Estimation Labwork

Master ATSI

(SU Sichen)

Estimation

January 18, 2024

Contents

1 A ball falling into a viscous fluid

We wish to characterize the friction experienced by a steel ball when it falls into a test tube filled with a viscous liquid. For this purpose, we will consider the following experimental device

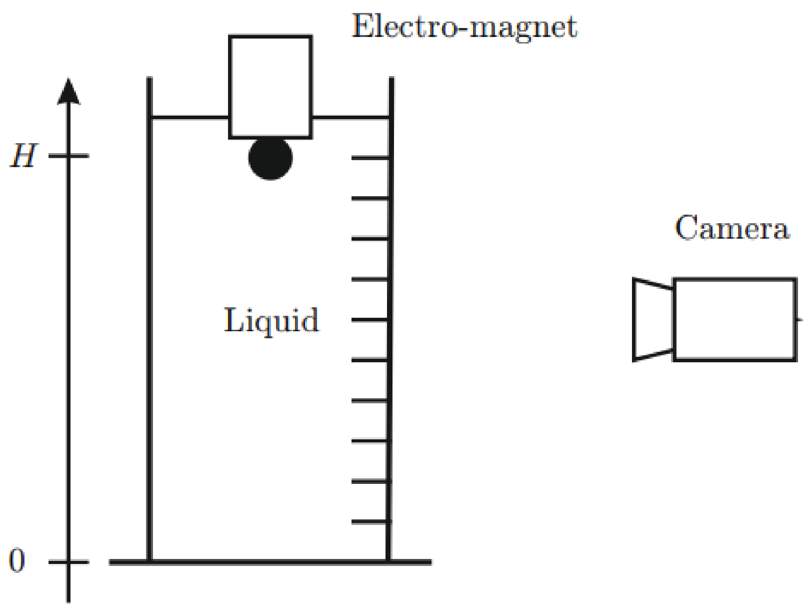

Figure 1: Experimental Device

A steel ball of density $\rho_b = 7870$ kg/m^3 and radius $r = 6.7$ mm is released from a height of $H = 1$ meter in glycerin with a density of $\rho_l = 1260$ kg/m^3. The fall is initiated by turning off an electromagnet, and the ball's fall is recorded at a rate of 1000 frames per second. This setup allows for the measurement of the ball's changing height in the liquid at specified time intervals. The resulting data consists of a vector of height measurements $\mathbf{y} = (h(t_0), \ldots, h(t_N))^T$ and a corresponding vector of time instances $\mathbf{t} = (t_0, \ldots, t_N)^T$. The experiment assumes that the tube is wide enough so that the ball only experiences forces due to its weight, buoyancy (Archimedes' thrust), and a viscous friction where the force is proportional to the ball's speed raised to the power n, with $k > 0$ and $n > 0$ being constants to be determined.

Problem 1

Give a differential equation satisfied by the height $h(t)$ of the ball in the test tube as a function of time. Also, give the initial conditions.

The parameters ρ_l, ρ_b, and r being known, an estimation of $\mathbf{p} = (k, n)^T$ in the least squares sense is sought. We consider that the acceleration of gravity is $g = 9.81$ m/s^2.

To derive a differential equation for the height $h(t)$ of a steel ball falling through a viscous fluid like glycerin, we can start with the forces acting on the ball. These forces are gravity, buoyancy, and viscous drag.

a) Gravity force: $F_g = m \cdot g$, where m is the mass of the ball and g is the acceleration due to gravity (9.81 m/s^2).

b) Buoyancy force: $F_b = V \cdot \rho_l \cdot g$, where V is the volume of the ball, and ρ_l is the density of the fluid (glycerin).

c) Viscous drag: Generally modeled as $F_d = k \cdot \left|\frac{dh}{dt}\right|^n$, where $\frac{dh}{dt}$ is the velocity of the ball, n is the constant to be determined, and k is a constant to be determined experimentally.

So based on these equations, we can build the following equation:

$$m\frac{d^2h}{dt^2} = mg - V\rho_l g - k\left|\frac{dh}{dt}\right|^n$$

The initial conditions for the system are:

$$h(0) = H \tag{1}$$

$$\left.\frac{dh}{dt}\right|_{t=0} = 0 \tag{2}$$

Problem 2

Show that the dynamics of the ball can be written as

$$\frac{dx}{dt} = f(x, p) \tag{3}$$

with $x(0) = x_0$.

To express the dynamics of the ball in the form of a state-space representation, we would typically define a state vector x that encapsulates all the necessary variables to describe the system at any given time. For the falling ball, the state vector would include the position and the velocity of the ball:

$$x(t) = \begin{bmatrix} h(t) \\ \frac{dh}{dt} \end{bmatrix}$$

The time derivative of the state vector x, which is $\frac{dx}{dt}$, would then be:

$$\frac{dx}{dt} = \begin{bmatrix} \frac{dh}{dt} \\ \frac{d^2h}{dt^2} \end{bmatrix}$$

We already have the second-order differential equation for $\frac{d^2h}{dt^2}$, which includes the forces acting on the ball and can be expressed in terms of the state vector x and parameters p (which includes k and n):

$$\frac{d^2h}{dt^2} = g - \frac{\rho_l}{\rho_b}g - \frac{k}{m}\left|\frac{dh}{dt}\right|^n$$

Therefore, the function $f(x, p)$ is given by:

$$f(x,p) = \begin{bmatrix} x_2 \\ g - \frac{\rho_l}{\rho_b}g - \frac{k}{\left(\frac{4}{3}\pi r^3 \rho_b\right)}|x_2|^n \end{bmatrix}$$

and where $p = (k, n)^T$, $x_2 = \frac{dh}{dt}$.

The initial condition for the state vector x at time $t = 0$ is given by:

$$x(0) = \begin{bmatrix} H \\ 0 \end{bmatrix}$$

Problem 3

Give the expression of the least-squares cost function $c(p)$ to minimize to obtain

$$\hat{p}_{MC} = \arg\min_{p} c(p),$$

i.e., the least squares estimate of p from the measurements stored in y.

Using the least squares estimation to estimate the parameters **P**, we need to build an estimator to estimate $\hat{y}$.

Given the state-space representation:

$$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), \mathbf{p})$$

$$\mathbf{x}(0) = \mathbf{x}_0$$

With Euler's method, you discretize time into small steps of size Δt and then estimate the state at the next step $\mathbf{x}(t + \Delta t)$ from the state at the current step $\mathbf{x}(t)$ using the equation:

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \Delta t \cdot f(\mathbf{x}(t), \mathbf{p})$$

This can be expanded for each component of the state vector. For our system:

$$\hat{y} = x_1(t + \Delta t) = x_1(t) - \Delta t \cdot x_2(t)$$

$$\hat{y}' = x_2(t + \Delta t) = x_2(t) + \Delta t \cdot \left(g - \frac{\rho_l}{\rho_b} g - \frac{k}{m} |x_2(t)|^n\right)$$

Here, $x_1(t)$ corresponds to the height $h(t)$ and $x_2(t)$ corresponds to the velocity

So

$$\hat{y}_{t+1} = \hat{y}_t - \Delta t \cdot \hat{y}'_t$$

$$\hat{y}'_{t+1} = \hat{y}'_t + \Delta t \cdot \left(g - \frac{\rho_l}{\rho_b} g - \frac{k}{m} |\hat{y}'_t|^n\right)$$

We can build our cost function in a time sequence based on the predicted value $\hat{y}$ and measurement.

$$c_t = (y_i - \hat{y}_t)^2$$

We have N nombre data so we sum our cost function each time and then find the best **p** that we get the minimum cost function sum.

$$c(p) = \sum_{1}^{N} (y_i - \hat{y}_i(p))^2$$

Problem 4

Write a function to evaluate the dynamics of the ball.

Based on the Dynamic equation, I built our function to evaluate the dynamics of the ball

```python
def dynball(x,t, p, const=constante):
    g = const.g
    rho_l = const.rho_l
    rho_b = const.rho_b
    r = const.r

    k = p[0]  # Viscous drag coefficient
    n = p[1]  # Power of velocity in the drag equation

    h = x[0]  # Ball's height
    v = x[1]  # Ball's velocity

    V = (4/3) * np.pi * r**3
    m = V * rho_b

    dhdt = v
    dvdt = -g + (rho_l * g / rho_b) + (k / m) * abs(v)**n

    dxdt = np.array([dhdt, dvdt])
    return dxdt
```

2 Data visualization

Problem 5

plot the evolution of the height of the ball as a function of the time

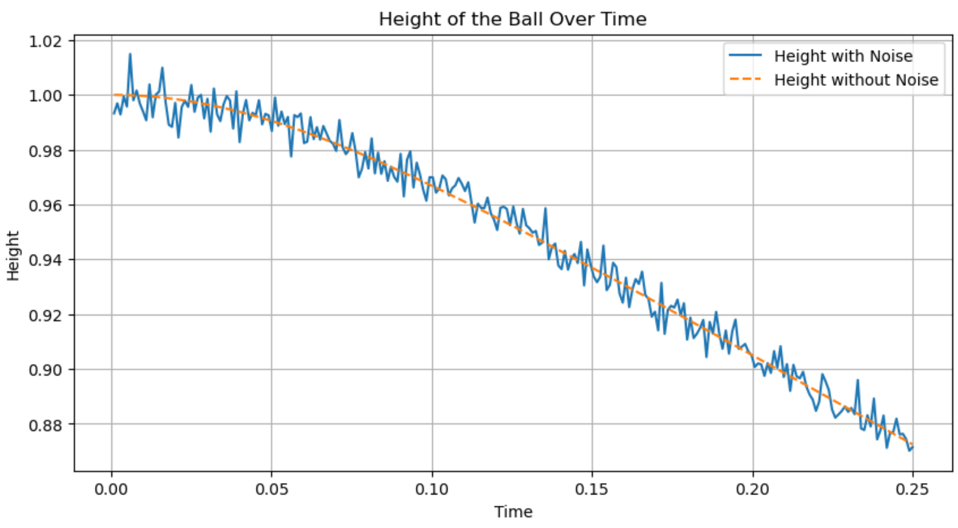

Figure 2: Data without Noise and with noise

3 Estimation using gradient descent

Problem 6

Construct a function taking as inputs p, y, and t allowing to calculate the value of the cost function $c(p)$ as well as its gradient obtained by finite difference. We will assume that all non-estimated parameters are stored in global variables. A possible structure is [c,g]=critgrad(p,y,t)

In this section, the Finite Difference method is employed, introducing a disturbance into our cost function to facilitate the computation of the gradient of the cost function.

```python
def critgrad(p, y, t):
    x0 = [constante.H, .0]  # Initial condition: starting at the first height value with zero velocity
    sol = odeint(dynball, x0, t, args=(p, constante))
    y_hat = sol[:, 0]  # Extract the height predictions from the solution

        # Calculate the cost function as the sum of squared differences
    c = np.sum((y - y_hat)**2)

    # Calculate the gradient of the cost function using finite difference
    delta = 1e-5  # A small perturbation for finite difference
    grad = np.zeros_like(p)
    for i in range(len(p)):
        p_delta = np.copy(p)
        p_delta[i] += delta
        sol_delta = odeint(dynball, x0, t, args=(p_delta, constante))
        y_hat_delta = sol_delta[:, 0]
        c_delta = np.sum((y - y_hat_delta)**2)
        grad[i] = (c_delta - c) / delta

    return c, grad
```

Problem 7

Build a program performing the estimation of p in the least squares sense using the gradient method to minimize the criterion.

Based on the critical function, we compute the gradient, following which a fixed step size is chosen to optimally determine the $\mathscr{P}$ parameter that minimizes the cost function.

```python
def gradient_descent(p_init, y, t, const, alpha, num_iterations):
    p = p_init.copy()
    p_history = []  # List to store the history of p values
    for i in range(num_iterations):
        c, grad = critgrad(p, y, t)
        p -= alpha * grad  # Update the parameter vector
        p_history.append(p.copy())  # Store the current value of p
        print(f"Iteration {i+1}: Cost = {c}, Parameters = {p}")
    return np.array(p_history)  # Return the history of p values as an array

# Run gradient descent and get the history of p values
p_init = np.array([1.0, 1.0])  # Initial parameter guess
alpha = 0.01  # Learning rate
num_iterations = 800  # Number of iterations
height = data_noise

```

```python
# Perform the gradient descent to estimate the parameters
p_history = gradient_descent(p_init, height, time, constante, alpha, num_iterations)
```

Problem 8

Plot the evolution of the estimate of p as a function of the iteration of the gradient descent.

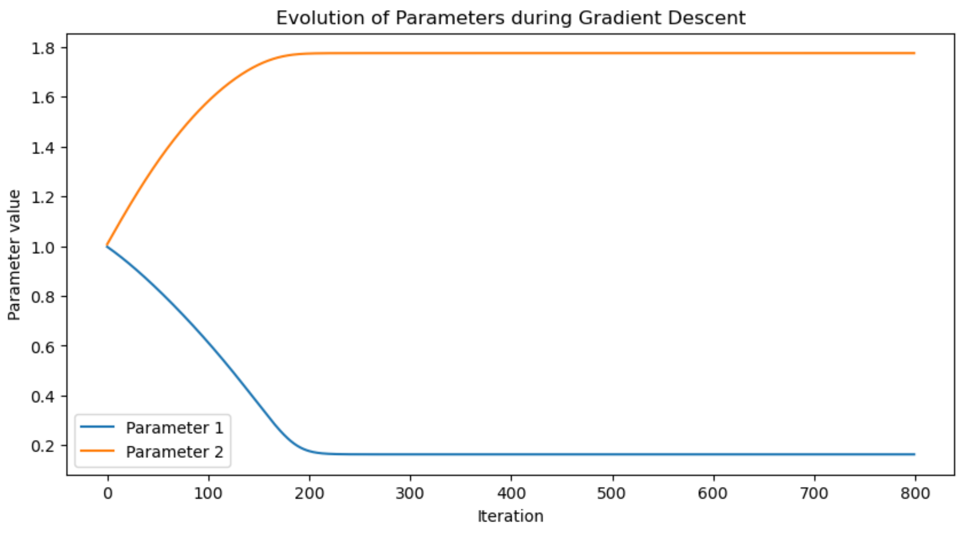

Figure 3: Evolution of the estimate of p, $\hat{p} = [0.16364295, 1.77462433]$

Problem 9

Compare the evolution with the time of the height of the ball simulated using the estimated value of p and that measured.

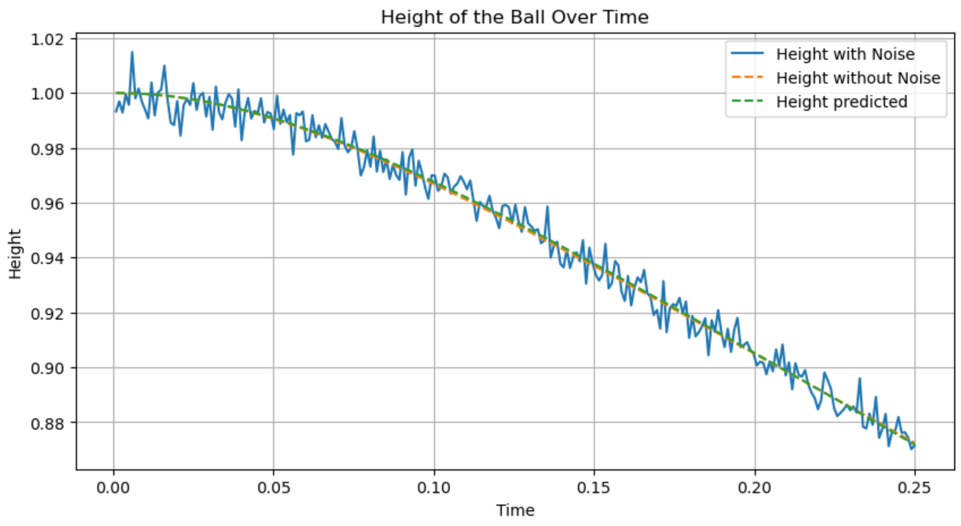

Figure 4: Evolution of the height of the ball simulated

Based on the analysis of diagram 4, it can be seen that the curve representing the simulated height of the ball is very close to the direction of descent of the measurement. Furthermore, the simulated curves show a good fit to the actual noise-free measurement curves. These observations suggest that the estimated parameters work well. We then aim to calculate the confidence interval for these parameters, which provides a statistical measure of their precision and stability.

Problem 10

Characterize the estimation uncertainty using the method that appears the most appropriate. Justify carefully your choice.

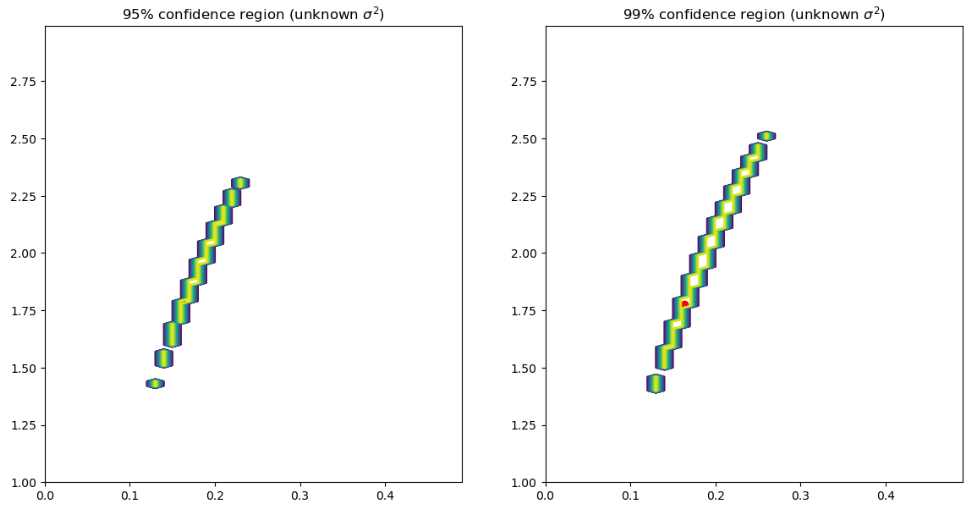

Figure 5: 95% confident and 99% confident interval and $\hat{p}$ estimated

In this section, we employ the method of confidence intervals to accurately characterize the uncertainty associated with our parameter estimations. This methodology entails evaluating the estimated parameters to ascertain if they reside within the predefined confidence intervals. Such an approach aims to provide a quantifiable assessment of the estimates' reliability and precision.

At the outset of our analysis, we hypothesize that the residuals – the discrepancies between the measured and predicted values – conform to a normal distribution, denoted as $\mathcal{N}(0, \sigma^2)$.

Subsequently, we construct the projection matrix $\Pi(p)$ for the estimated output $\hat{y}(p)$ using the Jacobian matrix, as follows:

$$\Pi(p) = \left(\frac{\partial y_m(p)}{\partial p^T}\right)\left(\left(\frac{\partial y_m(p)}{\partial p^T}\right)^T\left(\frac{\partial y_m(p)}{\partial p^T}\right)\right)^{-1}\left(\frac{\partial y_m(p)}{\partial p^T}\right)^T$$

Employing the orthogonal projection matrix $\Pi(p)$, we deduce that the projected residuals conform to a chi-squared distribution, taking into account that the dimension of the parameter space is $dim(p) = 2$, and the dimension of the error vector is $dim(e) = 250$:

$$e^T(p^*)\Pi(p^*)e(p^*) \sim \sigma^2\chi^2(n_p)$$

$$e^T(p^*)(I - \Pi(p^*))e(p^*) \sim \sigma^2\chi^2(n_t - n_p),$$

where n_p represents the number of parameters and n_t denotes the number of measurements. This alignment with the χ^2 distribution facilitates the derivation of confidence intervals for our estimated parameters.

Based on the two independent distributions and the Fisher-Snedecor law, we use the ratio of two independent χ^2 distributed random variables to obtain a variable that follows the Fisher distribution:

$$\frac{e^T(p^*)\Pi(p^*)e(p^*)}{e^T(p^*)(I-\Pi(p^*))e(p^*)}\frac{n_t-n_p}{n_p} \sim F(n_p, n_t-n_p)$$

Subsequently, we construct our confidence intervals by first determining the critical values corresponding to the 95% and 99% confidence levels from the F-distribution. We proceed to identify the parameter estimates, denoted by $\hat{p}$, where the cumulative distribution function of the F-distribution is less than these critical values for the respective confidence levels.

Upon delineating our confidence region, we graphically represent our estimated values within this region, (red point). It is then ascertained that the estimated value $\hat{p}$ falls within the bounds of the confidence region, affirming the precision and reliability of our estimates.

4 Using sensitivity functions

Problem 11

Show that the gradient of $c(\mathbf{p})$ may be written as

$$g(\mathbf{p}) = \begin{pmatrix} \frac{\partial c(\mathbf{p})}{\partial p_1} \\ \frac{\partial c(\mathbf{p})}{\partial p_2} \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{N-1}(h_m(t_i,\mathbf{p})-y(t_i))s_1^{p_1}(t_i) \\ \sum_{i=0}^{N-1}(h_m(t_i,\mathbf{p})-y(t_i))s_1^{p_2}(t_i) \end{pmatrix}$$

where $h_m(t_i,\mathbf{p})$ is the model output for a given value $\mathbf{p}$ of the vector of parameters obtained from numerical integration and

$$s_i^{p_j}(t) = \frac{\partial x_i(t)}{\partial p_j}$$

is the sensitivity function of the state component x_i, $i = 1,2$, with respect to the component p_j, $j = 1,2$ of the vector of parameters.

In our dynamic model, the state space has 2 components $x(t) = \begin{bmatrix} h(t) \\ \frac{dh}{dt} \end{bmatrix}$, In this question, we show the sensitivity function for the first elements based on the cost function $C(p)$

$$c(p) = \sum_{1}^{N}(y(t) - \hat{y}_i(t,p))^2$$

Start with the definition of the gradient of the cost function $c(\mathbf{p})$:

$$g(\mathbf{p}) = \begin{bmatrix} \frac{\partial c(\mathbf{p})}{\partial p_1} \\ \frac{\partial c(\mathbf{p})}{\partial p_2} \end{bmatrix}$$

Write the cost function $c(\mathbf{p})$ as a sum of squared errors:

$$c(\mathbf{p}) = \sum_{i=0}^{N-1}(y(t_i) - h_m(t_i,\mathbf{p}))^2$$

Take the partial derivative of $c(\mathbf{p})$ with respect to p_1 and p_2:

$$\frac{\partial c(\mathbf{p})}{\partial p_j} = \sum_{i=0}^{N-1} -2(y(t_i) - h_m(t_i,\mathbf{p}))\left(-\frac{\partial h_m(t_i,\mathbf{p})}{\partial p_j}\right)$$

Recognize that the sensitivity function $s_i^{p_j}(t) = \frac{\partial x_i(t)}{\partial p_j}$ can be related to the partial derivative of the model output $h_m(t_i, \mathbf{p})$ with respect to p_j:

$$\frac{\partial h_m(t_i, \mathbf{p})}{\partial p_j} = s_i^{p_j}(t_i)$$

Substitute the sensitivity functions into the expression for the partial derivatives:

$$\frac{\partial c(\mathbf{p})}{\partial p_j} = \sum_{i=0}^{N-1} -2(h_m(t_i, \mathbf{p}) - y(t_i))s_i^{p_j}(t_i)$$

Incorporate these into the gradient:

$$g(\mathbf{p}) = \begin{bmatrix} \sum_{i=0}^{N-1} -2(h_m(t_i, \mathbf{p}) - y(t_i))s_1^{p_1}(t_i) \\ \sum_{i=0}^{N-1} -2(h_m(t_i, \mathbf{p}) - y(t_i))s_1^{p_2}(t_i) \end{bmatrix}$$

Problem 12

12,13 Write a differential system whose solution gives the evolution of $s_1^{p_1}$ and $s_2^{p_1}$ as a function of time. Specify the initial conditions. Do the same for $s_2^{p_2}$, $s_2^{p_2}$.

To build the revolution of sensitive functions, we need use the system dynamic function $f(x, p)$

$$x = \begin{bmatrix} h \\ \frac{dh}{dt} \end{bmatrix}$$

And dynamic function $f(x, p)$ is:

$$f(x,p) = \begin{bmatrix} \dot{h} \\ g - \frac{p_l}{p_\rho}g - \frac{k}{m}|\dot{h}|^n \end{bmatrix}$$

Let's denote $s_h^k(t) = \frac{\partial h}{\partial k}$ and $s_{\dot{h}}^k(t) = \frac{\partial \dot{h}}{\partial k}$ as the sensitivities of the height and its rate of change with respect to the drag coefficient k, and similarly for n.

To compute these sensitivities, we need to set up the following differential equations based on the original state equations:

$$\frac{d}{dt}s_h^k(t) = \frac{d}{dt}\frac{\partial h}{\partial k} = \frac{\partial \dot{h}}{\partial k} = s_{\dot{h}}^k(t)$$

$$\frac{d}{dt}s_{\dot{h}}^k(t) = -\frac{1}{m}|\dot{h}|^n$$

$$\frac{d}{dt}s_h^n(t) = s_{\dot{h}}^n(t)$$

$$\frac{d}{dt}s_{\dot{h}}^n(t) = -\frac{k}{m}ln(|\dot{h}|)|\dot{h}|^n$$

The initial conditions for the sensitivities are usually zero, assuming that at time $t = 0$, the system's output is not sensitive to the parameters:

$$s_h^k(0) = s_{\dot{h}}^k(0) = 0$$

$$s_h^n(0) = s_{\dot{h}}^n(0) = 0$$

Problem 13

The systems describing the evolution of the sensitivity functions can be grouped into a single large system of nine first-order differential equations. Modify the dynball.m file to obtain the time derivatives of the different sensitivity functions in addition to $\frac{dx_1}{dt}$ and $\frac{dx_2}{dt}$. The syntax of this function should be

```python
def dynball_new(xe, t, p, const=constante):
    # Unpack constants
    g = const.g
    rho_l = const.rho_l
    rho_b = const.rho_b
    r = const.r

    # Unpack parameters and state variables
    k = p[0]  # Viscous drag coefficient
    n = p[1]  # Power of velocity in the drag equation

    h, v, s_k_h, s_k_v, s_n_h, s_n_v= xe

    # Volume and mass of the ball
    V = (4/3) * np.pi * r**3
    m = V * rho_b

    # Dynamics of the system
    dhdt = v
    dvdt = -g + (rho_l * g / rho_b) +(k / m) * abs(dhdt)**n

    # Sensitivity equations
    ds_k_h_dt = s_k_v
    ds_k_v_dt = (-1/m) *abs(v)**(n)

    # Assuming x2 is the velocity 'v', the ln term needs to handle v = 0
    if v != .0:
        ln_term = math.log(abs(dhdt))
    else:
        ln_term = 0.0
    ds_n_h_dt = s_n_v
    ds_n_v_dt = (-k/m) * ln_term * abs(dhdt)**(n)

    # Collect derivatives
    dxedt = [dhdt, dvdt, ds_k_h_dt, ds_k_v_dt, ds_n_h_dt, ds_n_v_dt]
    return np.array(dxedt)
```

Problem 14

What is the advantage of using sensitivity functions compared to the finite difference method?

Analytical Accuracy: The Sensitivity functions, when derived analytically, provide exact expressions for how to output of the system responds to infinitesimal changes in parameters. This is more accurate than finite difference, which provides an approximate solution that depends on the size of the perturbation used.

Computational Efficiency:

Computing sensitivity functions through analytical methods or differentiation can be more efficient than finite differences. with finite differences, we need to perturb each parameter and re-evaluate the entire system to estimate each derivative; which can be computationally expensive

Better fir Optimization:

The sensitivity functions are used to approximate the Hessian matrix and build the Jacobin matrix. This can lead to faster convergence and better stability than using finite differences for the same purpose.

Problem 15

Perform a new estimation by gradient descent when gradients are evaluated using sensitivity functions and plot the evolution of the estimate of p as a function of the iteration of the gradient descent.

Based on the sensitivity function and the residuals, we can obtain the gradient of the cost function

$$g(\mathbf{p}) = \begin{bmatrix} \sum_{i=0}^{N-1} -2(h_m(t_i,\mathbf{p}) - y(t_i))s_1^{p_1}(t_i) \\ \sum_{i=0}^{N-1} -2(h_m(t_i,\mathbf{p}) - y(t_i))s_1^{p_2}(t_i) \end{bmatrix}$$

After building the gradient descent algo, we can plot the evolution of the estimate of p, and we also get the estimated value $\hat{p} = [0.16712946, 1.80654545]$.

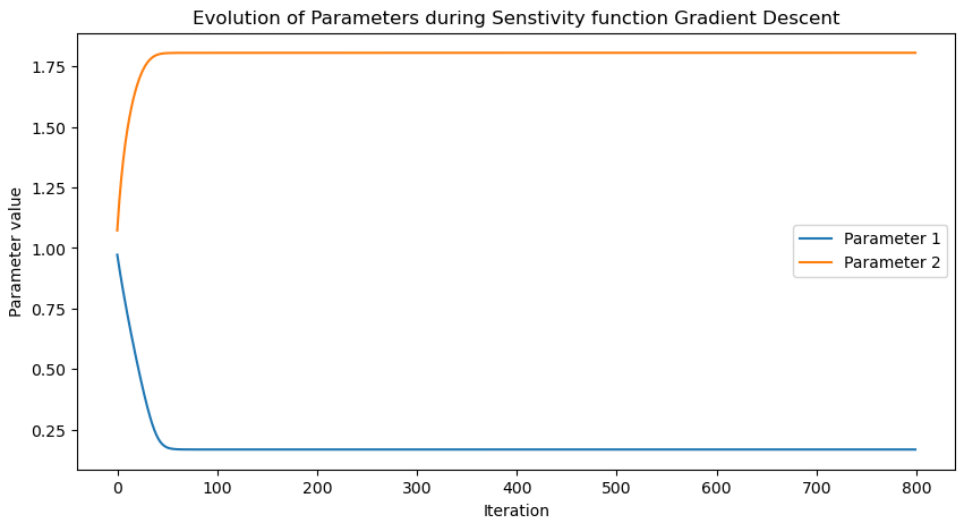

Figure 6: Gradient Descent based on the sensitivity function

Then we import the estimated parameters into the dynamic model, we can see that the predicted value $\hat{y}$ is very close to the real value of y without noise.

5 Using Gauss-Newton's approach

Problem 16

Perform an estimation of the parameters using Gauss-Newton's method.

By involving the Taylor expansion of the cost function $c(p)$ around p, we can approximate the quadratic

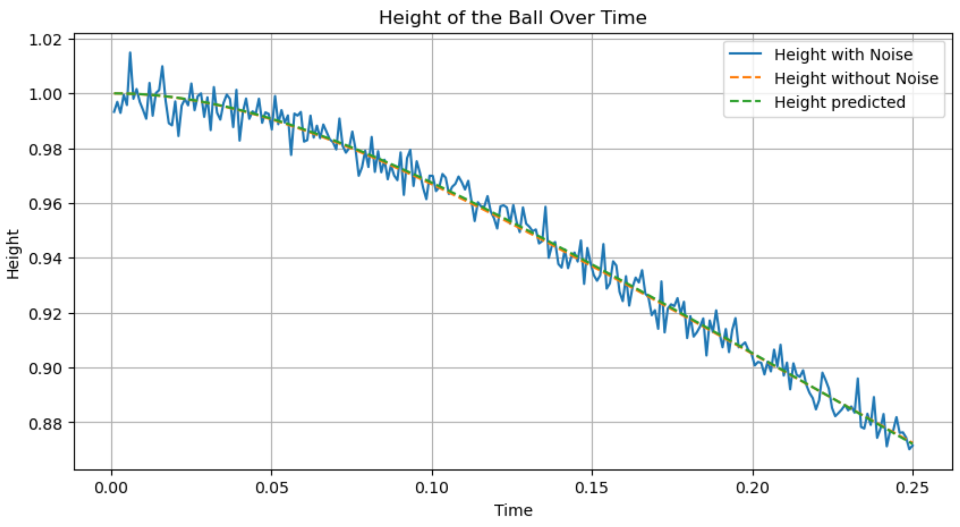

Figure 7: Comparing with measurement and non-noise value

cost function in the following form:

$$C(p + \Delta p) = f(x) + \Delta p^T J^T f + \frac{1}{2}\Delta p^T J^T J \Delta p. \tag{4}$$

To minimize the cost function, we take the first-order derivative, yielding:

$$\frac{\partial}{\partial \Delta p} C(p + \Delta p) = J^T f + J^T J \Delta p. \tag{5}$$

Then, we find that

$$\Delta p = -(J^T J)^{-1} J^T f, \tag{6}$$

where $J^T J$ is an approximation of the Hessian H, and J is the Jacobian matrix. This Jacobian can be constructed using the sensitivity function values to build the Jacobian matrix for h, the height of the ball.

```python
def crit_Gauss_NewTon_sensitive(p, y, t):
    x0 = [constante.H, 0.0, .0, .0,.0,0]
    sol = odeint(dynball_new, x0, t, args=(p, constante))
    y_hat = sol[:, 0]  # Extract the height predictions from the solution

    # Calculate the cost function as the sum of squared differences
    c = 0.5*np.sum((y - y_hat)**2)
    f = np.array(y-y_hat) # Residual
    J = np.vstack((sol[:, 2], sol[:, 4])).T# Sensitivity of height with respect to k

    ga = J.T.dot(f)
    Ha = J.T.dot(J)
    return c, ga,Ha
```

```python
def gauss_newton_gradient_descent(p_init, y, t, const, initial_alpha, num_iterations):
    p = p_init.copy()
    p_history = []  # List to store the history of p values
    p_history.append(p)
    alpha = initial_alpha  # Start with the initial alpha value
    last_cost = float('inf')  # Initialize last cost as infinity
    finished = False
    eps_j = 1e-8
    eps_g = 1e-8
    k_max = 800
    k = 0
    while not finished:
        c, grad, Ha = crit_Gauss_NewTon_sensitive(p, y, t)
        # Calculate the descent direction
        d0 = np.linalg.solve(Ha, -grad)
        # Update parameters tentatively
        p_new = p + alpha * d0
        # Check if any updated parameters are negative, reduce alpha if so
        while np.any(p_new <= 0):
            alpha *= 0.5  # Reduce step size
            p_new = p + alpha * d0  # Recalculate the tentative new parameters
            print(f"Parameter update resulted in negative value, reducing step size to {alpha}")
        # Calculate new cost for p_new and compare with last_cost
        new_cost, _, _ = crit_Gauss_NewTon_sensitive(p_new, y, t)
        if new_cost > last_cost:
            alpha *= 0.5  # Halve the step size if the cost increases
            print(f"Cost increased, reducing step size to {alpha}")
        else:
        # Accept the new parameters since they are all positive and cost has not increased
            p = p_new
            last_cost = new_cost  # Update the last cost

        if new_cost < eps_j:
            print("Too small decrease of cost")
            finished = True
        if np.sum(grad**2) < eps_g:
            print("Too small gradient")
            finished = True
        if k >= k_max:
            print("Maximum iteration count reached")
            finished = True
        # Store the current value of p and print the iteration details
        p_history.append(p.copy())
        print(f"Iteration {k+1}: Cost = {c}, Parameters = {p}")
        k += 1
    return np.array(p_history)  # Return the history of p values as an array
```

Problem 17

Compare the obtained results with those obtained with gradient descent

In this lab work, we developed four different optimization processes to estimate the parameter $p = [k, n]$. Based on finite differences, we constructed the Gradient Descent and Gauss-Newton Gradient Descent algorithms. Subsequently, we performed a sensitivity analysis to construct the sensitivity function. Utilizing this sensitivity function, we further developed both the Gradient Descent and Gauss-Newton algorithms.

Based on the analysis of the provided diagram, it is evident that the Gauss-Newton method exhibits a superior convergence velocity compared to the Gradient Descent method. Furthermore, when comparing the Finite Difference approach with the Sensitivity Analysis, it becomes apparent that employing the Sensitivity Function leads to a significantly faster convergence velocity.

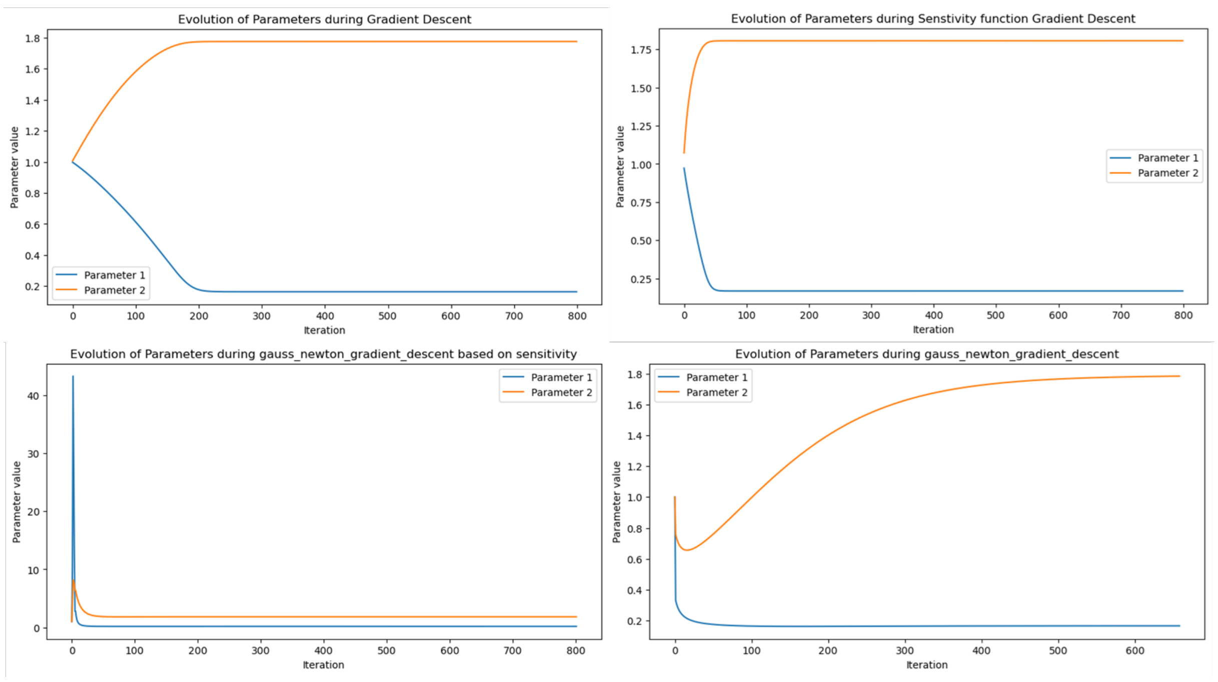

Figure 8: Comparing of the difference Optimization processing

Method / Metric	Gradient Descent	Gauss Newton
Finite Difference	0.16364295, 1.77462433	0.16510807, 1.78406
Residual	5.54×10^{-5}	5.70×10^{-5}
Sensitivity Analysis	0.16712946, 1.80654545	0.16871785, 1.82283597
Residual	4.48×10^{-5}	3.82×10^{-5}

In our comparative analysis, we evaluated the residuals obtained from the four methods under consideration. It was observed that the Sensitivity Function method demonstrated greater accuracy compared to the Finite Difference method. This conclusion was drawn from the consistently lower residual values associated with the Sensitivity Function method. Subsequently, we employed the concept of confidence

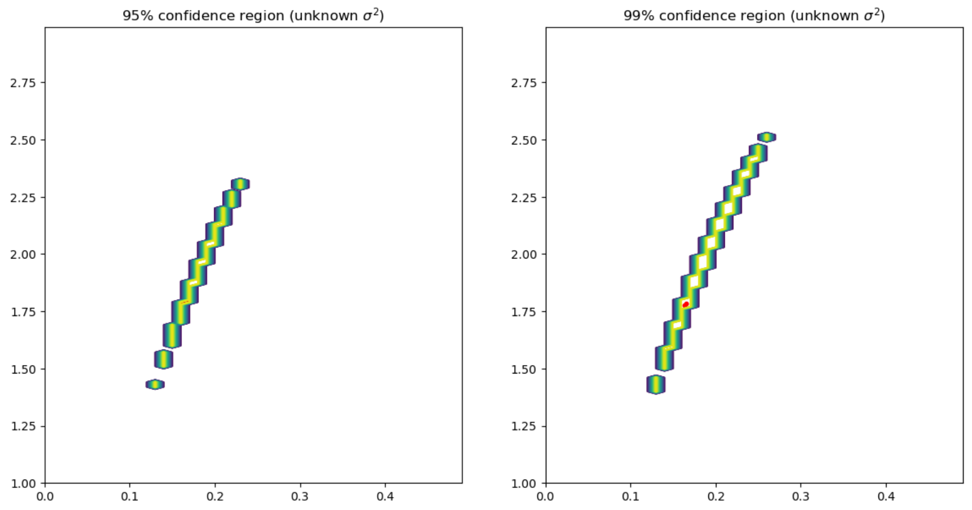

Figure 9: Distribution of the parameters in the confidence interval

intervals to validate the reliability of our estimated parameters. Through graphical representation, as illustrated in the accompanying figure, it was evident that all the estimated parameters fell within both the 95% and 99% confidence intervals. This alignment strongly suggests that our parameter estimations are statistically sound and reliable within the specified confidence levels.