수치해석
(2019학년도 1학기)

[6주/2차시 학습내용]: 이분법 (Bisection) 및 가위치법(False Position)의 오차 (Error) 분석

국민대학교 소프트웨어학부

Get the real root in Python

- 참값을 찾아보자

```
import numpy as np
from scipy.optimize import fsolve
fm=lambda m: np.sqrt(9.81*m/0.25)*np.tanh(np.sqrt(9.81*0.25/m)*4)-36
m=fsolve(fm, 1)

print("Real Root= ", m)

Real Root=  [ 142.73763311]
```

Error Calculation : Approximate error and true error

- Approximate error: 상대오차

$$|e_a| = \left| \frac{x_r^{new} - x_r^{old}}{x_r^{new}} \right| \times 100(\%)$$

- True error: 절대오차

$$|e_t| = \left| \frac{x_t - x_r}{x_t} \right| \times 100(\%)$$

Downward trend

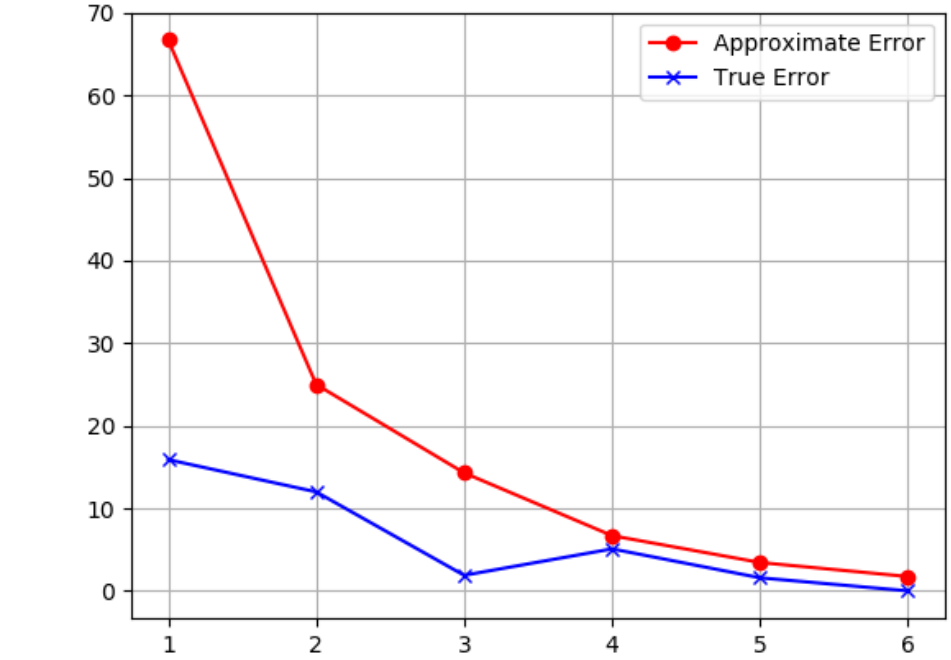
- 이분법에서의 상대오차와 절대오차 비교 그래프를 그려보자

```
import numpy as np
import matplotlib.pyplot as plt

ea=np.array([66.7, 25, 14.3, 6.7, 3.45, 1.75])
et=np.array([15.9, 12, 1.9, 5.1, 1.6, 0.016])

x=np.arange(1, 7)
plt.plot(x, ea, 'ro-', x, et, 'bx-')
plt.grid()
plt.legend(["Approximate Error", "True Error"])

plt.show()
```

https://github.com/SCKIMOSU/Numerical-Analysis/blob/master/bisection.py

Approximate error and true error

- **Although the approximate error does not provide an exact estimate of the true error, this figure suggests that approximate error captures the general downward trend of true error**



Error in Bisection

$$|e_a| = \left| \frac{x_r^{new} - x_r^{old}}{x_r^{new}} \right| \times 100(\%)$$

Iter	1. xrold	2. xr	3. Error ea	4. Test sign	5. xl	6. xu	7. ea <= es	8.iter >= maxit
1	40	120.0	66.6%	+	120	200	0	0
2	120.0	160.0	25.0%	-	120	160	0	0
3	160	140	14.3%	+	140	160	0	0
4	140	150	6.7%	-	140	150	0	0
5	150	145	3.4%	-	140	145	0	0
6	145	142.5	1.75%	+	142.5	145	0	0
7	142.5	143.75	0.89%	-	142.5	143.75	0	0
8	143.75	143.125	0.44%	-	142.5	143.125	0	0
9	143.125	142.8125	0.22%	-	142.5	142.8125	0	0
10	142.8125	142.6563	0.11%	+	142.5	142.6563	0	0

Error in False position

$$|e_a| = \left| \frac{x_r^{new} - x_r^{old}}{x_r^{new}} \right| \times 100(\%)$$

Iter	1. xrold	2. xr	3. Error ea	4. Test sign	5. xl	6. xu	7. ea <= es	8.iter >= maxit
1	40.0	179.8977	77.76%	-	40.0	200.0	0	0
2	179.89	166.85	7.81%	-	40.0	166.85	0	0
3	166.85	158.38	5.34%	-	40.0	158.38	0	0
4	158.38	152.89	3.59%	-	40.0	152.89	0	0
5	152.89	149.32	2.38%	-	40.0	149.32	0	0
6	149.32	147.01	1.57%	-	40.0	147.01	0	0
7	147.01	145.51	1.03%	-	40.0	145.51	0	0
8	145.51	144.53	0.67%	-	40.0	144.53	0	0
9	144.53	143.90	0.43%	-	40.0	143.90	0	0
10	143.90	143.49	0.28%	-	40.0	143.49	0	0

Help(list)

```
help(list)
Help on class list in module builtins:
class list(object)
 |  list() -> new empty list
 |  list(iterable) -> new list initialized from iterable's items
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
```

help(np.where(a))

```
import numpy as np
help(np.where())
Traceback (most recent call last):
  File "C:\Users\sckMacPro\Anaconda3\lib\site-
packages\IPython\core\interactiveshell.py", line 2910, in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "<ipython-input-22-053aec4148d2>", line 1, in <module>
    help(np.where())
TypeError: where() takes at least 1 argument (0 given)
help(np.where(a))
Help on tuple object:
class tuple(object)
 |  tuple() -> empty tuple
 |  tuple(iterable) -> tuple initialized from iterable's items
 |
 |  If the argument is a tuple, the return value is the same object.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
```

help(fsolve(fm, 1))

```
import numpy as np
import math
from scipy.optimize import fsolve
fm = lambda m: np.sqrt(9.81 * m / 0.25) * np.tanh(np.sqrt(9.81 * 0.25 / m) * 4) -
36
m = fsolve(fm, 1)

help(fsolve(fm, 1))
Help on ndarray object:

 class ndarray(builtins.object)
     | ndarray(shape, dtype=float, buffer=None, offset=0,
       | strides = None, order = None)
```

help(fsolve(fm, 1)) in minpack.py

C:\Users\sckMacPro\Anaconda3\Lib\site-packages\scipy\optimize

```
def fsolve(func, x0, args=(), fprime=None, full_output=0,
           col_deriv=0, xtol=1.49012e-8, maxfev=0, band=None,
           epsfcn=None, factor=100, diag=None):
    """
    Find the roots of a function.

    Return the roots of the (non-linear) equations defined by
    ``func(x) = 0`` given a starting estimate.
```

help(fsolve(fm, 1)) in minpack.py

```
Parameters
----------
func : callable ``f(x, *args)``
    A function that takes at least one (possibly vector) argument.
x0 : ndarray
    The starting estimate for the roots of ``func(x) = 0``.
args : tuple, optional
    Any extra arguments to `func`.
fprime : callable ``f(x, *args)``, optional
    A function to compute the Jacobian of `func` with derivatives
    across the rows. By default, the Jacobian will be estimated.
full_output : bool, optional
    If True, return optional outputs.
col_deriv : bool, optional
    Specify whether the Jacobian function computes derivatives down
    the columns (faster, because there is no transpose operation).
xtol : float, optional
    The calculation will terminate if the relative error between two
    consecutive iterates is at most `xtol`.
maxfev : int, optional
    The maximum number of calls to the function. If zero, then
    ``100*(N+1)`` is the maximum where N is the number of elements
    in `x0`.
```

Help in iPython

```
In [4]: import numpy as np

In [5]: help(np)
Help on package numpy:

NAME
    numpy

DESCRIPTION
    NumPy
    =====

    Provides
      1. An array object of arbitrary homogeneous items
      2. Fast mathematical operations over arrays
      3. Linear Algebra, Fourier Transforms, Random Number Generation

    How to use the documentation
    ----------------------------
    Documentation is available in two forms: docstrings provided
    with the code, and a loose standing reference guide, available from
    `the NumPy homepage <http://www.scipy.org>`_.

    We recommend exploring the docstrings using
    `IPython <http://ipython.scipy.org>`_, an advanced Python shell with
    TAB-completion and introspection capabilities.  See below for further
```