ЛР 4. Дифференцированние функции, заданной таблично.

ПИН-21 Чендемеров Алексей

May 22, 2021

1 ЛР 4. Дифференцированние функции, заданной таблично.

[1]:
```
import numpy as np
import sympy as sp
import pandas as pd
import math
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams["figure.figsize"] = (15,10)
plt.rcParams['lines.linewidth'] = 2
```

1.1 Задание 1

Выберите некоторую функцию (например, $sin(x)$, $cos(x)$, $exp(x)$, $sh(x)$, $ch(x)$, $ln(x)$, ...) и некоторую точку x из области определения функции. Найдите значение производной функции в выбранной точке (используя любую формулу численного дифференцирования) с точностью 10^{-3} , 10^{-6} . Пользоваться точным значением производной в качестве эталона запрещено.

[2]:
```
f = lambda x: np.exp(x)

def Rh2(F3, x0, h):
    x = np.linspace(x0-h, x0+h)
    y = abs(F3(x) * h**2 / 6)
    return max(y)

def derivative(f, F3, x0, eps):
    h = 1
    while Rh2(F3, x0, h) >= eps:
        h = h / 2
    return (f(x0+h) - f(x0-h)) / (2*h)

print('diff(exp(x)) = exp(x)')
F3 = np.exp
x0 = 2
eps1 = 1e-3
eps2 = 1e-6
print(f'Exp(x) at x0 = {x0}:', np.exp(x0))
```

```python
print(f'Diff of exp(x) at x0 = {x0} with eps = {eps1}:', derivative(f, F3, x0,␣
 ↪eps1))
print(f'Diff of exp(x) at x0 = {x0} with eps = {eps2}:', derivative(f, F3, x0,␣
 ↪eps2))
```

```
diff(exp(x)) = exp(x)
Exp(x) at x0 = 2: 7.38905609893065
Diff of exp(x) at x0 = 2 with eps = 0.001: 7.389356764063223
Diff of exp(x) at x0 = 2 with eps = 1e-06: 7.3890563925451715
```

Мы знаем, что $e'(x) = e(x)$. Будем использовать это для самопроверки. Для численного дифференцирования выбрал формулу центральной разности $f'(x) = \frac{f(x+h)-f(x-h)}{2h} + O(h^2)$, где $O(h^2) = \max \frac{h^2 f'''(\eta)}{6}, \eta \in [x-h, x+h]$ - погрешность вычисления.

2 Задание 2

Выберите некоторую функцию (например, $sin(x)$, $cos(x)$, $exp(x)$, $sh(x)$, $ch(x)$, $ln(x)$, ...) и некоторую точку x из области определения функции. Сравните погрешности у формул с разными порядками погрешностей (например, $y'(x) \approx \frac{y(x+h)-y(x)}{h}$ и $y'(x) \approx \frac{y(x+h)-y(x-h)}{2h}$) для последовательности убывающих шагов (например, $h = \frac{1}{2}, \frac{1}{4}, \frac{1}{8}$). С какими скоростями убывают погрешности для каждой формулы? Дайте теоретическую оценку и подтвердите ответ экспериментом.

[3]:
```python
f = lambda x: np.exp(x)

def Rh(F2, x0, h):
    x = np.linspace(x0, x0+h)
    y = abs(F2(x) * h / 2)
    return max(y)

def Rh2(F3, x0, h):
    x = np.linspace(x0-h, x0+h)
    y = abs(F3(x) * h**2 / 6)
    return max(y)

def der_formula1(f, x0, h):
    return (f(x0+h) - f(x0)) / h

def der_formula2(f, x0, h):
    return (f(x0+h) - f(x0-h)) / (2*h)

x0 = 2
F2 = np.exp
F3 = np.exp
h = np.array([1, 1/2, 1/4, 1/8, 1/16])
Oh = np.array([Rh(F2, x0, c) for c in h])
Oh2 = np.array([Rh2(F3, x0, c) for c in h])
```

```
ideal = np.array([f(x0)] * len(h))
f1_values = np.array([der_formula1(f, x0, c) for c in  h])
f2_values = np.array([der_formula2(f, x0, c) for c in  h])
columns = {
    "h" : h,
    "real": ideal,
    "f1(2)": f1_values,
    "f2(2)": f2_values,
    "O(h)": Oh,
    "diff_1": abs(f1_values - ideal),
    "O(h^2)": Oh2,
    "diff_2": abs(f2_values - ideal)
}
df = pd.DataFrame(columns, columns=columns.keys())
df
```

[3]:

```
        h      real      f1(2)     f2(2)       O(h)   f1_diff    O(h^2)  \
0  1.0000  7.389056  12.696481  8.683628  10.042768  5.307425  3.347589
1  0.5000  7.389056   9.586876  7.700805   3.045623  2.197820  0.507604
2  0.2500  7.389056   8.394719  7.466266   1.185967  1.005663  0.098831
3  0.1250  7.389056   7.870731  7.408313   0.523306  0.481675  0.021804
4  0.0625  7.389056   7.624851  7.393868   0.245800  0.235795  0.005121

    f2_diff
0  1.294571
1  0.311749
2  0.077210
3  0.019257
4  0.004812
```

$O(h)$ и $O(h^2)$ - теоретические оценки погрешонсти. *f1_diff* и *f2_diff* - экспериментальные.

Погрешность первого порядка уменьшается примерно во столько раз, во сколько раз уменьшается h. Погрешность второго порядка - в квадрат этой величины.

3 Задание 3

Неустойчивость численного дифференцирования. Выберите некторую функцию (например, $sin(x)$, $cos(x)$, $exp(x)$, $sh(x)$, $ln(x)$, ...) и некоторую точку x из области определения функции. Попробуйте применить формулу $y'(x) \approx \frac{y(x+h)-y(x)}{h}$ для стремящейся к нулю последовательности $h = \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, ...)$. Будет ли погрешность $\varepsilon = |y'(x) - \frac{y(x+h)-y(x)}{h}|$ монотонно убывать при уменьшении h? Сравните практический и теоретический результаты.

[4]:

```
def der_formula1(f, x0, h):
    return (f(x0+h) - f(x0)) / h

def Rh(F2, x0, h):
    x = np.linspace(x0, x0+h)
```

```python
    y = abs(F2(x) * h / 2)
    return max(y)

f = np.exp
F2 = np.exp
MAX_ITER = 500

h = [1/2]
x0 = 2
theor_err = []
exper_err = []
df = []
for i in range(MAX_ITER):
    theor_err.append(Rh(F2, x0, h[-1]))
    df.append(der_formula1(f, x0, h[-1]))
    exper_err.append(abs(df[-1]-f(x0)))
    if df[-1] == 0:
        break
    h.append(h[-1]/2)

h = np.array(h)
theor_err = np.array(theor_err)
exper_err = np.array(exper_err)
df = np.array(df)

columns = {
    "h": h,
    "df": df,
    "theor_err": theor_err,
    "exper_err": exper_err
}
dframe = pd.DataFrame(columns, columns=columns.keys())
dframe
```

[4]:

	h	df	theor_err	exper_err
0	5.000000e-01	9.586876	3.045623e+00	2.197820e+00
1	2.500000e-01	8.394719	1.185967e+00	1.005663e+00
2	1.250000e-01	7.870731	5.233061e-01	4.816750e-01
3	6.250000e-02	7.624851	2.458003e-01	2.357947e-01
4	3.125000e-02	7.505722	1.191189e-01	1.166661e-01
5	1.562500e-02	7.447085	5.863607e-02	5.802884e-02
6	7.812500e-03	7.417995	2.908988e-02	2.893881e-02
7	3.906250e-03	7.403507	1.448823e-02	1.445056e-02
8	1.953125e-03	7.396277	7.229982e-03	7.220575e-03
9	9.765625e-04	7.392665	3.611463e-03	3.609112e-03
10	4.882812e-04	7.390860	1.804850e-03	1.804262e-03
11	2.441406e-04	7.389958	9.022046e-04	9.020578e-04

12	1.220703e-04	7.389507	4.510472e-04	4.510105e-04
13	6.103516e-05	7.389282	2.255099e-04	2.255007e-04
14	3.051758e-05	7.389169	1.127515e-04	1.127492e-04
15	1.525879e-05	7.389112	5.637488e-05	5.637430e-05
16	7.629395e-06	7.389084	2.818723e-05	2.818701e-05
17	3.814697e-06	7.389070	1.409356e-05	1.409342e-05
18	1.907349e-06	7.389063	7.046766e-06	7.046569e-06
19	9.536743e-07	7.389060	3.523380e-06	3.522910e-06
20	4.768372e-07	7.389058	1.761689e-06	1.760848e-06
21	2.384186e-07	7.389057	8.808443e-07	8.816792e-07
22	1.192093e-07	7.389057	4.404221e-07	4.420950e-07
23	5.960464e-08	7.389056	2.202110e-07	2.111270e-07
24	2.980232e-08	7.389056	1.101055e-07	1.068189e-07
25	1.490116e-08	7.389056	5.505276e-08	4.721422e-08
26	7.450581e-09	7.389056	2.752638e-08	1.239043e-08
27	3.725290e-09	7.389056	1.376319e-08	1.315997e-07
28	1.862645e-09	7.389056	6.881595e-09	1.068189e-07
29	9.313226e-10	7.389056	3.440797e-09	1.068189e-07
30	4.656613e-10	7.389055	1.720399e-09	8.468555e-07
31	2.328306e-10	7.389057	8.601993e-10	1.060493e-06
32	1.164153e-10	7.389053	4.300997e-10	2.754204e-06
33	5.820766e-11	7.389053	2.150498e-10	2.754204e-06
34	2.910383e-11	7.389038	1.075249e-10	1.801299e-05
35	1.455192e-11	7.389038	5.376246e-11	1.801299e-05
36	7.275958e-12	7.389038	2.688123e-11	1.801299e-05
37	3.637979e-12	7.388916	1.344061e-11	1.400833e-04
38	1.818989e-12	7.389160	6.720307e-12	1.040573e-04
39	9.094947e-13	7.388672	3.360154e-12	3.842239e-04
40	4.547474e-13	7.388672	1.680077e-12	3.842239e-04
41	2.273737e-13	7.386719	8.400384e-13	2.337349e-03
42	1.136868e-13	7.390625	4.200192e-13	1.568901e-03
43	5.684342e-14	7.390625	2.100096e-13	1.568901e-03
44	2.842171e-14	7.375000	1.050048e-13	1.405610e-02
45	1.421085e-14	7.375000	5.250240e-14	1.405610e-02
46	7.105427e-15	7.375000	2.625120e-14	1.405610e-02
47	3.552714e-15	7.250000	1.312560e-14	1.390561e-01
48	1.776357e-15	7.500000	6.562800e-15	1.109439e-01
49	8.881784e-16	7.000000	3.281400e-15	3.890561e-01
50	4.440892e-16	6.000000	1.640700e-15	1.389056e+00
51	2.220446e-16	0.000000	8.203500e-16	7.389056e+00

При малых h (порядка 10^{-16}) ЭВМ считает $x + h = x$. Следовательно, численная производная в таких случаях будет равна 0. В таком случае погрешность будет максимальной и будет равна $y'(x_0)$. До тех пор, пока это не произойдёт погрешность действительно будет монотонно убывать.