_0}")
```

5. Modify the code given in question **4** to avoid fixing the number of digits (**n**). Hint: use bit_length() method of integer object.

6. Use the codes given in the question **4** as a starting point and write Python codes which converts the decimal of a base-10 number into binary system.

7. Try to write a general function which converts a base-10 floating point number into any base including the decimal part.

```
def to_any_base(r: float, b: int) -> str:
    '''This function returns the base-b '''
    '''conversion of r, which is a      '''
    '''floating-point number.           '''
    '''Example:                         '''
    '''  to_any_base(3.5, 2) -> '11.1'  '''
```

IEEE 754 Representation of Floating-Point Numbers

13.375

$$13 \,\%\, 2 = 1$$
$$6 \,\%\, 2 = 0$$
$$3 \,\%\, 2 = 1$$
$$1$$

$$0.375 \cdot 2 = 0.750 \rightarrow 0$$
$$0.750 \cdot 2 = 1.500 \rightarrow 1$$
$$0.500 \cdot 2 = 1.000 \rightarrow 1$$

$$(13.375)_{10} = (1101.011)_2$$

$$(1101.011)_2 = 1.\underline{101011} \times 2^3$$

$$3 + 15 = 18$$

$$18 \,\%\, 2 = 0$$
$$9 \,\%\, 2 = 1$$
$$4 \,\%\, 2 = 0$$
$$2 \,\%\, 2 = 0$$
$$1$$

16-bit Half Precision

0	1	0	0	1	0	1	0	1	0	1	1	0	0	0	0

Sign (0: +, 1: -)

Exponent (5 bit)

Mantissa (10 bit)

0	0	0	0	0	**Denormal**

1	2	3	4	5	6	7	8	9	10	11	12	13	14		
−14	−13	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1		
15															
0															
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

$\boxed{15} \rightarrow$ **Bias**

30 values for exponents from min to max

1	1	1	1	1	**Inf**

Homework

Week06/halfprecision_firstname_lastname.py

Submit your work to GitHub

Implementing a Half-Precision Floating Point Converter

In computing, half-precision floating-point is a binary floating-point computer number format that occupies 16 bits (two bytes in modern computers) in computer memory. They are designed for use in situations where a wide dynamic range is not required, and where storage space is at a premium.

Your task is to implement a Python class named **HalfPrecision** that converts a Python floating-point number (**float**) to its binary **string** representation in half-precision floating-point format according to IEEE 754 standards.

All the details that you can't understand from this assignment should be extracted from the tests: **Week06/test_halfprecision.py**

Requirements

- Implement the class **HalfPrecision** with an **__init__** method and a **__str__** method.
- The **__init__** method should take a single argument: the number to be converted and also raises a **TypeError** if the number is **not a float**.
- The __str__ method should return a 16-character string representing the binary format as:
    - The first bit is the sign bit.
    - The following five bits represent the exponent.
    - The last ten bits represent the mantissa.
- Your class should correctly handle special numbers such as NaN (Not a Number), infinities, and denormal numbers. Only infinities will be tested at this stage, other special numbers will be studied next week in the classroom.

Problem Set

1. Find the smallest and the largest value that you can represent with 16-bit IEEE 754 standard?

2. Find the 16 bit IEEE 754 representation of -5.875.

3. Calculate the error if we use 16 bit IEEE 754 representation to store the value 0.17 in memory.

4. Use a custom IEEE 754 representation as 1-bit for the sign of the number and (4-bit exponent) + (20-bit mantissa). Convert 0.17 into this representation and compare the result with the previous question.

5. Calculate the bias for the 8 bit exponent part.

Precisions for IEEE 754 Representation

0	1	0	0	1	0	1	0	1	0	1	1	0	0	0	0

Sign (0: +, 1: -) **Exponent** **Mantissa**

IEEE 754 Precisions

Precision	Sign	Exponent	Mantissa	Total
Half	1	5	10	16
Single	1	8	23	32
Double	1	11	52	64
Quadruple	1	15	112	128
Octuple	1	19	236	256

Alternative method to convert floats to binary?

$$13.375$$

$13.375 \cdot 2 = 26.75$

$26.750 \cdot 2 = 53.50$

$53.500 \cdot 2 = 107.0$

Number times 2
To get an integer

$$(13.375)_{10} \cdot 2^3 = 107$$

$107 \,\%\, 2 = 1$

$53 \,\%\, 2 = 1$

$26 \,\%\, 2 = 0$

$13 \,\%\, 2 = 1$

$6 \,\%\, 2 = 0$

$3 \,\%\, 2 = 1$

1

Legacy Way

$$13.375$$

$13 \,\%\, 2 = 1$ | $0.375 \cdot 2 = 0.750 \rightarrow 0$

$6 \,\%\, 2 = 0$ | $0.750 \cdot 2 = 1.500 \rightarrow 1$

$3 \,\%\, 2 = 1$ | $0.500 \cdot 2 = 1.000 \rightarrow 1$

1

$$(13.375)_{10} = (1101.011)_2$$

$$(107)_{10} = (13.375)_{10} \cdot 2^3 = (1101011)_2$$

$$(13.375)_{10} = (1101011)_2 \cdot 2^{-3}$$

$$(13.375)_{10} = (1101.011)_2$$

The power must be added to exponent

Precisions for IEEE 754 Representation

13.375

$$13.375 \cdot 2 = 26.75$$
$$26.750 \cdot 2 = 53.50$$
$$53.500 \cdot 2 = 107.0$$

Number times 2 To get an integer

$$(13.375)_{10} \cdot 2^3 = 107$$

$$107 \% 2 = 1$$
$$53 \% 2 = 1$$
$$26 \% 2 = 0$$
$$13 \% 2 = 1$$
$$6 \% 2 = 0$$
$$3 \% 2 = 1$$
$$1$$

$$(107)_{10} = (13.375)_{10} \cdot 2^3 = (1101011)_2$$

$$(13.375)_{10} = (1101011)_2 \cdot 2^{-3}$$

$$(1.101011)_2 \cdot 2^6 \cdot 2^{-3}$$

$$(1.\underline{101011})_2 \cdot 2^3$$

Exponent

Bias must be added!

Mantissa

- ✓ Define the precisions
- ✓ Enable the custom precision
- ✓ Calculate the bias
- ✓ Find the power of 2 to scale up the number to an integer
- ✓ Use the bias and scale to calculate the exponent
- ✓ Convert exponent to binary
- ✓ Fill the mantissa with trailing zeros
- ✓ Find the sign of the number
- ✓ Return the result as a string with __str__ method

- ✓ Input validation
- ✓ Edge cases such as Inf, NaN, and signed zero
- ✓ Find the normalization range, raise error for denormals

Convert the IEEE 754 Representation to Float?

Problem Set

1. The numbers used in the following equation are given in Half Precision IEEE 754 format, but in hexadecimal notation. Please find the result as a base-10 number.

67C8 + 3C00 =

2. Suppose that we want to save the value 0.1 in our PC. How many bits do we need for the mantissa part?

3. Which one is the correct representation of zero in IEEE 754 half precision?

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

or

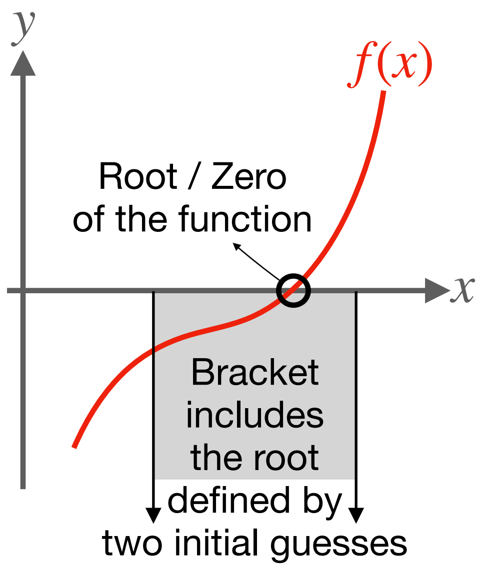
0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0

Explain your answer.

4. You can find the current version of the file ieee754.py in the folder Week06 of GitHub repo of this course. List the weak points of this code that must be fixed.

5. Using NumPy arrays to save the zeros and ones in ieee754.py, was it a correct choice or not? Explain your answer.

Roots of Equations



Bracketing Methods

- ➡ Bisection Method
- ➡ False-Position Method (Regula-Falsi Method)

❌ Two initial guesses
✔ Slow guaranteed convergence

Open Methods

- ➡ Fixed-Point Iteration
- ➡ Newton-Raphson Method
- ➡ Secant Method

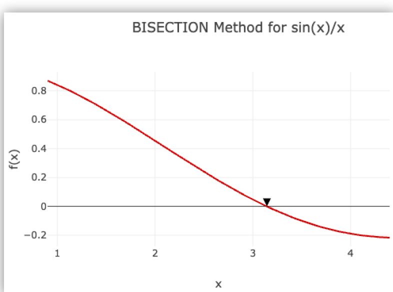
✔ One initial guess
✔ Converge faster if it does

Example Function

$$f(x) = \frac{\sin(x)}{x}$$

$$a = 1 \,, b = 4$$

$$\varepsilon = 1.E-03$$



Can we write a single class to use for both methods?

Bisection Method

Based on the Intermediate Value Theorem, which states that if the function is continuous on an interval $[a, b]$ and changes sign over that interval, then there exists at least one root in the interval.

Inputs	**Condition**
$f(x), a, b, \varepsilon$	$f(a) \cdot f(b) < 0$

Iterations

$$c = \frac{a+b}{2}$$

$$\Downarrow$$

$$f(a) \cdot f(c) < 0 \quad \Rightarrow \quad b = c$$

$$f(b) \cdot f(c) < 0 \quad \Rightarrow \quad a = c$$

$$\Downarrow$$

$$f(c) \leq \varepsilon \;\Rightarrow\; \text{Stop}$$

$$|b-a| \leq \varepsilon \;\Rightarrow\; \text{Stop}$$

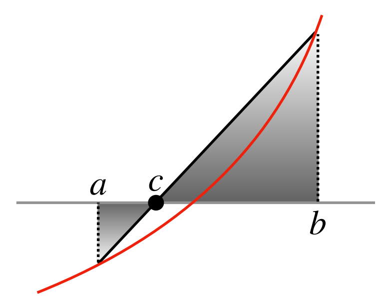
Number of Iterations

$$n \geq \log_2 \frac{b-a}{\varepsilon}$$

False-Position Method

Based on using the values of the function at the endpoints of the interval to draw a straight line and taking the x-intercept of this line as the new root. It converges to the root more quickly, if the function is linear or nearly linear.

Inputs	**Condition**
$f(x), a, b, \varepsilon$	$f(a) \cdot f(b) < 0$



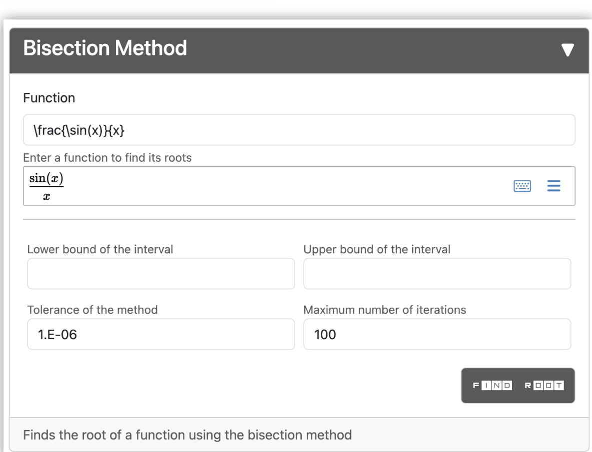
Iterations

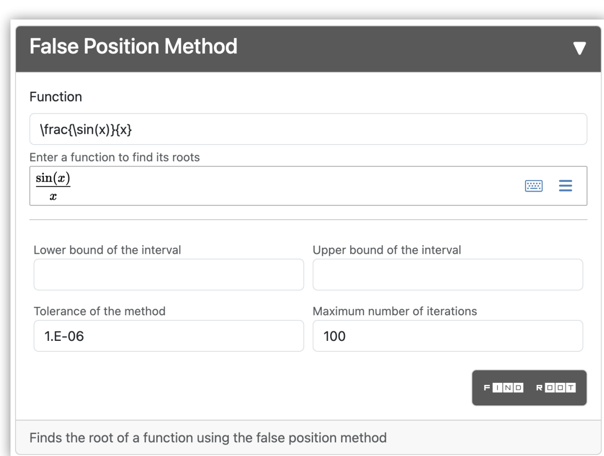
$$c = b - f(b) \cdot \frac{b-a}{f(b)-f(a)}$$

$$\Downarrow$$

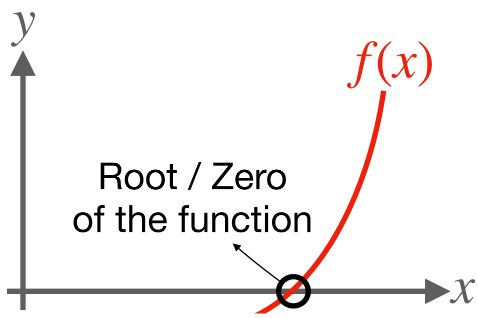
$$f(c) \leq \varepsilon \;\Rightarrow\; \text{Stop}$$

$$|b-a| \leq \varepsilon \;\Rightarrow\; \text{Stop}$$





Roots of Equations



Bracketing Methods	Open Methods
➡ Bisection Method ➡ False-Position Method (Regula-Falsi Method)	➡ Fixed-Point Iteration ➡ Newton-Raphson Method ➡ Secant Method
✗ Two initial guesses ✓ Slow guaranteed convergence	✓ One initial guess ✓ Converge faster if it does

Fixed-Point Method

Iterative process for finding an approximation to a root of a function by repeatedly applying a transformation of that function, starting from an initial guess, until the process converges to a fixed point.

Inputs

$f(x)\,,x_0\,,\varepsilon$

Transformation

$f(x)=0\,,x=g(x)$

Iterations

$$x_{n+1}=g(x_n)$$

$$\Downarrow$$

$$f(x_n)\le\varepsilon \;\Rightarrow\; \text{Stop}$$

$$|x_n-x_{n-1}|\le\varepsilon \;\Rightarrow\; \text{Stop}$$

Homework

Visual Representation

Automatic Transformation

API + GUI

10% of your Grade

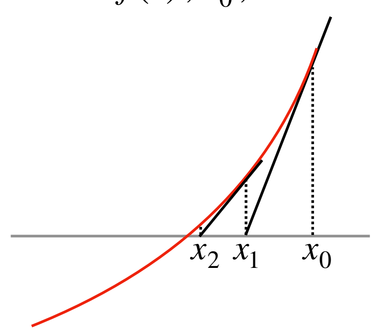
Group up to 4

Newton-Raphson Method

Used to find the roots of a real valued function. Very popular due to its quadratic convergence speed. Based on the idea that a continuous and differentiable function can be approximated by its tangent line.

Inputs

$f(x)\,,x_0\,,\varepsilon$



Iterations

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}$$

$$\Downarrow$$

$$f(x_n)\le\varepsilon \;\Rightarrow\; \text{Stop}$$

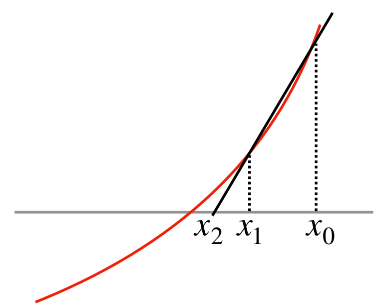
$$|x_n-x_{n-1}|\le\varepsilon \;\Rightarrow\; \text{Stop}$$

Secant Method

Similar to NR, but instead of using the derivative of the function, approximates it using two nearby points. This makes the Secant method useful when the derivative of the function is difficult to compute or unknown.

Inputs

$f(x)\,,x_0\,,x_1\,,\varepsilon$



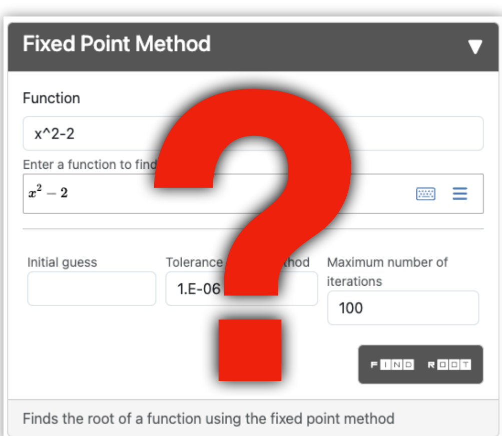
Iterations

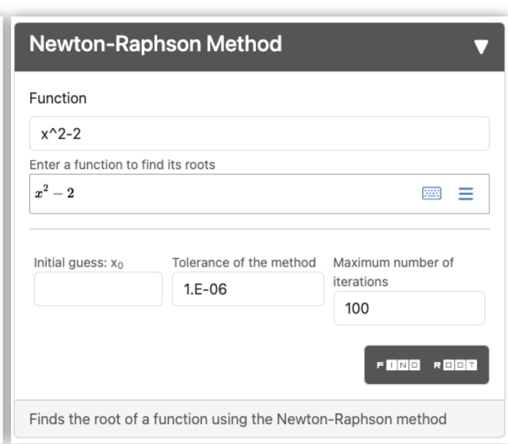
$$x_{n+1}=x_n-f(x_n)\cdot\frac{x_n-x_{n-1}}{f(x_n)-f(x_{n-1})}$$

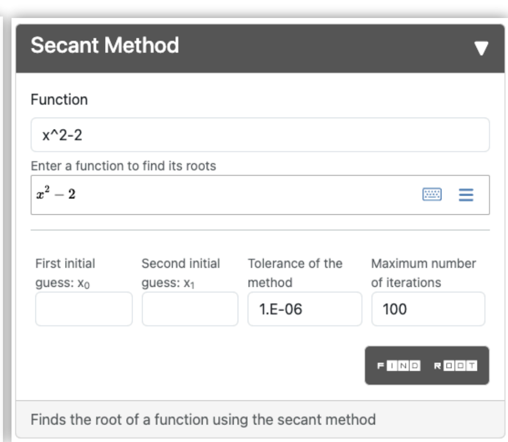
$$\Downarrow$$

$$f(x_n)\le\varepsilon \;\Rightarrow\; \text{Stop}$$

$$|b-a|\le\varepsilon \;\Rightarrow\; \text{Stop}$$







Problem Set

1. Fill the table with the intervals for bisection method for each iteration to find the root of the function given below:

$$f(x) = x^3 - 3x - 5$$

Intervals for Bisection Method		
Iteration	a	b
1	1	2

2. The function given below has multiple roots in interval [-5, 5]. Combine bisection method with a random number generator to improve the code given below to be able to find and return multiple roots.

$$f(x) = x^3 - 9x + 3$$

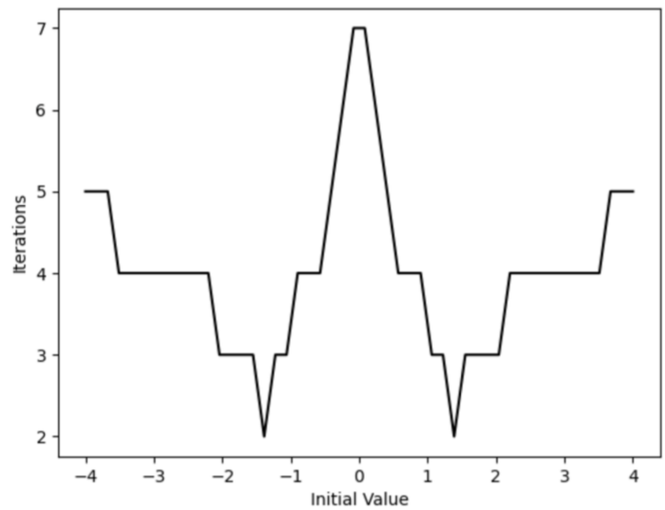
```python
import numpy as np

def bisection(func, a, b, tolerance=1e-5):
    fa = func(a)
    fb = func(b)
    if np.sign(fa) == np.sign(fb):
        raise ValueError("f(a) and f(b) must "\
                         "have opposite signs")
    m = (a + b)/2
    fm = func(m)
    if np.abs(func(m)) < tolerance:
        return m
    if np.sign(fa) == np.sign(fm):
        return bisection(func, m, b, tolerance)
    if np.sign(fb) == np.sign(fm):
        return bisection(func, a, m, tolerance)
```

3. Find the relationship between the initial value and the number of iterations which is needed to find the root of a function by using Newton-Raphson method.
Example:

$$f(x) = x^2 - 2$$

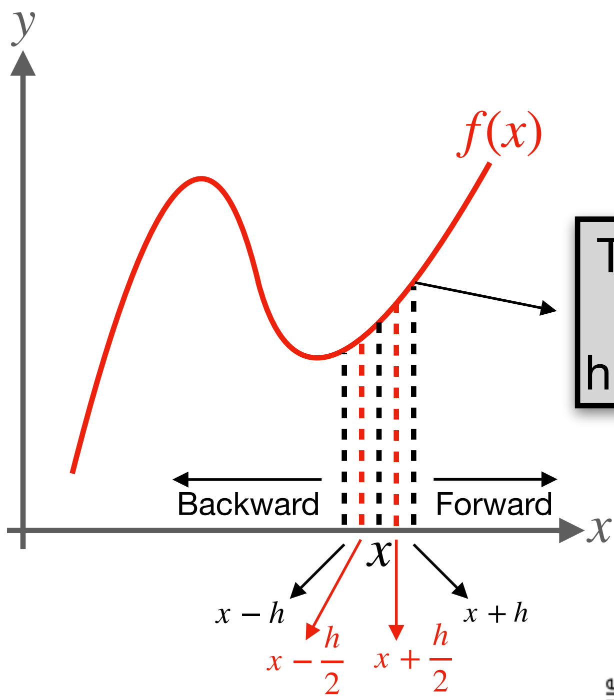
Tolerance is 10^{-5}.



4. Use the last two digits of your student id to build the number AB. Use this number in the function given below and find the root by using Newton-Raphson method:

$$f(x) = x - \sqrt[3]{BA}$$

Finite Difference Approach for Numerical Derivatives



First Derivative of the Function

$$f(x+h) = f(x) + h\boxed{f'(x)} + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \dots$$

Taylor Expansion of this point if h is small enough.

How significant are these terms?

They are not!

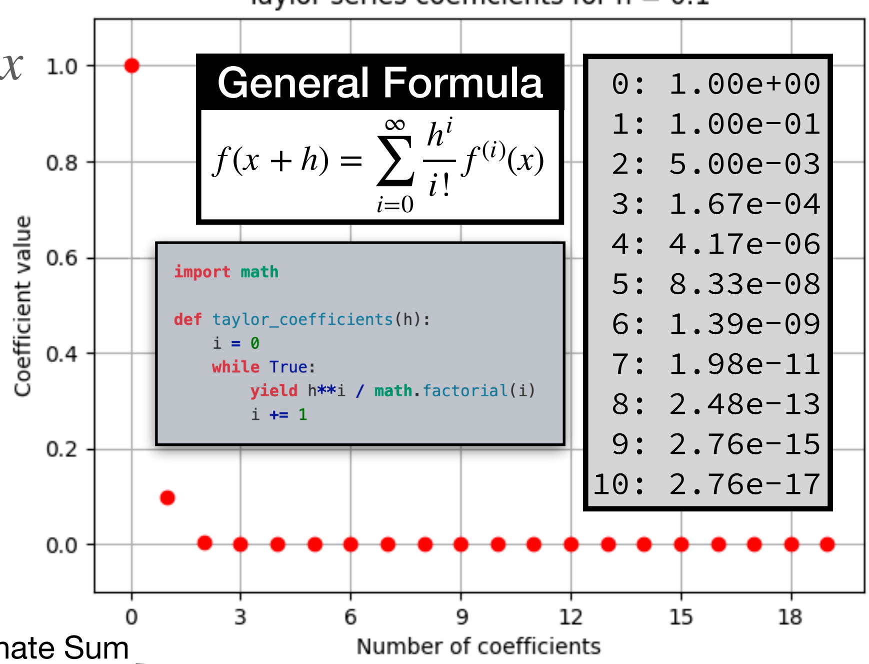
$$f(x+h) = f(x) + hf'(x) + \dots$$

$$f(x+h) = f(x) + hf'(x) + \epsilon$$

First Forward Diff. Approximation

$$f'(x) = \frac{f(x+h) - f(x)}{h}$$

Taylor series coefficients for h = 0.1



General Formula

$$f(x+h) = \sum_{i=0}^{\infty} \frac{h^i}{i!} f^{(i)}(x)$$

```python
import math

def taylor_coefficients(h):
    i = 0
    while True:
        yield h**i / math.factorial(i)
        i += 1
```

```
0:  1.00e+00
1:  1.00e-01
2:  5.00e-03
3:  1.67e-04
4:  4.17e-06
5:  8.33e-08
6:  1.39e-09
7:  1.98e-11
8:  2.48e-13
9:  2.76e-15
10: 2.76e-17
```

Going backward

Alternate Sum

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + \dots$$

First Backward Diff. Approximation

$$f'(x) = \frac{f(x) - f(x-h)}{h}$$

$$f(x) = \sin(x)$$
$$f'(x) = \cos(x)$$

Absolute Error

h	Forward	Backward	Central
1.00e-01	4.41e-02	4.24e-02	2.08e-04
1.00e-02	4.34e-03	4.32e-03	2.08e-06
1.00e-03	4.33e-04	4.33e-04	2.08e-08
1.00e-04	4.33e-05	4.33e-05	2.08e-10
1.00e-05	4.33e-06	4.33e-06	3.28e-12
1.00e-06	4.33e-07	4.33e-07	6.99e-11
1.00e-07	4.30e-08	4.36e-08	8.18e-10
1.00e-08	3.04e-09	8.06e-09	3.04e-09
1.00e-09	4.14e-08	4.14e-08	4.14e-08
1.00e-10	4.14e-08	4.14e-08	4.14e-08
1.00e-11	4.14e-08	4.14e-08	4.14e-08
1.00e-12	4.45e-05	4.45e-05	4.45e-05
1.00e-13	4.00e-04	4.00e-04	4.00e-04
1.00e-14	4.00e-04	4.00e-04	1.07e-02
1.00e-15	5.51e-02	5.51e-02	5.59e-02

BEST

Going both ways but keep the precision

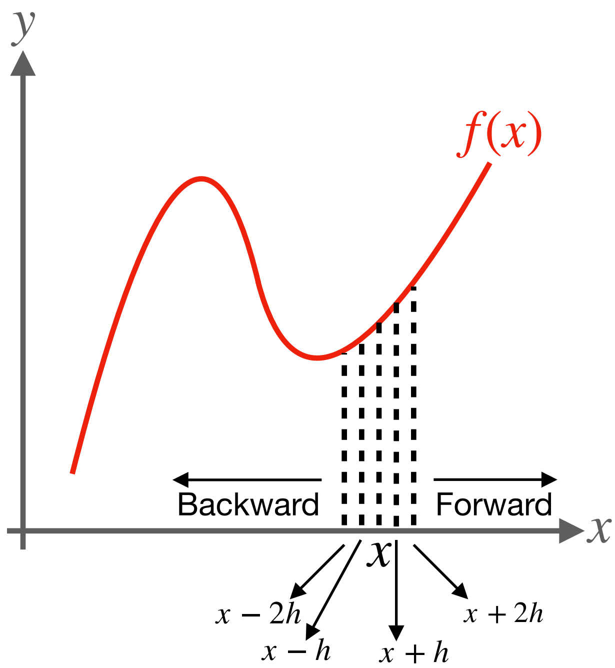
$$f\left(x - \frac{h}{2}\right) = f(x) - \frac{h}{2}f'(x) + \dots$$

$$f\left(x + \frac{h}{2}\right) = f(x) + \frac{h}{2}f'(x) + \dots$$

Central Difference Approximation

$$f'(x) = \frac{f\left(x + \frac{h}{2}\right) - f\left(x - \frac{h}{2}\right)}{h}$$

Multiple Point Finite Difference Approach



First Forward Diff. Approximation

$$f'(x) = \frac{f(x+h) - f(x)}{h}$$

First Backward Diff. Approximation

$$f'(x) = \frac{f(x) - f(x-h)}{h}$$

Central Difference Approximation

$$f'(x) = \frac{f(x+\frac{h}{2}) - f(x-\frac{h}{2})}{h}$$

$$-4 \,/\; f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \dots$$

$$+ \quad f(x+2h) = f(x) + 2hf'(x) + \frac{4h^2}{2}f''(x) + \dots$$

$$-4f(x+h) + f(x+2h) = -3f(x) - 2hf'(x)$$

Three Point Forward Approximation

$$f'(x) = \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h}$$

If we keep the precision

$$f'(x) = \frac{-3f(x) + 4f(x+\frac{h}{2}) - f(x+h)}{h}$$

$$-4 \,/\; f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) + \dots$$

$$+ \quad f(x-2h) = f(x) - 2hf'(x) + \frac{4h^2}{2}f''(x) + \dots$$

$$-4f(x-h) + f(x-2h) = -3f(x) + 2hf'(x)$$

Three Point Backward Approximation

$$f'(x) = \frac{3f(x) - 4f(x-h) + f(x-2h)}{2h}$$

$$+1 \,/\; f(x-2h) = f(x) - 2hf'(x) + \frac{4h^2}{2}f''(x) +$$

$$-8 \,/\; f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) + \dots$$

$$+8 \,/\; f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \dots$$

$$-1 \,/\; f(x+2h) = f(x) + 2hf'(x) + \frac{4h^2}{2}f''(x) +$$

$$+ \quad f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h) =$$

Four Point Central Approximation

$$f' = \frac{f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)}{12h}$$

Absolute Error

h	Three-p Forward	Three-p Backward	Four-p Central
1.00e-01	1.44e-03	1.88e-03	1.66e-06
1.00e-02	1.64e-05	1.69e-05	1.67e-10
1.00e-03	1.66e-07	1.67e-07	1.48e-13
1.00e-04	1.66e-09	1.67e-09	7.95e-13
1.00e-05	[illegible]	[illegible]	[illegible]12
1.00e-06	[illegible]	[illegible]	[illegible]11
1.00e-07	[illegible]	[illegible]	[illegible]10
1.00e-08	8.59e-09	[illegible]	3.44e-09

Higher precision even with larger h values

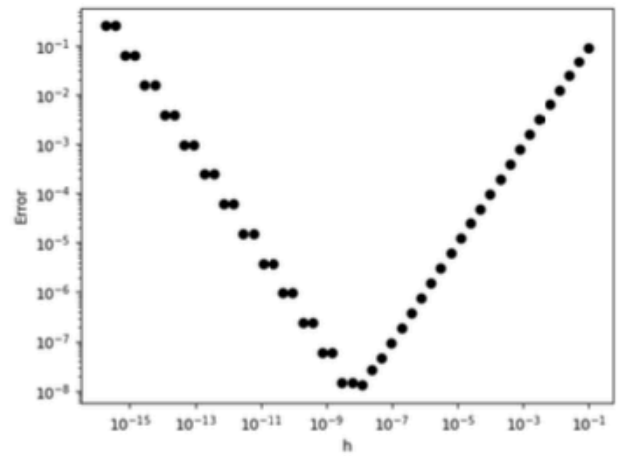
More Points More Precision More Cost

Problem Set

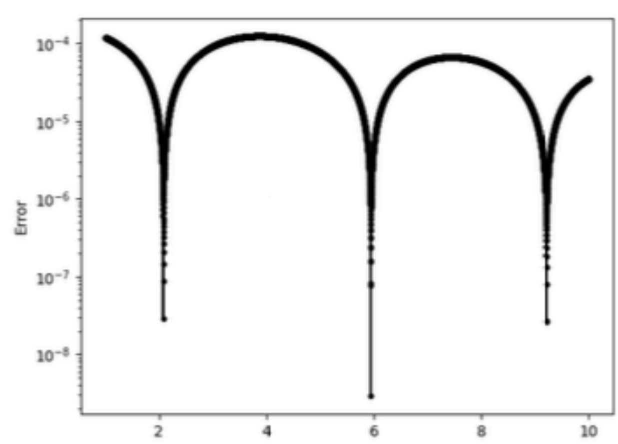
1. Derive the expression of Four Point Backward Difference Approximation.

2. To find a second derivative of a function, is it a good idea that use the same approximation two times or a combination of the approximations?

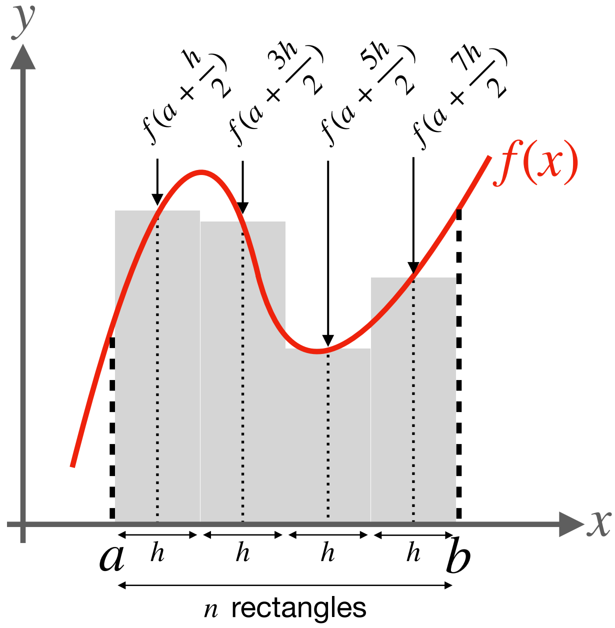
3. Find the relationship between the value of **h** and the error of the finite difference approximation. You can find an example below for the First Forward Approximation and f(x)=1/x at x=1.



4. Find the relationship between the **x** values and the resulting error of numerical derivative of the function sin(x) / x. You can find an example below.



Numerical Integration Methods



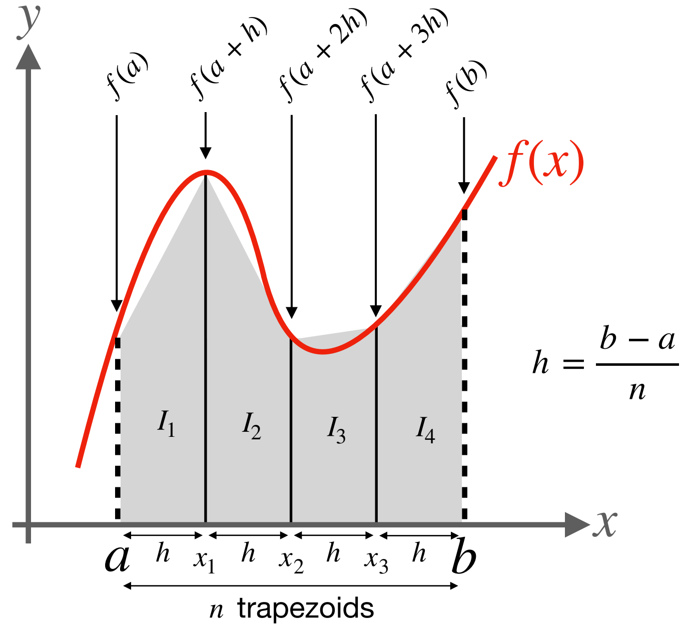
$$h = \frac{b-a}{n}$$

$$I_1 = f\left(a + \frac{1}{2}h\right)h \qquad I_3 = f\left(a + \frac{5}{2}h\right)h$$

$$I_2 = f\left(a + \frac{3}{2}h\right)h \qquad I_4 = f\left(a + \frac{7}{2}h\right)h$$

Midpoint Rule

$$\int_a^b f(x)\,dx = h\sum_{i=0}^{n-1} f\left(a + \left(i + \frac{1}{2}\right)h\right)$$



$$I_1 = f(a)h + \frac{1}{2}\left[f(a+h) - f(a)\right]h$$

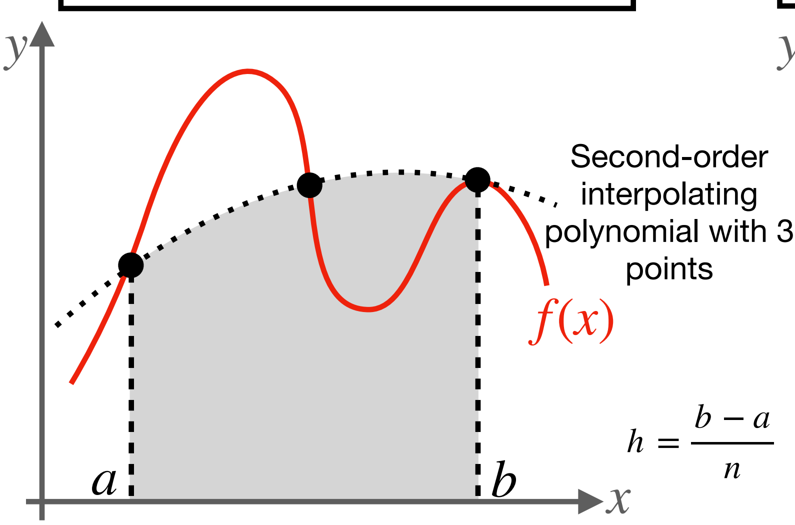
$$I_2 = f(a+h)h + \frac{1}{2}\left[f(a+2h) - f(a+h)\right]h$$

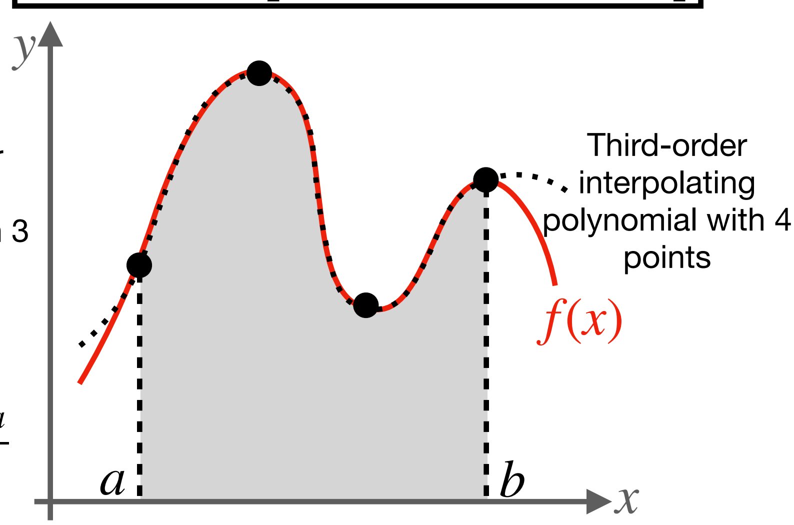
$$I_3 = f(a+2h)h + \frac{1}{2}\left[f(a+3h) - f(a+2h)\right]h$$

$$I_4 = f(a+3h)h + \frac{1}{2}\left[f(b) - f(a+3h)\right]h$$

Trapezoidal Rule

$$\int_a^b f(x)\,dx = \frac{h}{2}\left[f(a) + 2\sum_{i=1}^{n-1} f(a+ih) + f(b)\right]$$





Simpson's Rules

1/3

$$\int_a^b f(x)\,dx = \frac{h}{3}\left[f(a) + 4\sum_{i=1,3,5}^{n-1} f(a+ih) + 2\sum_{j=2,4,6}^{n-2} f(a+jh) + f(b)\right]$$

3/8

$$\int_a^b f(x)\,dx = \frac{3h}{8}\left[f(a) + 3\sum_{i=1,4,7}^{n-2}\left[f(a+ih) + f(a+(i+1)h)\right] + 2\sum_{j=3,6,9}^{n-3} f(a+jh) + f(b)\right]$$

Problem Set

1. Find the result of the integral given below by using midpoint rule and trapezoidal rule, then compare results.

$$\int_1^2 1/x$$

2. The y-coordinates of the points on a circle with the radius r can be given as:

$$y = \sqrt{r^2 - x^2}.$$

Use midpoint rule with 10 steps in the interval [0, r] to calculate the area of a circle with r=7. Calculate the error by comparing your numerical result with

$$\pi r^2.$$

3. Develop a numerical integration method which uses a well-known shape to cover the area that you want to integrate your function. As given in the example plot, you can use a rectangle which covers $x^2 + 2$ from 0 to 1. Generate random coordinates for large number of points in the covering area. Count the points which are below the function and find the ratio to the total number of points. This ratio gives you an estimation of the integral.