Project 1

Fitting to Experimental Data with `scipy.optimize`

Joseph Temple

Arkansas Tech University
PHYS 4023 Computational Physics
September 26, 2025

1 Introduction

Experimental data is often wrought with deviation from theoretically predicted behavior. Be it through human mistakes, imprecise measurement tools, or the slightest shift in environment, it is nigh-on-impossible to perform an experiment full unperturbed. As such, it behooves us to have computational tools to deal with such imperfections. One such tool is *curve fitting*, in which we define an abstract functional form (either through theory or observation) to be fit to the experimental data. The ubiquitous Python module `scipy` provides an implementation of curve fitting, which performs a least-squares optimization, minimizing the error between a fit curve and the experimental data by adjusting the parameters defining the functional form.

In this project, we were provided with three experimental datasets without context of the experiments from which they were generated. After deciding upon appropriate functional forms for each, we were tasked with fitting curves to each dataset, analyzing the accuracy of said curve fits, and deciphering plausible physical phenomena each represents. To do so required a healthy use of `matplotlib`, `scipy.optimize.curve_fit()`, `pandas`, and `numpy`. All computations were performed in Jupyter notebook.

2 Initial Analysis

Each of the three datasets was provided as a Comma-Separated Values file: `Dataset_A.csv`, `Dataset_B.csv`, and `Dataset_C.csv`. Each file has two columns, one labeled t and one labeled $f(t)$. Reading in each as a `pandas` DataFrame and splitting the two columns into separate 1-dimensional `numpy` arrays (named `X_t` and `X_ft` respectively, for $\texttt{X} \in \{\texttt{A}, \texttt{B}, \texttt{C}\}$), we performed an initial inspection of the datasets by plotting them with `matplotlib`. See Figure 4 and Figure 5 in Section 7 to see the Python code. In Figure 1, we have plotted each as individual points, and then connected said dots to create a continuous curve.

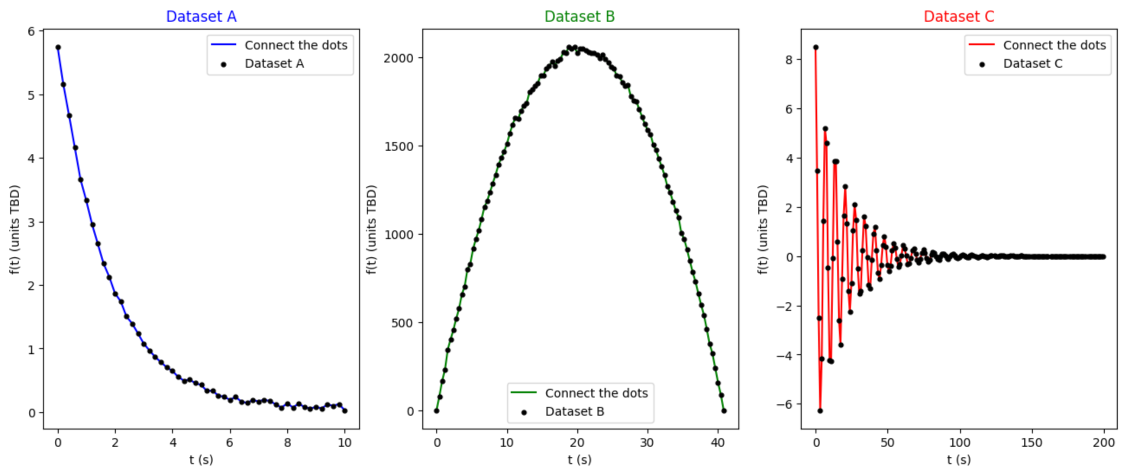


Figure 1: Initial visualization of three datasets.

By inspection, we assume Dataset A to be an exponential decay function, Dataset B to be a parabola, and Dataset C to be a damped oscillator (an exponentially decaying sinusoid). As such, we define three functions: $f_A(t) = ae^{-bt} + c$, $f_B(t) = at^2 + bt + c$, and $f_C(t) = be^{-at} \sin(ct + d)$. See Figure 6 in Section 7 for code defining these functions. In testing, we also analyzed a variant f_C using a linear combination of $\sin(ct)$ and $\cos(ct)$ rather than the phase shift–this is the source of the non-alphabetical variable ordering as that came first and this ordering was chosen to match–but we find the phase-shifted version to be more easily interpretable.

3 Curve Fitting

The process of curve fitting is then a pretty straightforward process, thanks to `scipy`. Calling the function `scipy.optimize.curve_fit(f_X, X_t, X_ft)` with parameters of the function to be fit, the time series data array, and the dependent variable data array returns two useful objects for us. First is a numpy array containing the fitting parameters a, b, c, and possibly d (one of which we will refer to generally as y) fit to the values that most closely model the experimental data. Second is a symmetric covariance matrix, of size $p \times p$ where p is the number of fitting parameters, in which the diagonal terms give the variance σ_y^2 (squared uncertainty) in each parameter. The off-diagonal terms give covariances $\sigma_{y_1}\sigma_{y_2}$, how correlated pairs of variables are. In each case, a good fit is indicated by low numbers; low variance means our parameters neatly fit to the data, and low covariance means the fitting parameters are independent of one another, which we expect to be the case. See Figure 7 in Section 7 for the code of curve fitting.

We'll start with the first return of the `curve_fit` function: the parameters themselves. After fitting, we found the following functional forms, with each parameter rounded to 3 decimal places:

$$\begin{aligned} f_A(t) &= 5.733e^{-0.566t} + 0.049 \\ f_B(t) &= -4.902t^2 + 200.062t + 3.995 \\ f_C(t) &= 7.813e^{-0.048t}\sin(0.923t + 1.543) \end{aligned}$$

Then, we can plug each array `X_t` into the corresponding functional form with the fitted parameters to graph our fitted model. Doing so for each functional form (including the later decided-against linear combination version of f_C) gives Figure 2. See Figure 8 in Section 7 for the code.

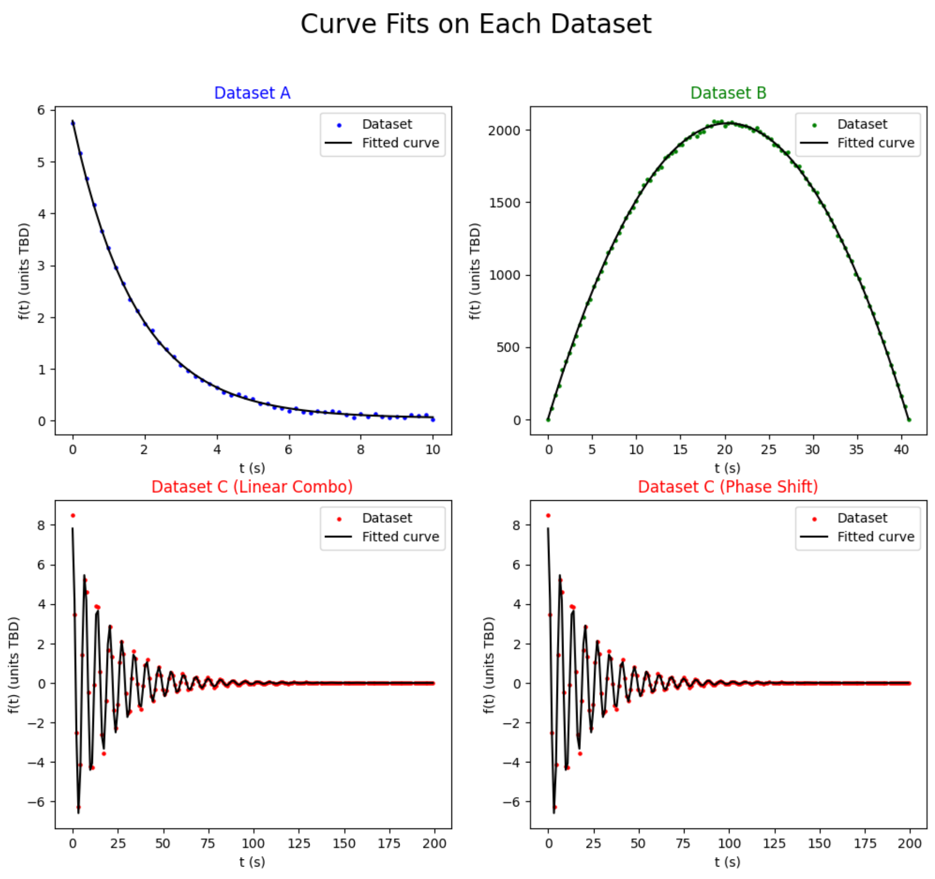
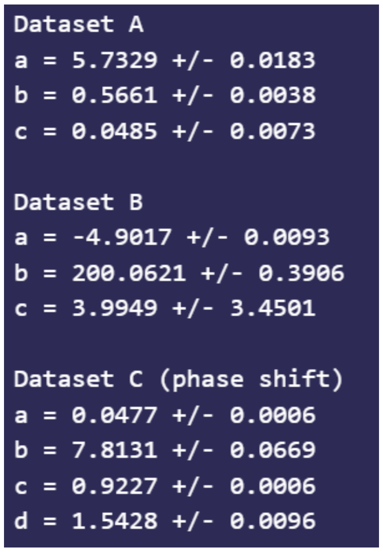


Figure 2: All three datasets with fitted curves overlaid.

4 Uncertainty Analysis

As we see, no datapoints are particularly distant from their respective curve, meaning the fits performed very well. Of course, the covariance matrix previously discussed provides us with quantitative tools to analyze beyond mere visual inspection. The full covariance matrices are given in in Figure 9 of Section 7, but as discussed prior the most important values are the variances along the diagonal. See Figure 3 below for the fit values of the parameters with uncertainty determined by taking the square root of each variance value.



```
Dataset A
a = 5.7329 +/- 0.0183
b = 0.5661 +/- 0.0038
c = 0.0485 +/- 0.0073

Dataset B
a = -4.9017 +/- 0.0093
b = 200.0621 +/- 0.3906
c = 3.9949 +/- 3.4501

Dataset C (phase shift)
a = 0.0477 +/- 0.0006
b = 7.8131 +/- 0.0669
c = 0.9227 +/- 0.0006
d = 1.5428 +/- 0.0096
```

Figure 3: All three datasets with fitted curves overlaid.

We see that nearly every parameter has uncertainty on the verge of negligibility, two or more orders of magnitude less than the value itself, with the notable exception of the c parameters of Dataset `A` and especially of Dataset `B`. Both of these represent $y-$intercepts, the value of $f_X(t)$ when $t = 0$, which is a notoriously sensitive parameter. Additionally, in Dataset `B` the values $f_B(t)$ are very far from 0 when t is far from zero, meaning slight differences in the other parameters can change c somewhat drastically. Indeed, we see in Figure 9, the off-diagonal terms of the covariance matrix for parameter c (along the third row and third column) are orders of magnitude greater than the other off-diagonal entries, meaning that there is some coupling between c and those other parameters. Overall though, we see that the variance is quite low and our curve fit rather accurately maps to the data.

5 Physical Interpretation

5.1 Dataset `A`

A common exponential decay in physics is the number of particles in a sample in a radioactive material. A typical way to write this would be $N(t) = N_0 e^{-\lambda t}$. Since our fit parameter has $c \approx 0$, it is fair for us to ignore that term and let $a = N_0$ and $b = \lambda$.

Now, it wouldn't make physical sense to start with 5.733 particles, so I'll take the units of $N_0 = a$ to be, say "mega-particles", where 5.773 megaparticles $= 5.773 \times 10^6$ particles, rounding that down to the nearest integer. $\lambda = b$ is then the decay constant, which has physical units of inverse time.

$$f_A(t) = (5732880 \text{ particles})\, e^{-\left(0.566\,\mathrm{s}^{-1}\right)t}$$

5.2 Dataset `B`

A common parabolic shape like this in physics would be the trajectory of a projectile $y(t) = -\frac{1}{2}gt^2 + v_0t + y_0$. So here we have $a = -\frac{1}{2}g$, with units of $\frac{\mathrm{m}}{\mathrm{s}^2}$, $b = v_0$ with units of $\frac{\mathrm{m}}{\mathrm{s}}$, and $c = y_0$ with units of m. Calculating g from our a fit we find $g = 9.803\,\frac{\mathrm{m}}{\mathrm{s}^2}$, and taking the other coefficients directly, we find the following curve fit function:

$$f_B(t) = \left(9.803\,\tfrac{\mathrm{m}}{\mathrm{s}^2}\right)t^2 + \left(200.062\,\tfrac{\mathrm{m}}{\mathrm{s}}\right)t + (3.995\,\mathrm{m})$$

5.3 Dataset `C`

Plenty of things could be an underdamped oscillator. I'll go with the prototypical example of a damped mass spring. The physical meaning of the parameters takes a fair bit more work here, so I'll actually do the derivation. Say we have a mass m (in kg) oscillating on a spring with spring constant k (in $\frac{\mathrm{N}}{\mathrm{m}}$ or $\frac{\mathrm{kg}}{\mathrm{s}^2}$), and damping constant β (in $\frac{\mathrm{kg}}{\mathrm{s}}$ or $\frac{\mathrm{N\,s}}{\mathrm{m}}$)

$$\begin{aligned} &m\ddot{x} + \beta\dot{x} + kx = 0 \\ &\ddot{x} + \frac{\beta}{m}\dot{x} + \frac{k}{m}x = 0 \\ &\ddot{x} + 2\gamma\dot{x} + \omega_0^2 x = 0 \qquad \text{where } \gamma = \frac{\beta}{2m} \text{ and } \omega_0 = \sqrt{\frac{k}{m}} \end{aligned}$$

This is underdamped when the discriminant of the characteristic equation is negative. That is, if $\gamma^2 < \omega_0^2$, and the solution involves the damped frequency $\omega_d = \sqrt{\omega_0^2 - \gamma^2}$. That solution is this:

$$x(t) = Be^{-\gamma t}\sin(\omega_d t + \phi)$$

What we notice is that coefficient out front (our b) is the maximum amplitude of the oscillation for $t = 0$. So that would be $b = B = 7.813\,\mathrm{m}$. The exponential decay factor is $a = \gamma = 0.048\,\mathrm{s}^{-1}$, and the damped frequency is $c = \omega_d = 0.923\,\frac{\mathrm{rad}}{\mathrm{s}}$. Now, what's interesting about the phase is that it is *very* close to $\frac{\pi}{2}$. We see that $2d \approx 3.09$, which means d itself is only a degree or 2 away from being $\frac{\pi}{2}$. We can use that to transform it into a cos function, since $\sin(\omega t + \frac{\pi}{2}) = \cos(\omega t)$. This brings our final functional form to the following:

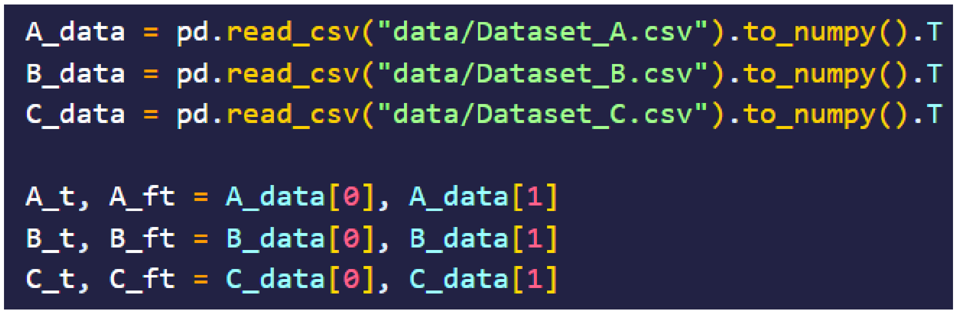
$$f_C(t) = (7.813\,\mathrm{m})\,e^{-\left(0.048\,\mathrm{s}^{-1}\right)t}\cos\left(\left(0.923\,\tfrac{\mathrm{rad}}{\mathrm{s}}\right)t\right)$$

6 Conclusion

Curve fitting is a computational method by which a mathematical function is fit to an experimental dataset such that the squared distance between the curve and the data is as close to zero as possible. We applied the `scipy` implementation of this algorithm to determine the physical process measured in each dataset. In general, each fit performed very well–closely hugging the points with rather low variance–meaning our experimental data was very neat and our fitting was rather precise. Analyzing the results of eachd we were able to ascribe physical meaning to each fitted parameter. As such, we consider the Dataset Mystery to be succinctly resolved.

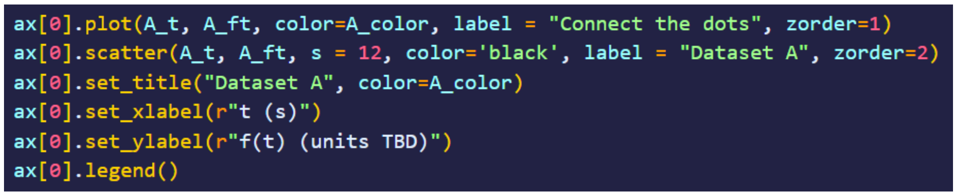
7 Appendix: Code

The full code is provided at `https://github.com/josephtemple/JT_PHYS4023` under the `/Project 1/` directory. Indeed, one could not locate the directory of this PDF wtithout passing the code along the way blocks of code and certain outputs are also provided below.



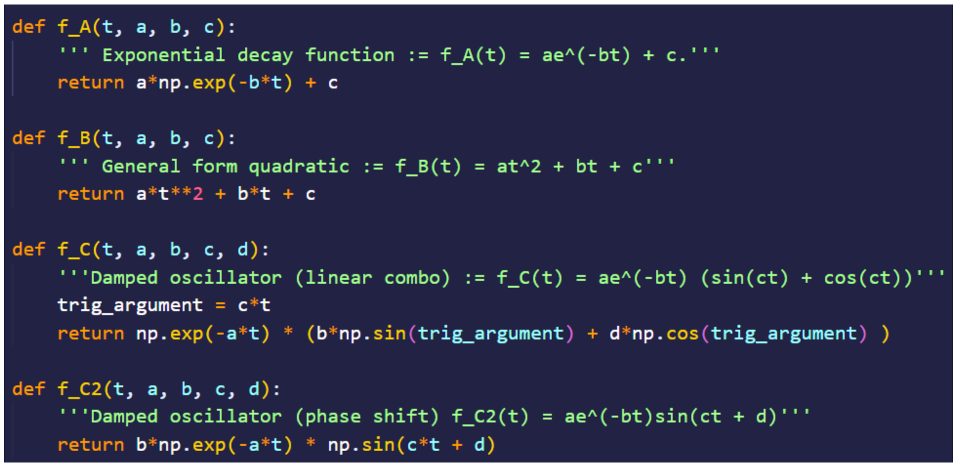
```python
A_data = pd.read_csv("data/Dataset_A.csv").to_numpy().T
B_data = pd.read_csv("data/Dataset_B.csv").to_numpy().T
C_data = pd.read_csv("data/Dataset_C.csv").to_numpy().T

A_t, A_ft = A_data[0], A_data[1]
B_t, B_ft = B_data[0], B_data[1]
C_t, C_ft = C_data[0], C_data[1]
```

Figure 4: Code to read `csv` files into pandas dataframes and convert them to numpy arrays.



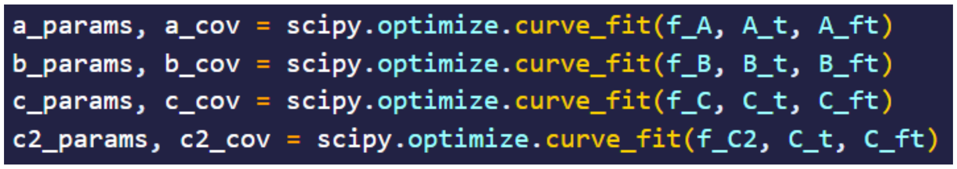
```python
ax[0].plot(A_t, A_ft, color=A_color, label = "Connect the dots", zorder=1)
ax[0].scatter(A_t, A_ft, s = 12, color='black', label = "Dataset A", zorder=2)
ax[0].set_title("Dataset A", color=A_color)
ax[0].set_xlabel(r"t (s)")
ax[0].set_ylabel(r"f(t) (units TBD)")
ax[0].legend()
```

Figure 5: Initial `matplotlib` code for visualizing raw datasets.



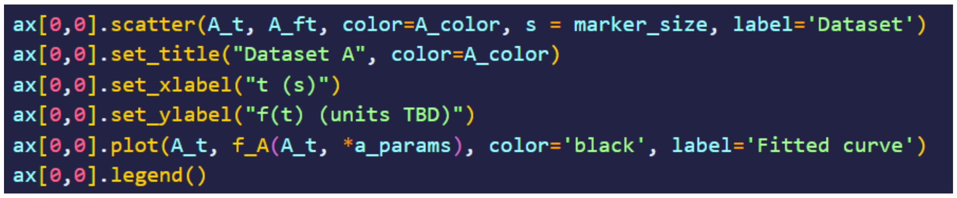
```python
def f_A(t, a, b, c):
    ''' Exponential decay function := f_A(t) = ae^(-bt) + c.'''
    return a*np.exp(-b*t) + c

def f_B(t, a, b, c):
    ''' General form quadratic := f_B(t) = at^2 + bt + c'''
    return a*t**2 + b*t + c

def f_C(t, a, b, c, d):
    '''Damped oscillator (linear combo) := f_C(t) = ae^(-bt) (sin(ct) + cos(ct))'''
    trig_argument = c*t
    return np.exp(-a*t) * (b*np.sin(trig_argument) + d*np.cos(trig_argument) )

def f_C2(t, a, b, c, d):
    '''Damped oscillator (phase shift) f_C2(t) = ae^(-bt)sin(ct + d)'''
    return b*np.exp(-a*t) * np.sin(c*t + d)
```

Figure 6: Python functions defining mathematical models of each dataset.



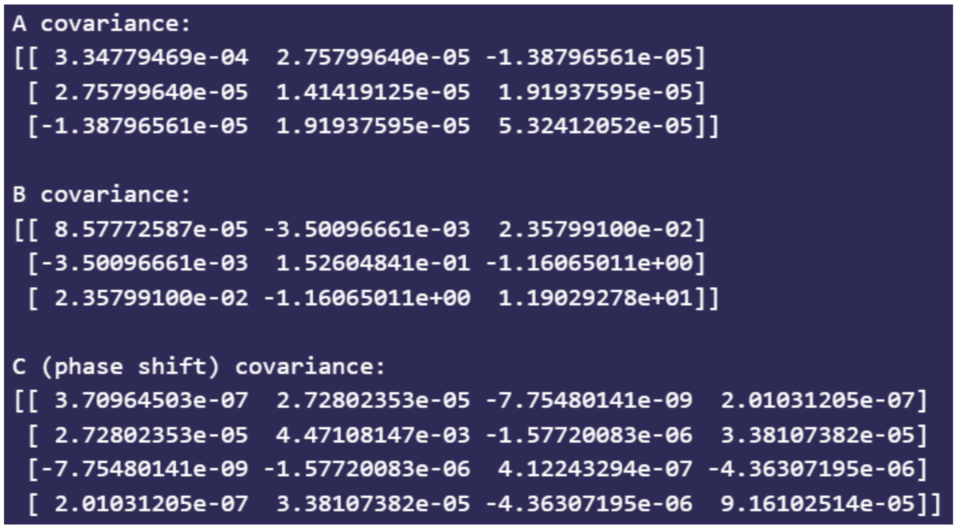
```python
a_params, a_cov = scipy.optimize.curve_fit(f_A, A_t, A_ft)
b_params, b_cov = scipy.optimize.curve_fit(f_B, B_t, B_ft)
c_params, c_cov = scipy.optimize.curve_fit(f_C, C_t, C_ft)
c2_params, c2_cov = scipy.optimize.curve_fit(f_C2, C_t, C_ft)
```

Figure 7: Curve fitting to define parameter arrays and covariance matrices.



```python
ax[0,0].scatter(A_t, A_ft, color=A_color, s = marker_size, label='Dataset')
ax[0,0].set_title("Dataset A", color=A_color)
ax[0,0].set_xlabel("t (s)")
ax[0,0].set_ylabel("f(t) (units TBD)")
ax[0,0].plot(A_t, f_A(A_t, *a_params), color='black', label='Fitted curve')
ax[0,0].legend()
```

Figure 8: `matplotlib` code for visualizing curve fits overlaid with raw data.



```
A covariance:
[[ 3.34779469e-04  2.75799640e-05 -1.38796561e-05]
 [ 2.75799640e-05  1.41419125e-05  1.91937595e-05]
 [-1.38796561e-05  1.91937595e-05  5.32412052e-05]]

B covariance:
[[ 8.57772587e-05 -3.50096661e-03  2.35799100e-02]
 [-3.50096661e-03  1.52604841e-01 -1.16065011e+00]
 [ 2.35799100e-02 -1.16065011e+00  1.19029278e+01]]

C (phase shift) covariance:
[[ 3.70964503e-07  2.72802353e-05 -7.75480141e-09  2.01031205e-07]
 [ 2.72802353e-05  4.47108147e-03 -1.57720083e-06  3.38107382e-05]
 [-7.75480141e-09 -1.57720083e-06  4.12243294e-07 -4.36307195e-06]
 [ 2.01031205e-07  3.38107382e-05 -4.36307195e-06  9.16102514e-05]]
```

Figure 9: Covariance matrix of primary three curve fits.