Report file - Problem Set #5

Matteo Dell'Acqua
GitHub: MatteoDellAcqua6121

October 8, 2024

Abstract

This is the report for the problem set #5. Since the problem set is composed of three exercises, we divide the report into three sections, one for each problem. The scripts (labelled as ps_4.'problem number') and the raw file of the images are in this directory.

1 Problem 1

1.1 Formulation of the problem

We are asked to compute the derivative of the function:

$$f(x) = 1 + \frac{1}{2}\tanh(2x), \tag{1}$$

both using symmetric difference and autodiff methods.

1.2 Computational methods

The symmetric difference approximation consists in computing the derivative of a function:

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = \lim_{\mathrm{d}x \to 0} \frac{f(x+\mathrm{d}x) - f(x)}{\mathrm{d}x}, \tag{2}$$

via the increment between two points whose distance from x is finite but small (and such that x is their average):

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} \sim \frac{f(x+\mathrm{d}x/2) - f(x-\mathrm{d}x/2)}{\mathrm{d}x}, \tag{3}$$

The actual implementation can be taken from the Jupiter notebook of the class:

```
def diff_central(func=None, x=None, dx=None):
    return((func(x + 0.5 * dx) - func(x - 0.5 * dx)) / dx)
```

An alternative way to compute the derivative is by rewriting my original function f as a composition of easier functions f_i:

$$f(x) = f_2 \circ f_1 \circ f_0(x), \tag{4}$$

whose derivative is known, and use the chain rule:

$$f'(x) = f_2'(f_1(f_0(x)) \cdot f_1'(f_0(x)) \cdot f_0'(x) \tag{5}$$

In our case, we have:

$$f_0(x) = 2x, \quad f_1(x) = \tanh(x), \quad f_2(x) = 1 + \frac{x}{2} \tag{6}$$

$$f_0'(x) = 2, \quad f_1'(x) = 1 - \tanh^2(x), \quad f_2'(x) = \frac{1}{2}. \tag{7}$$

In order to implement this method, called *autodifference* we need to define the intermediate functions f_i such that they output both the value of the function itself and its derivative:

$$F_i(x) := (f_i(x), f_i'(x)) \tag{8}$$

and smartly rewrite the chain rule in terms of these new functions.

The process just described is easily implemented via the `jax` package, which automatically takes care of all the steps described:

```
dv_jax = jax.grad(f_jax)
dv = jax.vmap(dv_jax)(x)
```

1.3 Results

We report the results of our computations (plotted using `matplotlip.pyplot`) in fig. 1. We compare them with the analytical result:

$$f(x) = 1 - \tanh^2(2x). \tag{9}$$

2 Problem 2

2.1 Formulation of the problem

In this problem we are going to compute the Gamma function:

$$\Gamma(a) = \int_0^\infty x^{a-1} e^{-x} \mathrm{d}x. \tag{10}$$

Before explaining e computational methods, we are asking to show to simple results:

Lemma 1 *The following two statements are true:*

1. *Defining $f_a(x)$ the integrand appearing in the definition of the Gamma function 10:*

$$f_a(x) = x^{a-1} e^{-x}, \quad f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0} \tag{11}$$

its maximum value is realized for $x = a - 1$ (for $a > 1$).

2. *Given the change of variable:*

$$z = \frac{x}{c + x}, \tag{12}$$

we have that $z = 1/2$ for $\overline{x} = c$.

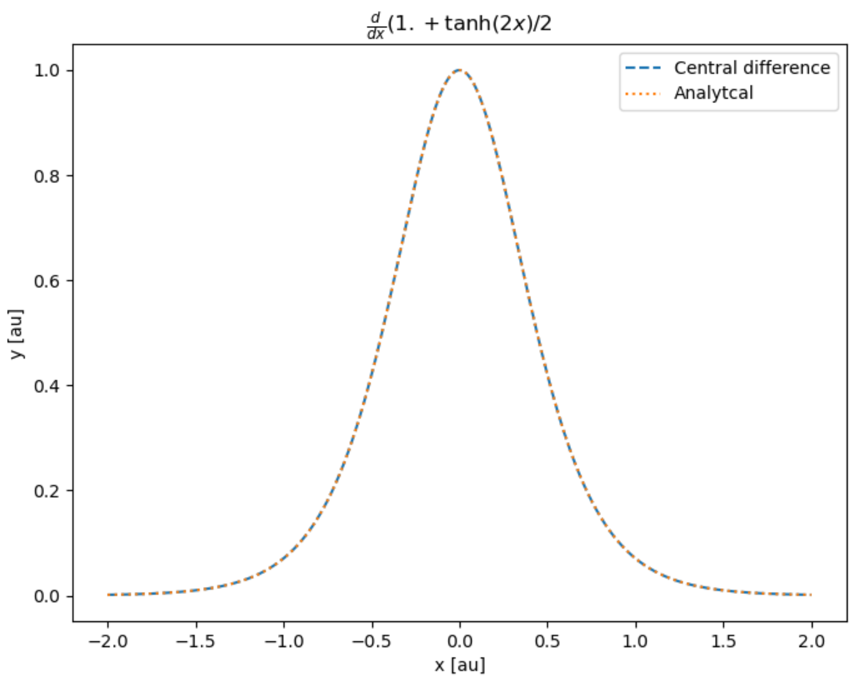


Figure 1: Derivative of the function $f(x)$ defined in 1, computed either analytically (solid line), using symmetric difference (dotted line) or autodifference (implemented via `jax`, dashed line); plotted in the interval $x \in [-2, 2]$.

The second part of the lemma immediately follows from inverting the relation:

$$x = \frac{cz}{1-z} \quad \Rightarrow \quad \overline{x} = \frac{\frac{c}{2}}{\frac{1}{2}} = c. \tag{13}$$

To show the first part of the lemma we can compute the derivative of $f_a(x)$:

$$f_a'(x) = (a-1)x^{a-2}e^{-x} - x^{a-1}e^{-x} = [(a-1)-x]x^{a-2}e^{-x}. \tag{14}$$

This function has two roots, for $x = 0$ and $x = a-1$ (the condition $a > 1$ is needed for the second root to lie inside the function domain and be distinct from the first one). A quick computation shows:

$$f_a(0) = 0, \quad f_a(a-1) = (a-1)^{a-1}e^{1-a} > 0, \quad \lim_{x\to\infty} f_a(x) = 0. \tag{15}$$

As a final check one can also evaluate the second derivative at the critical point[1]:

$$f_a''(a-1) = (a^2 - 3a + 2 - x^2)x^{a-3}e^{-x}\big|_{x=a-1} = -(a-1)^{a-2}e^{-(a-1)} \le 0. \tag{16}$$

showing that $x = a-1$ is a local maximum, it is the only critical point in the interior of the domain and it is greater than the value of the function at the boundaries of the domain: it is thus the global maximum. □

Corollary 1 *The change of variable for which the maximum of the integrand function happens at $z = 1/2$ is given by eq. (12) with $c = a-1$.*

2.2 Computational methods

The implementation consists of a Gaussian quadrature as done in the last problem set. The only peculiarity is given by the change of variable:

$$z = \frac{x}{a-1+x}, \qquad x = \frac{(a-1)z}{1-z}, \qquad \frac{(a-1)}{(a-1+x)^2}\mathrm{d}x = \mathrm{d}z \tag{17}$$

which is reflected in the code as follows:

```
def gamma(a):
    xp,wp=np.polynomial.legendre.leggauss(N)
    xp=q(xp,a)
    wp=wp/(a-1)
    int_temp=np.zeros(N, dtype=np.float32)
    def g(x,a):
         return f(x,a)*(a-1+x)**2
    for i in np.arange(N):
         int_temp[i]=wp[i]*g(xp[i],a)
    return int_temp.sum()
```

with q the function implementing the change of variable $q = x(z)$ in eq. (17), and $f(x,a) = f_a(x)$ the integrand.

It is important to note that defining f_a as in eq. (10) leads to numerical overflow and

[1]With the help of Mathematica.

underflow errors for $x >> 1$ because x^{a-1} becomes really large and is multiplied by e^{-x} which is instead really small. It is thus useful to compute it using the equivalent formulation:

$$f_a(x) = e^{(a-1)\log(x)-x} \tag{18}$$

since $(a-1)\log(x)$ and x are much closer to each other than x^{a-1} and e^{-x} (due to the properties of the logarithm if the *magnitude difference* of the second two is equal to the *ration* of the first two).

Alternatively one could Gauss-Laguerre quadrature which, similarly to the Gauss Hermite, allows for computations of integrals of the form:

$$\int_0^{+\infty} e^{-x}h(x) \simeq \sum_{i=0}^{N-1} w_i h(x_i) \tag{19}$$

where now x_i are the roots of the Nth Laguerre polynomial. Roots and weight are found using `np.polynomial.laguerre.lagauss`. In our case, $h(x) = x^{a-1}$, thus the result is exact for $N \geq \frac{a}{2} - 1$ points.

2.3 Results

We report the plot of the integrand of the gamma function for various values of a, created using `matplotlip.pyplot`, in fig. 2. We can visually check that the maximum occurs at $x = a - 1$.

Moreover, we can compute the entire gamma function (with $N = 10$ roots) for $a \in \{1.5, 3, 6, 10\}$ and compare them with the tabulated results ($\delta\Gamma$ denotes the *relative* error in absolute value):

$$\Gamma(1.5) = \frac{1}{2}\sqrt{\pi} \sim 0.886, \quad \Gamma_{leg}(1.5) = 0.9103, \quad \delta\Gamma_{leg}(1.5) = 0.0272 \quad \Gamma_{\mathrm{lag}}(1.5) = 0.889, \quad \delta\Gamma_{\mathrm{lag}}(1.5) = 0.0116, \tag{20}$$

$$\Gamma(3) = 2, \quad \Gamma_{leg}(1.5) = 2.033, \quad \delta\Gamma_{leg}(1.5) = 0.0165 \quad \Gamma_{\mathrm{lag}}(3) = 1.999, \quad \delta\Gamma_{\mathrm{lag}}(3) = -5.960 \cdot 10^{-8}, \tag{21}$$

$$\Gamma(6) = 120, \quad \Gamma_{leg}(1.5) = 110.95, \quad \delta\Gamma_{leg}(1.5) = 0.075 \quad \Gamma_{\mathrm{lag}}(6) = 120.00, \quad \delta\Gamma_{\mathrm{lag}}(6) = 0.00, \tag{22}$$

$$\Gamma(10) = 362880, \quad \Gamma_{leg}(1.5) = 301804.44, \quad \delta\Gamma_{leg}(1.5) = 0.1683 \quad \Gamma_{\mathrm{lag}}(10) = 3.6288 \cdot 10^5, \quad \delta\Gamma_{\mathrm{lag}}(10) = 8.612 \cdot 10^{-8}. \tag{23}$$

As expected the Gauss-Laguerre quadrature is significantly more precise (it is exact except for $a = 1.5$).

3 Problem 3

3.1 Formulation of the problem

In this problem, we are going to fit some data using the SVD technique.

Theorem 1 *Any $M \times N$ matrix A admits a singular value decomposition (SVD):*

$$A = UDV^T, \tag{24}$$

where U is $M \times N$ matrix whose columns are orthonormal, W is an $N \times N$ diagonal matrix, and V^T is the transpose of an $N \times N$ orthonormal matrix.

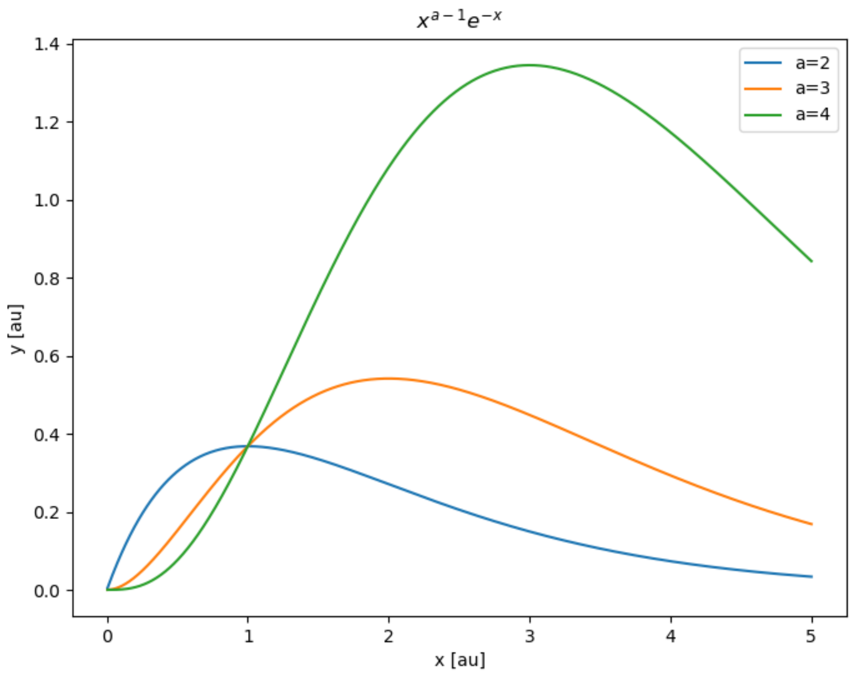


Figure 2: Plot of the integrand of the gamma function for $a \in \{2, 3, 4\}$ in the interval $x \in [0, 5]$

We can use this theorem to fit some data in the following way. Let's say our fit function is a polynomial of degree m:

$$f(x) = \sum_{i=0}^{m} a_i x^i. \tag{25}$$

The above espression can be rewritten as:

$$f(x) = A \cdot b, \tag{26}$$

where A is a $1 \times (m+1)$ matrix with entries $A = (1, x, x^2, \ldots, x^m)$ and b is the coefficient vector $b^T = (a_0, a_1, \ldots, a_m)$. Now, if we have some (M) data points whose coordinates are stored in two vectors x, y (mind the abuse of notation) the fit problem consists of finding the best coefficients b such that:

$$y = A \cdot b \tag{27}$$

where now A is line-wise as before:

$$A_i = (1, x_i, x_i^2, \ldots, x_i^m). \tag{28}$$

The problem is solved by simply inverting the matrix:

$$b = A^{-1} \cdot y. \tag{29}$$

Unfortunately, the inverse of a matrix exists only for square non-degenerate ones! The SVD allows us to circumvent this problem and write:

$$b = \big(V \cdot W^{-1} \cdot U^T\big) y. \tag{30}$$

Obviously, a similar procedure applies to other types of fit functions. For example, for:

$$f(x) = c + \sum_{j=1}^{m} a_j \sin 2\pi j x/T + \sum_{j=1}^{m} b_j \cos 2\pi j x/T, \tag{31}$$

we just need to define:

$$A_i = (1, \sin 2\pi x/T, \sin 2\pi 2x/T, \ldots, \sin 2\pi m x/T, \cos 2\pi x/T, \cos 2\pi 2x/T, \ldots, \cos 2\pi m x/T), \tag{32}$$

$$b^T = (x, a_1, \ldots, a_m, b_1, \ldots, b_m). \tag{33}$$

3.2 Computational methods

Before starting with the actual solution, we import the data using the `genfromtxt` funciton of NumPy:

```
data = np.genfromtxt('signal.dat',
                     skip_header=1,
                     skip_footer=0,
                     dtype=np.float32,
                     delimiter='|')
```

and, since the independent variable is always of the form $c \cdot 10^8$ with c of the order of the unit, we rescale it:

```
x=(x-np.mean(x))/np.std(x)
```

in order for the various fits to work better (and not suffer from round-off errors).

The implementation is straightforward. For the polynomial fit:

```
A = np.zeros((len(x), N))
for i in np.arange(N):
    if i==0:
        A[:, i] = 1.
    else:
        A[:, i] = x**i

(u, w, vt) = np.linalg.svd(A, full_matrices=False)
ainv = vt.transpose().dot(np.diag(1. / w)).dot(u.transpose())
b = ainv.dot(y)
bm=A.dot(b)
```

with b_m being the points resulting from the actual fit. And analogously for the sin/cosine fit, we just change the lines:

```
t=np.max(x)
A = np.zeros((len(x), 2*M+1))
for i in np.arange(2*M+1):
    if i==0:
        A[:, i] = 1.
    elif i<=M:
        A[:, i] = np.sin(i*np.pi*x/t)
    else:
        A[:, i] = np.cos((i-M)*np.pi*x/t)
```

where we set the lowest frequencies to be the inverse of the maximum t of the (rescaled) independent variable (and the others being its multiples).

3.3 Results

We report the original signal as well as teh results of the fits and their error (realized using `matplotlip.pyplot`) respectively in figs. 3 to 6.

It is evident that a polynomial of degree three is *not* a good fit: visually, there is a pattern in the residual and their magnitude is significantly bigger of the known standard deviation of 2.0[2].

We step-increment the degree N of the polynomial until we reach a visually satisfying fit whose residual no longer follow a pattern and are somewhat concentrated in the $[-2.0, 2.0]$ interval. These conditions are achieved around $N = 30$. However, this cannot be considered a good explanation of the data either, since the condition number is too close to the machine's precision to guarantee stable numerics:

$$c_{30} \sim 1.031 \times 10^{11}. \tag{34}$$

[2]For completeness, the condition number in this case is given by $c_3 = 5.91995$.

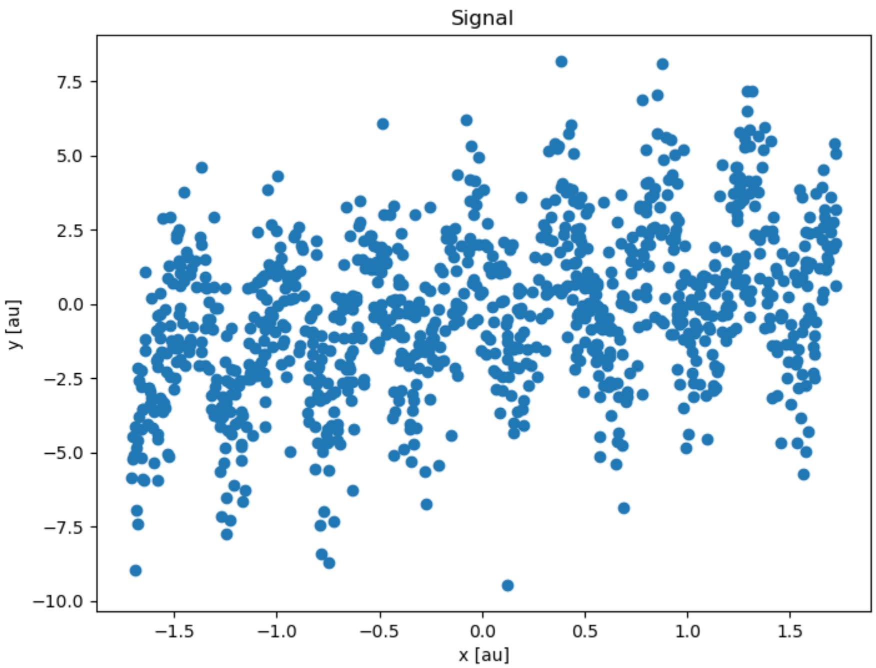


Figure 3: Raw signal (with the time rescaled as explained in the main text).

Finally, we can see that the harmonic fit works better than the previous two: it is visually evident that it is a good explanation for the frequency of the data, but not so good for the amplitudes; and the residual do not show particular patterns and have limited magnitude.

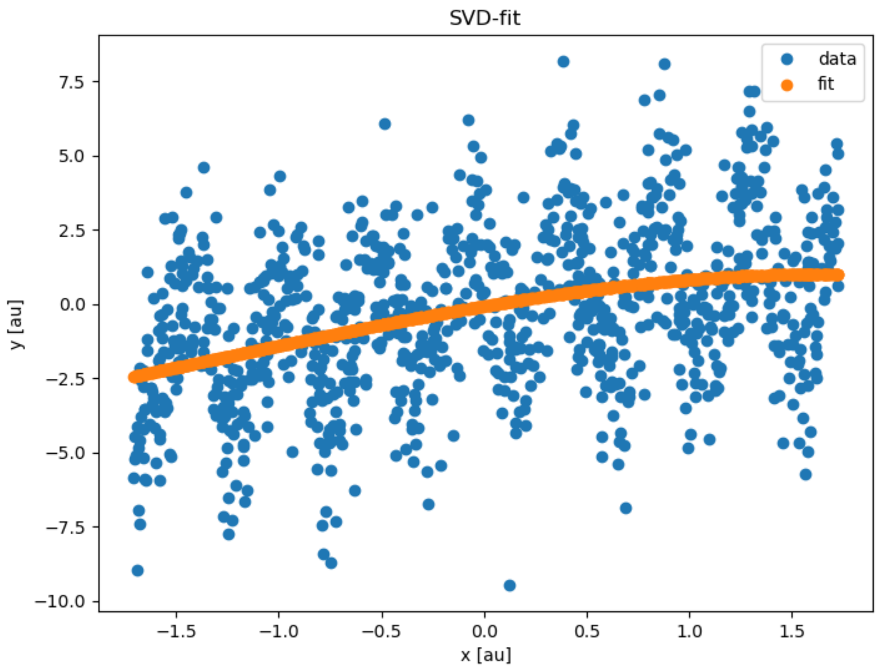
From the plot one could deduce a period of roughly:

$$\frac{T_{\max} - T_{\min}}{7.5} \sim 133219268 \text{time unit} \tag{35}$$

by counting the number of peaks.

The condition number is given by:

$$c_{\sin} = 1.8648. \tag{36}$$



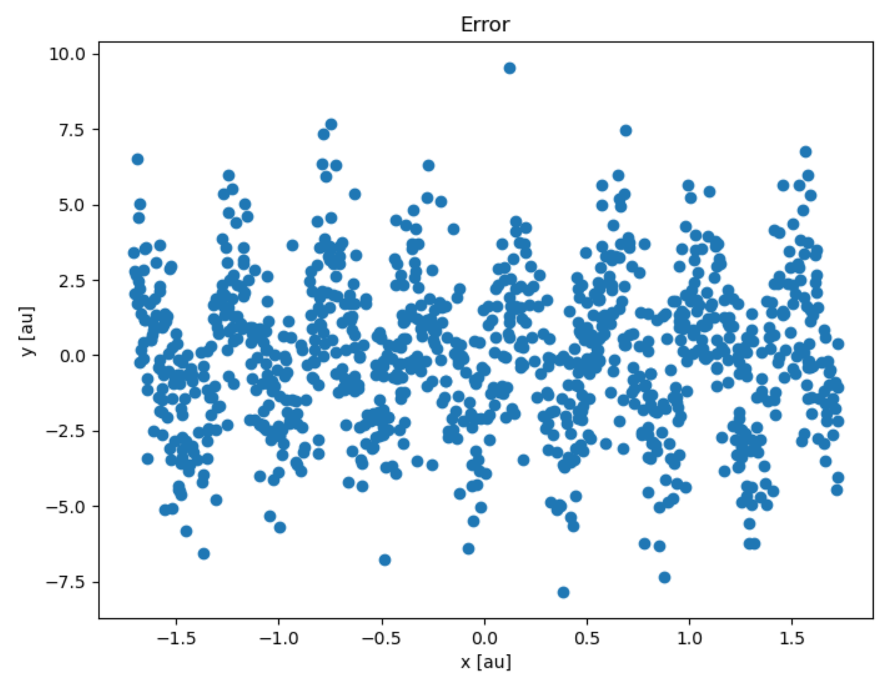


Figure 4: Result of a fit with a polynomial of degree 4 and the corresponding residuals.

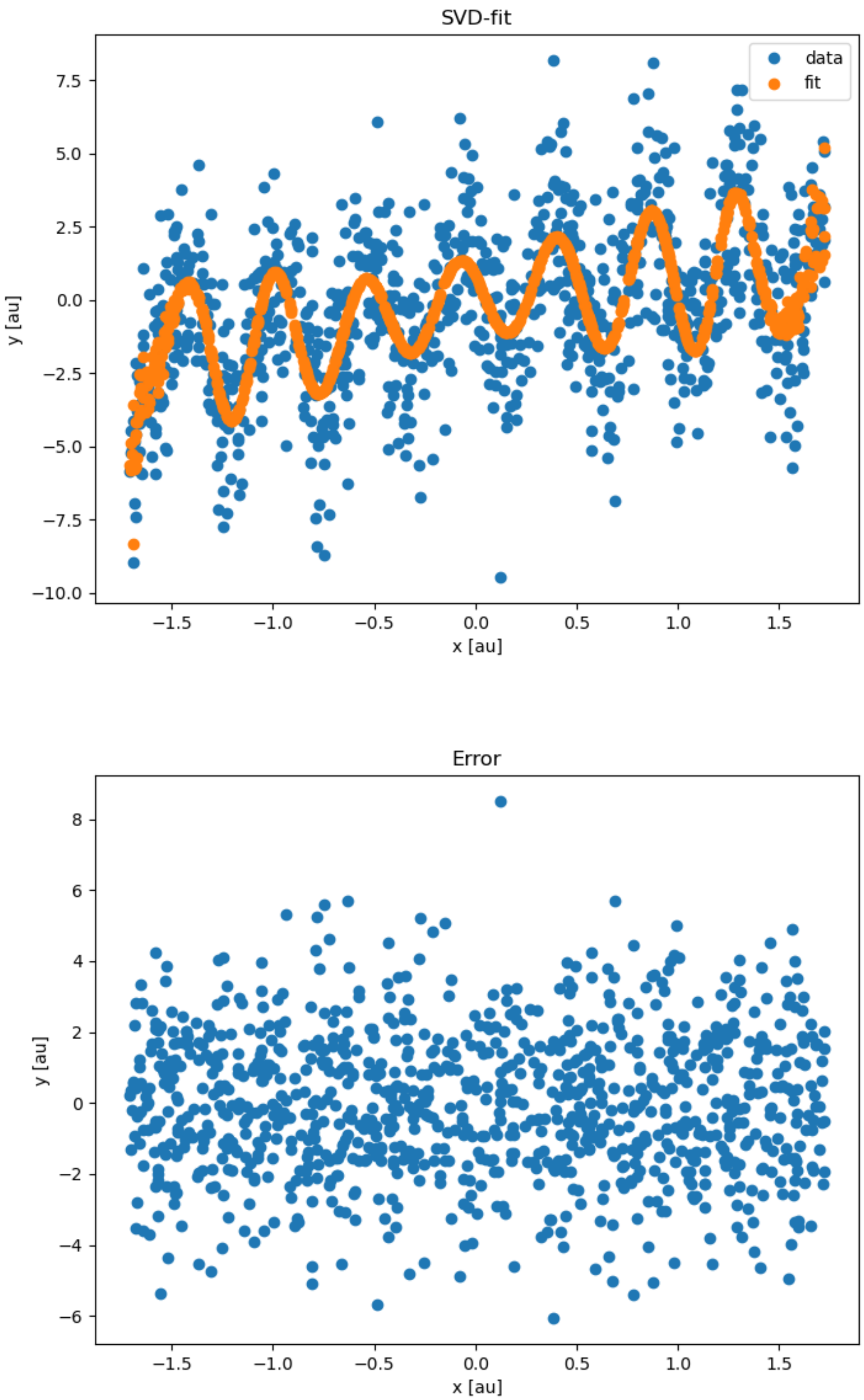
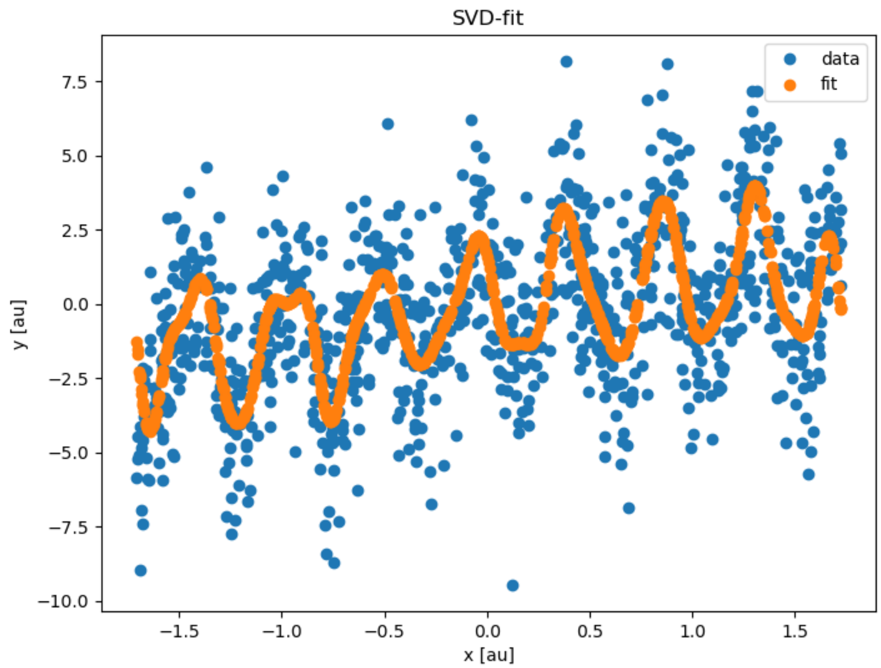


Figure 5: Result of a fit with a polynomial of degree 30 and the corresponding residuals.



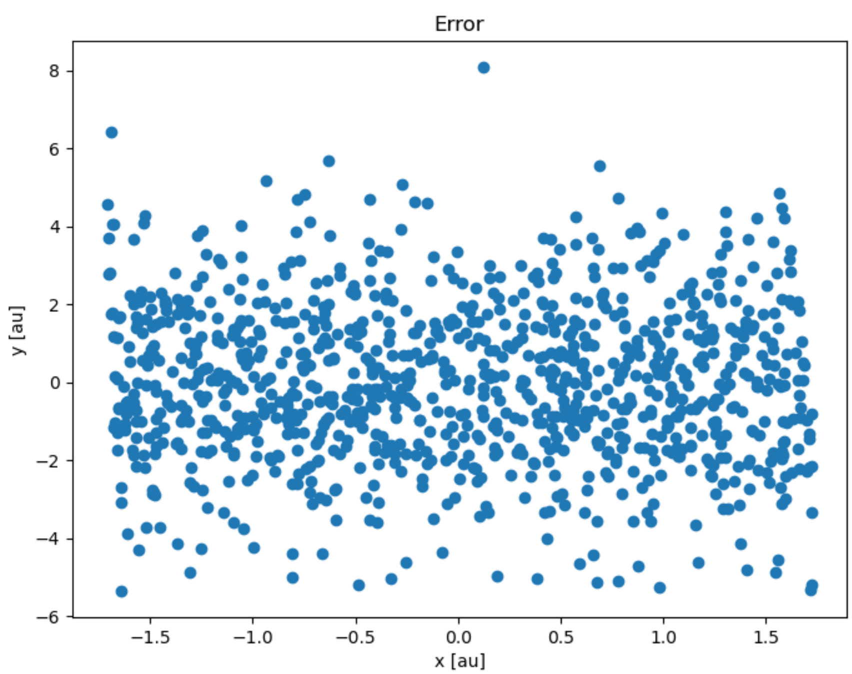


Figure 6: Result of a fit with a series of 30 sine and cosine function of increasing frequencies and the corresponding residuals.