# FYS-4096 (2017) Exercise 2: Numerical calculus

## Summary

You're going to derive a peculiar finite-difference approximation for derivative, calculate a lot of integrals, and differentiate functions using FFT.

Return you solution following instructions on the protected course web page to GitLab **before Friday 5 am**.

Wednesday's tutorial in TC217 starting at 12:15.

**Write answers to open questions in the `README.rst`-file.**

## 1. Numerical differentiation (4 XPs)

A mysterious stranger approaches you as you are walking home. He conjures up a papyrus scroll which supposedly holds important data for his life's work on finding a potion for eternal youth. Unfortunately, there's an important piece of data missing: the first derivative of time-series data.

This idiot has measured the data with weird sampling intervals:

| ‘$t$‘ (hours) | ‘$f(t)$‘ (undisclosed units) |
|---|---|
| 0.1333333333 | 0.399936791 |
| 0.2 | 0.399680042 |
| 0.25 | 0.399219004 |
| 0.2666666666 | 0.398989068 |
| 0.5333333333 | 0.383927081 |
| 0.6 | 0.374358729 |
| 0.65 | 0.364826674 |
| 0.6666666666 | 0.361139867 |

Your task is to estimate the derivative of ‘$f$‘ at ‘$t = 0.2$‘ hours and ‘$t = 0.6$‘ hours.

### 1.A Deriving a finite-difference scheme (2 / 4 XPs)

Derive analytically a finite-difference differentiation scheme for the first derivative of ‘$f(x)$‘ when the function values are know at points ‘$x - \frac{h}{3}$‘, ‘$x$‘, ‘$x + \frac{h}{4}$‘, and ‘$x + \frac{h}{3}$‘. Derive such a scheme that the **error term** is of third order, i.e., ‘$\mathcal{O}(h^3)$‘.

If you do this on a paper, please scan your solution and save it as a PDF at the root of your repository.

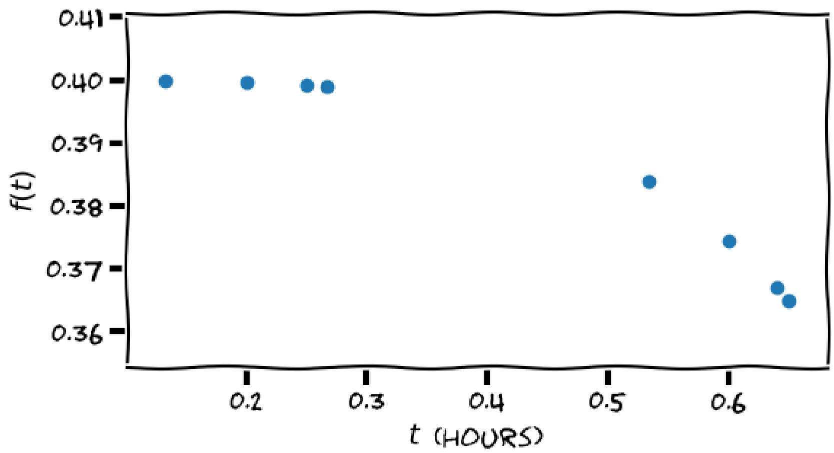


Figure 1: Scatter plot of the stranger's data

### 1.B Implementation (2 / 4 XPs)

In a similar fashion as in last week's exercises, implement your scheme as a Python module under `num_calculus.differentiation`, add testing framework, documentation etc.

The signature should be

```
def eval_derivative_mysterious_stranger(x, y)
```

and should be callable like

```
x = np.array([ 0.1333333333, 0.2, 0.25, 0.26666666666666])
y = np.array([ 0.399936791, 0.399680042, 0.399219004, 0.398989068])

der = eval_derivative_mysterious_stranger(x, y)
```

Finally, write a script `scripts/mysterious_stranger.py` which evaluates and prints out the derivative of '$f$' from the above data at '$t = 0.2$' and '$t = 0.6$'.

## 2. Numerical integration (3 XPs)

### 2.A Implementation

For this exercise you need **not** make your own implementation of the integration routines. We'll use those available in `scipy.integrate` throughout this exercise.

**Remember to comment your scripts so that they are easy to read.**

### 2.B Easy integrals (1 / 3 XPs)

Create a script `scripts/easy_integrals.py` which numerically evaluates and prints out the following integrals when the integrand is evaluated **only** at points ‘$\{0, 0.1, 0.2, 0.3, ..., 4.9, 5\}$‘. Use trapezoidal rule and Simpson’s rule for each of these integrals. Compare to the exact value.

‘$\int\limits_0^5 x^2 \, \mathrm{d}x$‘

‘$\int\limits_0^5 \exp(\sin(x^3)) \, \mathrm{d}x$‘

### 2.C Non-trivial integrals (1 / 3 XPs)

Create a script `scripts/non_trivial_integrals.py` which numerically evaluates and prints out the following integrals when you are allowed to evaluate the integrand **only** at finite number of **pre-determined** points (i.e., no adaptive integration allowed, but you can choose the sampling points as you wish). You are allowed to do pre-prosessing or reformulation of the integrals manually, but please make a note the the source files if you do this.

‘$\int\limits_0^1 \frac{1}{\sqrt{x}} \, \mathrm{d}x$‘

‘$\int\limits_0^1 \frac{\sin(x)}{x} \, \mathrm{d}x$‘

‘$\int\limits_0^\infty \exp(-x) \, \mathrm{d}x$‘

### 2.D Insanely difficult integrals (1 / 3 XPs)

Create a script `scripts/insanely_difficult_integrals.py` which numerically evaluates and prints out the following integrals. You are allowed to use all the tricks up your sleeve.

‘$\int\limits_0^\infty \mathrm{J}_1(x) \, \mathrm{d}x$‘ where ‘$J_n(x)$‘ is the ‘$n$‘th Bessel function of the first kind (`scipy.special.j1`).

‘$\int\limits_2^{15} \log[|\Gamma(-x)|] \, \mathrm{d}x$‘ where ‘$\Gamma(x)$‘ is the Gamma function (`scipy.special.gamma`, `numpy.real`).

‘$\oint\limits_{\{z \in \mathbb{C}: |z|=2\}, \mathrm{CCW}} \frac{1}{z-1} \, \mathrm{d}z$‘, i.e., a complex line integral **counter-clockwise** along a circle of radius 2 in the complex plane.

## 3. FFT and differentiation (3 XPs)

### 3.A Implementation (1 / 3 XPs)

Implement a function with signature

```
def deriv_fft(x, y)
```

in the Python module `num_calculus.differentiation` in your repository. The function should evaluate the first order derivative of the input '$y(x)$' using the Fourier method. Introduced in lectures. `x` should be an equally spaced array of numbers (floats) and `y` the corresponding function values.

Remember to include appropriate documentation, packaging files, tests etc.

This function should be callable like

```
x = numpy.linspace(0, 5, 100)
y = numpy.sin(2*numpy.pi*x/5.)**2

y_deriv = deriv_fft(x, y)
```

### 3.B Periodic functions (1 / 3 XPs)

Evaluate the function '$f(x) = 2\sin\left(\frac{36\pi}{7}x\right)\sin\left(\frac{\pi}{7}x\right)$' on an equidistant grid from '$x = 0$' to '$x = 7$' (non-inclusive). Calculate its first derivative using the function you implemented in 3.A. Evaluate the derivative also using some finite difference scheme and plot your resulting $y'(x)$. Automate all this in a script `scripts/diff_with_fft_periodic_fun.py`. Is there a major difference between the FFT and FD derivatives? Why/why not?

### 3.C Non-periodic functions (1 / 3 XPs)

Evaluate the function '$f(x) = 2\sin\left(\frac{36\pi}{7}x^2\right)\exp[\cos\left(\frac{\pi}{7}x\right)]$' on a grid from '$x_0 = 0$' to '$x_0 = 2$', and calculate its first derivative using the function you implemented in 3.A. Evaluate the derivative also using some finite difference schemes and plot your results. Automate all this in a script `scripts/diff_with_fft_nonperiodic_fun.py`. Is there a difference between the FFT and FD derivatives? Why/why not?

## Returning your solution

Return your solution to a new GitLab project under your group for this course. Name the project `exercise2`, and tag the final version with `final`.

**Add 'problems_solved' to the repo's root, and write a comma separated list of solved problems in the file. For example:**

```
1A,2A,2B,3A,3C
```

In the end, you should have at least the following files in your repository:

- `scripts/diff_with_fft_nonperiodic_fun.py`
- `scripts/diff_with_fft_periodic_fun.py`
- `scripts/easy_integrals.py`
- `fd_scheme_derivation.pdf`
- `scripts/insanely_difficult_integrals.py`
- `scripts/mysterious_stranger.py`
- `scripts/non_trivial_integrals.py`
- `num_calculus/__init__.py`
- `num_calculus/differentiation.py`
- `num_calculus/tests/__init__.py`
- `num_calculus/tests/test_differentiation_using_fft.py`
- `setup.py`
- `README.rst` <- GitLab apperently knows how to interpret these :)
- `license.txt`
- `problems_solved`

Remember that the XPs you gain from each exercise will be based on, e.g.,

- the final report including figures (quality, understanding of the numerical method, . . . ), (`README.rst`)
- correct and working implementation,
- quality and amount of automatic tests,
- implementation design (modularity, ease of use, easy to test, . . . ),
- documentation and code comments,
- Git commits (comment quality, amount),
- PEP 8 compliant programming style.