3. Gauss-Newton method

The Gauss-Newton algorithm is an iterative method commonly used to solve nonlinear least-squares problems. This type of problem is formulated as an unconstrained optimization problem, where the sum of squared errors (SSE) is defined as the objective function to be minimized. In this section, we will describe the Gauss-Newton method.

3.1 Problem formulation

The Gauss-newton algorithm procedure consists of a series of linear approximations to the initial nonlinear problem. At each step, the nonlinear problem is approximated by a linear problem, which is then solved using direct or iterative techniques. By repeating this process, the solutions obtained gradually converge toward the optimal solution. Throughout this procedure, the SSE is minimized, enabling the unknown parameters of the proposed model to be optimally determined [2].

Problems of this nature [6, 7], frequently arise in areas such as optimal control, optimal filtering, and data fitting. Suppose we have N observed data $(t_i, b_i)_{i=1,\dots,N}$ which we wish to fit using a model $g(\mathbf{q}, t_i)$, $\mathbf{q}$ is the parameter. If we define the N components of $f(\mathbf{q})$ as $f_i(\mathbf{q}) = g(\mathbf{q}, t_i) - b_i$, then the solution of the nonlinear least squares problem (NLSP) provides the best fit of the model to the data, minimizing the sum of squared errors. The residual function f is the map defined by

$$f : \mathbb{R}^p \to \mathbb{R}^N, p < N,$$

with

$$f(\mathbf{q}) = g(\mathbf{q}, t_i) - b_i; \quad \text{for } i = 1, \dots N \tag{3.1.1}$$

3.2 Optimization problem

If $g(\mathbf{q})$ is strictly convex and coercive then the problem is to find a solution $\mathbf{q}^*$ such that

$$\|f(\mathbf{q}^*)\|^2 = \min_{\mathbf{q}} \|f(\mathbf{q})\|^2. \tag{3.2.1}$$

To find the value of $\mathbf{q}$, we need to solve $f(\mathbf{q}^*) = 0$ which doesn't have ~~an explicit~~ solution. Then look for the minimization of the objective function such that the predicted function is ~~very near to the variables measured over time~~. The objective function is defined by:

$$\begin{array}{rccc} G: & (\mathbb{R}^p) & \longrightarrow & \mathbb{R} \\ & \mathbf{q} & \longmapsto & \|f(\mathbf{q})\|^2 \end{array} \tag{3.2.2}$$

$$G(\mathbf{q}) = \sum_{i=1}^{N} f_i^2(\mathbf{q}) = \sum_{i=1}^{N} (g(\mathbf{q}, t_i) - bi)^2, \tag{3.2.3}$$

where $f(\mathbf{q})$ is the residual function, for $i = 1, \ldots, N$.

Let us consider the N residual functions which are smooth and continuously differentiable. To find $\mathbf{q}^*$, we compute

$$\nabla G(\mathbf{q}^*) = 0. \tag{3.2.4}$$

The gradient of the function is defined as follows:

$$\nabla G(\mathbf{q}) = 2\sum_{i=1}^{N} f_i(\mathbf{q})\nabla f_i(\mathbf{q}) = 2J_f(\mathbf{q})^T f(\mathbf{q}), \tag{3.2.5}$$

where

$$J_f = \begin{bmatrix} \frac{\partial f_1}{\partial q_1} & \frac{\partial f_1}{\partial q_2} & \cdots & \frac{\partial f_1}{\partial q_p} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_N}{\partial q_1} & \frac{\partial f_N}{\partial q_2} & \cdots & \frac{\partial f_N}{\partial q_p} \end{bmatrix} = \begin{bmatrix} \frac{\partial g(q,t_1)}{\partial q_1} & \frac{\partial g(q,t_1)}{\partial q_2} & \cdots & \frac{\partial g(q,t_1)}{\partial q_p} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial g(q,t_N)}{\partial q_1} & \frac{\partial g(q,t_N)}{\partial q_2} & \cdots & \frac{\partial g(q,t_N)}{\partial q_p} \end{bmatrix}. \tag{3.2.6}$$

This problem in (3.2.4) can be solved by applying Newton's method

$$\nabla G(q) = 2(J_f(\mathbf{q}))^T f(\mathbf{q}) = 0,$$

then we have:

$$J_f(\mathbf{q}))^T f(\mathbf{q}) = 0 \tag{3.2.7}$$

this is a system of N equations and p equations. if $N \neq p$ you cannot use Newton method

3.2.1 Newton method

The method employed aims to solve the system of nonlinear gradient equations (3.2.5) using a successive linearization approach. This consists of a series of iterations in which the non-linear equations are approximated by linear equations (see [1]). This requires the use of the full-rank Jacobian matrix of $f(\mathbf{q})$ ($f : \mathbb{R}^p \to \mathbb{R}^p$) to solve the system more explicitly. OK only if $N = p$

We are essentially trying to estimate $\nabla G(\mathbf{q})$ by using its linear approximation within a small area surrounding an initial estimate $\mathbf{q_0}$, it means that:

$$\nabla G(\mathbf{q}^*) = \nabla G(\mathbf{q_0}) + J_G(\mathbf{q_0}) \cdot (\mathbf{q}^* - \mathbf{q_0}) + O(||\mathbf{q}^* - \mathbf{q_0}||). \tag{3.2.8}$$

The initial approximation can be computed as follows:

$$J_{\nabla G}(\mathbf{q_0}) \cdot d_0 = -\nabla G(\mathbf{q_0}); \quad \mathbf{q_1} = \mathbf{q_0} + d_0. \tag{3.2.9}$$

Upon repeating this procedure knowing that the Jacobian matrices $J_{\nabla G}(\mathbf{q_0})$ are invertible for all possible input values q. Fixed point algorithm to solve a non-linear equation defined by $\Phi(\mathbf{q}) = q$,

with $\Phi(\mathbf{q}) = \mathbf{q} - \frac{\nabla G(\mathbf{q_0})}{J_{\nabla G}(\mathbf{q_0})}$, we arrive at the Newton iteration using approximation by fixed point sequence

$$\mathbf{q}_{k+1} = \Phi_{k+1}. \tag{3.2.10}$$

Then we have:

$$d_k = (J_{\nabla G}(\mathbf{q}_k))^{-1} \cdot (-\nabla G(\mathbf{q}_k)); \quad \mathbf{q}_{k+1} = \mathbf{q}_k + d_k; \quad k = 0, 1, \ldots, \tag{3.2.11}$$

where $J_{\nabla G}(\mathbf{q}) = \dfrac{\partial \nabla G(\mathbf{q})}{\partial q} = (\nabla G(\mathbf{q}))^2$ and d_k is the gradient descent direction .

Equation (3.2.11) is called the local Newton method and takes its name from its quadratic convergence in a restricted area around a single solution point(see[1]).

3.3 Local Gauss Newton

To estimate our parameter we need to compute the Hessian, which is given by:

$$\nabla^2 G(\mathbf{q}) = \sum_{i=1}^{N} \nabla f_i(\mathbf{q}) \nabla f_i(\mathbf{q})^T + \sum_{j=1}^{N} f_i(\mathbf{q}) \nabla^2 f_i(\mathbf{q}), \tag{3.3.1}$$

$$\nabla^2 G(\mathbf{q}) = J_f(\mathbf{q})^T J(\mathbf{q}) + \sum_{i=1}^{N} f_i(\mathbf{q}) \nabla^2 f_i(\mathbf{q}). \tag{3.3.2}$$

Calculating the second term on the right-hand side is costly for large problems. If the residuals are almost linear or very small near the solution, this term becomes negligible compared to the first one. This often makes the approximation effective, especially when $J_f(\mathbf{q})^T J_f(\mathbf{q})$ is dominant. In this context, if $|f_i(\mathbf{q})|\nabla^2 f_i(\mathbf{q})$ is small compared to the eigenvalues of $J(\mathbf{q})^T J_f(\mathbf{q})$, the Gauss-Newton method converges quickly, almost quadratically. Otherwise, the approximation is less reliable and the convergence becomes linear, requiring quasi-Newton or hybrid methods [1, 3]. The key to minimizing least squares problems is to approximate the Hessian from the Jacobian information in $\mathbf{q}_k$.

$$\nabla^2 G(q) = J_f(q_k)^T J_f(q_k) \approx \nabla^2 f_k. \tag{3.3.3}$$

This hessian is symmetrical and positively defined because the Jacobian is a full-rank matrix. To minimize the function (3.2.1), we can apply the Newton method by solving (3.2.11) in the least square sense, then we have:

$$J_f(\mathbf{q}_k)^T J_f(\mathbf{q}_k) d_k = -J_f(\mathbf{q}_k)^T f(\mathbf{q}_k). \tag{3.3.4}$$

Finally, we have:

$$d_k = -(J_f(\mathbf{q}_k)^T J_f(\mathbf{q}_k))^{-1} J_f(\mathbf{q}_k)^T f(\mathbf{q}_k). \tag{3.3.5}$$

The Gauss-Newton algorithm is given as follows:

Algorithm 1: Gauss-Newton Algorithm

1: **Step 1** Initialise $\mathbf{q} = \mathbf{q}_0$, tolerance $\varepsilon > 0$, and $k = 0$.
2: **Step 2** Compute the residual function $f_i(\mathbf{q}_k)$, $i = 0, \ldots, N$ from (3.1.1).
3: **Step 3 while**$(k < k_{max})$
Calculate the Jacobian matrices $J_f(\mathbf{q}_k)$, $H_k = J_f(\mathbf{q}_k)^T J_f(\mathbf{q}_k)$, and $S_k = \big(J_f(\mathbf{q}_k)^T f(\mathbf{q}_k)\big)$.
4: **Step 4** Calculate d_k by solving $(H_k d_k = -S_k)$. If $\|f_k\| < \varepsilon$, break.
5: **Step 5** Update $\mathbf{q}_{k+1} = \mathbf{q}_k + d_k$. Set $k = k + 1$

3.3.1 Convergence of Gauss-Newton

where did you find this proof.

Knowing that $J_f(\mathbf{q})$ is full rank and $G(\mathbf{q})$ is stricly convex coercive, let's d_k the direction given by (3.3.5) satisfies:

$$\langle \nabla G(\mathbf{q}_k), d_k \rangle \leq 0. \tag{3.3.6}$$

If $\mathbf{q}_k \neq \mathbf{q}^*$, then

$$\langle \nabla G(\mathbf{q}_k), d_k \rangle < 0. \tag{3.3.7}$$

So d_k is a descent direction for G at $\mathbf{q}^*$ If the sequence $\mathbf{q}_k$ converges, then its limit is $\mathbf{q}^*$.

Now let's show that Gauss-Newton works well if you start close to the solution q_0 near the q: From (3.2.11) we have:

$$\mathbf{q}_{k+1} = \mathbf{q}_k + d_k. \tag{3.3.8}$$

By iteration, for $i = 0, 1, \ldots k$, we have:

$$\begin{aligned} \mathbf{q}_1 &= \mathbf{q}_0 + d_0 \\ \mathbf{q}_2 &= \mathbf{q}_1 + d_1 \\ &\vdots \\ \mathbf{q}_k &= \mathbf{q}_{k-1} + d_{k-1} \\ \mathbf{q}_{k+1} &= \mathbf{q}_k + d_k. \end{aligned} \tag{3.3.9}$$

After the Addition of all iterations, we obtain:

$$\mathbf{q}_{k+1} = \mathbf{q}_0 + \sum_{i=0}^{k} d_i, \tag{3.3.10}$$

at the k^{th} iteration, we have:

$$\mathbf{q}_k = \mathbf{q}_0 + \sum_{i=0}^{k-1} d_i. \tag{3.3.11}$$

Since the convergence is satisfied if it respects the criterion $||d_k|| \leq \varepsilon$, then we have:

$$||\mathbf{q}_k - \mathbf{q}_0|| = ||\sum_{i=0}^{k-1} d_i||. \tag{3.3.12}$$

Apply triangular inequality we have:

$$||\mathbf{q}_k - \mathbf{q}_0|| = ||\sum_{i=0}^{k-1} d_i|| \leq \sum_{i=0}^{k-1} ||d_i||, \tag{3.3.13}$$

or $||\sum_{i=0}^{k-1} d_i|| \leq \sum_{i=0}^{k-1}, \varepsilon_i = k\varepsilon$, then we have:

$$||\mathbf{q}_k - \mathbf{q}_0|| \leq k\varepsilon, \tag{3.3.14}$$

For ε too small, $k\varepsilon \to 0$, therefore $||\mathbf{q}_k - \mathbf{q}_0|| \to 0$ which implies that

$$\mathbf{q}_k \to \mathbf{q}_0. \tag{3.3.15}$$

3.4 Advantages and Disadvantages of the Gauss Algorithm

Despite the Gauss-Newton algorithm has a quick convergence around the initial condition, it deals with many problems. Let's give some advantages and disadvantages of this method

this sounds like chatgpt

Advantages of the Gauss Algorithm:

- The Gauss algorithm is straightforward and easy to implement, making it accessible to beginners and useful for quick solutions.
- For small to moderate-sized systems of linear equations, the Gauss algorithm can be computationally efficient, especially when compared to more complex methods.
- When properly implemented, the Gauss algorithm is numerically stable, meaning it produces accurate results even with slight perturbations in the input data.

Disadvantages of the Gauss Algorithm:

- The basic Gauss algorithm doesn't incorporate pivoting, which can lead to numerical instability and inaccuracies, especially when dealing with ill-conditioned systems.
- The Gauss algorithm is less effective for large systems of equations or systems with special structures, such as sparse matrices.
- It doesn't provide error estimates for the computed solution, making it difficult to assess the quality of the solution.

In conclusion, the Gauss-Newton method converges for suitably nonlinear least squares problems, provided it begins in a sufficiently small neighborhood of a solution $\mathbf{q}$. That's why (3.3.5) is referred to as the local Gauss-Newton method(see[1]). In order to have a good convergence even for values far from the initial solution, some other algorithms are applied to solve this problem.

4. Presentation of linear test case

In dynamical systems, the common goal is to determine an equation that effectively describes the dynamic behavior of a given set of observed variables. In this section, we will apply the Gauss-Newton method to a two-dimensional linearly damped oscillator to identify the governing equations from noisy data.

4.1 Description of system

This study aims to solve the following parameters dependent linear system of differential equations:

$$\begin{cases} \frac{dx}{dt} = ax + by \\ \frac{dy}{dt} = cx + dy \end{cases} \tag{4.1.1}$$

which can be written, more compactly, as:

$$\frac{d}{dt}\begin{bmatrix} x \\ y \end{bmatrix} = A\begin{bmatrix} x \\ y \end{bmatrix}, \tag{4.1.2}$$

where A is the following square matrix made up of the unknown parameters a, b, c, d of the system (4.3.1):

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

and the vector $X := \begin{bmatrix} x \\ y \end{bmatrix}$ is called the state vector of the system.

Theoretically, the solution of the differential equation (4.1.2) is known and given by:

$$X(t) = \exp(At)X_0, \tag{4.1.3}$$

where $X_0 = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$ represents the initial condition of the system. Note that the matrix exponential $\exp(At)$ can be explicitly computed once the matrix A is known (and diagonalizable). However, the matrix A is unknown and our main goal is to find a matrix A that best fits the system (4.3.1).

To do this, let's assume that we have a device that provides us N measurements of the state vector X given by $X_m = \begin{bmatrix} X_1 \\ \vdots \\ X_N \end{bmatrix}$ where each measure $X_{m_i} = \begin{bmatrix} x_m(t_i) \\ y_m(t_i) \end{bmatrix}$ is given at time $t_i, \forall i = 1, \ldots, N$, the prediction function is defined by $g(A, t) = X(t) = \exp(At)X_0$.

Let us consider the following function:

$$\begin{array}{rccc} f_t: & \mathcal{M}_2(\mathbb{R}) & \longrightarrow & \mathbb{R}^{2N} \\ & A & \longmapsto & \exp(At)X_0 - X_m, \end{array} \tag{4.1.4}$$

where, $\mathcal{M}_2(\mathbb{R})$ is the set of square 2×2 matrices with real entries and $t = \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix}$ is a time measurement vector so that:

$$f_t(A) = \begin{bmatrix} \exp(At_1)X_0 - X_{m_1} \\ \exp(At_2)X_0 - X_{m_2} \\ \vdots \\ \exp(At_N)X_0 - X_{m_N} \end{bmatrix} \in \mathbb{R}^{2N}, \quad \forall A \in \mathcal{M}_2(\mathbb{R}). \tag{4.1.5}$$

One can naively try to find a matrix A such $f_t(A) = 0_{\mathbb{R}^{2N}}$ but that is impossible because of the complexity of the system we consider. However, we can try to find a matrix A which minimizes the mean square norm of $f_t(A)$. Therefore, we consider the following optimization problem:

$$\min_{A \in \mathcal{M}_2(\mathbb{R})} \|f_t(A)\|^2, \tag{4.1.6}$$

Let's consider N measurements over time. The goal is to find the values of unknown parameters $A = (a, b, c, d)$ that minimize the objective function $G(A)$ between the predicted values and the observed data (measurements). The **objective function** is express as.

$$\begin{array}{rccc} G: & \mathcal{M}_2(\mathbb{R}) & \longrightarrow & \mathbb{R} \\ & A & \longmapsto & \|f_t(A)\|^2 \end{array} \tag{4.1.7}$$

represents its corresponding objective function which can be explicitly written as:

$$G(A) = \sum_{i=1}^{N} |f_{t_i}(A)|^2 \tag{4.1.8}$$

Since

$$f_t(A) = \exp(At)X_0 - X_m$$

$$G(A) = \sum_{i=1}^{N} (\exp(At_i)X_0 - X_m))^2$$

with $(t_i)_{i=1,\ldots,N}$,

If our measurements are noisy then $X_m = X_i + d$, where d is a Gaussian noise distribution with zero mean defined as $d = np.random.norm(0, sigma)$, $sigma$ is the noise level.

4.2 Optimization problem

To find the values of the parameters a, b, c, and d, we need to look for $f_t(A) = 0$ which is not possible, then we search to minimize $G(A)$ so that the predicted function is as close as possible to the variables measured over the time.

Jacobian matrix $J \in \mu_{2N,4}$, assume $\forall A \in \mathcal{M}_2(\mathbb{R})$, $Jf_t(A)$ has a full rank, then it is symmetric defined positive.

If g is strictly convex and coercive then the problem $G(A)$ defined by (4.1.6) admits a unique solution $A^* = (a^*, b^*, c^*, d^*)$ such that $\nabla G(A^*) = 0$,

$$\nabla G(A) = 2(J_{f_t(A)})^T f_t(A), \tag{4.2.1}$$

where the jacobian matrix is defined by (3.2.6).

Replacing f_{t_i} in (3.2.6), we obtain:

$$J_f = \begin{bmatrix} \frac{\partial X_1}{\partial a} & \frac{\partial X_1}{\partial b} & \frac{\partial X_1}{\partial c} & \frac{\partial X_1}{\partial d} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial X_N}{\partial a} & \frac{\partial X_N}{\partial b} & \frac{\partial X_1}{\partial c} & \frac{\partial X_N}{\partial d} \end{bmatrix}. \tag{4.2.2}$$

Replacing $X_i(t)$ by its values we have:

$$J_f = \begin{bmatrix} \frac{\partial x_1}{\partial a} & \frac{\partial x_1}{\partial b} & \frac{\partial x_1}{\partial c} & \frac{\partial x_1}{\partial d} \\ \frac{\partial y_1}{\partial a} & \frac{\partial y_1}{\partial b} & \frac{\partial y_1}{\partial c} & \frac{\partial y_1}{\partial d} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial x_N}{\partial a} & \frac{\partial x_N}{\partial b} & \frac{\partial x_N}{\partial c} & \frac{\partial x_N}{\partial d} \\ \frac{\partial y_N}{\partial a} & \frac{\partial y_N}{\partial b} & \frac{\partial y_N}{\partial c} & \frac{\partial y_N}{\partial d} \end{bmatrix}.$$

Finally, we have the following Jacobian

$$J_f = \begin{bmatrix} \frac{\partial x}{\partial a} & \frac{\partial x}{\partial b} & \frac{\partial x}{\partial c} & \frac{\partial x}{\partial d} \\ \frac{\partial y}{\partial a} & \frac{\partial y}{\partial b} & \frac{\partial y}{\partial c} & \frac{\partial y}{\partial d} \end{bmatrix}, \tag{4.2.3}$$

with $x = x_i(t)$, $y = y_i(t)$ represent the sensitivity of the parameters at the measurement over the time, $i = 1, \ldots, N$, where J_f is a $2N \times 4$.

Let's find the zeros of $\nabla G(A)$, to do this, we use the Newton method requires $Hf(A)$. Finding the zeros by (3.2.11), we have:

$$A_{k+1} = A_k + d_k, \tag{4.2.4}$$

$$Jf_t(A)d_k = -f_t(A). \tag{4.2.5}$$

Then solving (4.2.5) using the least square method we have:

$$Jf_t(A)^T Jf_t(A)d_k = -Jf_t(A)^T f_t(A).$$

Therefore

$$d_k = -(Jf_t(A)^T Jf_t(A))^{-1} Jf_t(A)^T f_t(A), \tag{4.2.6}$$

where d_k is the gradient descent.

The Gauss newton algorithm is given by:

Algorithm 2: Gauss newton algorithm

Data: Dataset
Initialise a, b, c, d, k
while $k < k_{max}$ **do**
  $H_k \leftarrow \mathsf{Jf}_t(A_k)^T Jf_t(A_k)$
  $S_k \leftarrow \mathsf{Jf}_t(A_k)^T f_t(A_k)$
  solve$(H_k d^k = -S_k)$
  Update
  $A_{k+1} \rightarrow A_k + dk$ `// a while loop`

  if $np.linalg.norm(d_k) < eps:$ **then**
    break
    $k \rightarrow k + 1$ `// an if...else conditional loop`
  return a,b,c,d,k

with $a, b, c, d \in \mathbb{R}$, k is the itterations, $Jf_t(A)^T$ is $4 \times 2N$ matrix.

4.3 Jacobian Matrix calculation

In addition to the map f_t, we need the Jacobian matrix in (4.2.2) to apply the Gauss-Newton method. To find it we need to find each parameter's sensitivities.

Let's analyze the sensitivity of the system's output with respect to changes in its parameters. Given the system of differential equations:

$$\frac{dX}{dt} = AX = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}. \tag{4.3.1}$$

We aim to find the Jacobian matrix. To simplify this process, we introduce new variable $Z(t)$ define as:

$$Z(t) = \begin{pmatrix} X \\ Z_X \\ Z_Y \end{pmatrix},$$

where $Z(t)$ is the state transition matrix

Let,

$$Z_X = \begin{pmatrix} \frac{\partial x}{\partial a} \\ \frac{\partial x}{\partial b} \\ \frac{\partial x}{\partial c} \\ \frac{\partial x}{\partial d} \end{pmatrix}, \quad Z_Y = \begin{pmatrix} \frac{\partial y}{\partial a} \\ \frac{\partial y}{\partial b} \\ \frac{\partial y}{\partial c} \\ \frac{\partial y}{\partial d} \end{pmatrix}, \quad X = \begin{pmatrix} x \\ y \end{pmatrix}$$

which are the components of the new variable. Then, we want to find Z_X and Z_Y representing the Jacobian's components which will be extracted using $Z(t)$.

The derivative of $Z(t)$ with respect to t becomes:

$$\frac{dZ(t)}{dt} = \begin{pmatrix} \frac{dX}{dt} \\ \frac{dZ_X}{dt} \\ \frac{dZ_Y}{dt} \end{pmatrix}. \tag{4.3.2}$$

In this system:

- X is the state vector defined by (4.3.1).

- Z_X contains derivatives of x with respect to a, b, c, and d.

- Z_Y contains derivatives of y with respect to a, b, c, and d.

Let's find Z_X and Z_Y, these parameters are calculated using the chain rule [1] as follows:

$$\begin{aligned} \tfrac{d}{dt}(\tfrac{\partial x}{\partial a}) &= \tfrac{\partial}{\partial a}(\tfrac{dx}{dt}) \\ \tfrac{d}{dt}(\tfrac{\partial y}{\partial a}) &= \tfrac{\partial}{\partial a}(\tfrac{dy}{dt}) \end{aligned} \tag{4.3.3}$$

replacing $\frac{dx}{dt}$ and $\frac{dy}{dt}$ by theirs values in (4.3.1), we obtain:

$$\tfrac{d}{dt}(\tfrac{\partial x}{\partial a}) = x + a\tfrac{\partial x}{\partial a} + b\tfrac{\partial y}{\partial a}$$

$$\tfrac{d}{dt}(\tfrac{\partial y}{\partial a}) = y + a\tfrac{\partial x}{\partial b} + b\tfrac{\partial y}{\partial b}.$$

By expressing the derivatives using the same process in (4.3.3) and replacing them in (4.3.2), we obtain:

$$\frac{dZ(t)}{dt} = \begin{pmatrix} ax + by \\ cx + dy \\ x + a\frac{\partial x}{\partial a} + b\frac{\partial y}{\partial a} \\ y + a\frac{\partial x}{\partial b} + b\frac{\partial y}{\partial b} \\ a\frac{\partial x}{\partial c} + b\frac{\partial y}{\partial c} \\ a\frac{\partial x}{\partial d} + b\frac{\partial y}{\partial d} \\ c\frac{\partial x}{\partial a} + d\frac{\partial y}{\partial a} \\ c\frac{\partial x}{\partial b} + d\frac{\partial y}{\partial b} \\ x + c\frac{\partial x}{\partial c} + d\frac{\partial y}{\partial c} \\ y + c\frac{\partial x}{\partial d} + d\frac{\partial y}{\partial d} \end{pmatrix}. \tag{4.3.4}$$

Thus, the system becomes:

$$\frac{dZ}{dt} = \begin{pmatrix} a & b & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ c & d & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & a & 0 & 0 & 0 & b & 0 & 0 & 0 \\ 0 & 1 & 0 & a & 0 & 0 & 0 & b & 0 & 0 \\ 0 & 0 & 0 & 0 & a & 0 & 0 & 0 & b & 0 \\ 0 & 0 & 0 & 0 & 0 & a & 0 & 0 & 0 & b \\ 0 & 0 & c & 0 & 0 & 0 & d & 0 & 0 & 0 \\ 0 & 0 & 0 & c & 0 & 0 & 0 & d & 0 & 0 \\ 1 & 0 & 0 & 0 & c & 0 & 0 & 0 & d & 0 \\ 0 & 1 & 0 & 0 & 0 & c & 0 & 0 & 0 & d \end{pmatrix} \begin{pmatrix} x \\ y \\ \frac{\partial x}{\partial a} \\ \frac{\partial x}{\partial b} \\ \frac{\partial x}{\partial c} \\ \frac{\partial x}{\partial d} \\ \frac{\partial y}{\partial a} \\ \frac{\partial y}{\partial b} \\ \frac{\partial y}{\partial c} \\ \frac{\partial y}{\partial d} \end{pmatrix}.$$

Where M is a 10×10 transitivity matrix defined by:

$$M = \begin{pmatrix} a & b & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ c & d & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & a & 0 & 0 & 0 & b & 0 & 0 & 0 \\ 0 & 1 & 0 & a & 0 & 0 & 0 & b & 0 & 0 \\ 0 & 0 & 0 & 0 & a & 0 & 0 & 0 & b & 0 \\ 0 & 0 & 0 & 0 & 0 & a & 0 & 0 & 0 & b \\ 0 & 0 & c & 0 & 0 & 0 & d & 0 & 0 & 0 \\ 0 & 0 & 0 & c & 0 & 0 & 0 & d & 0 & 0 \\ 1 & 0 & 0 & 0 & c & 0 & 0 & 0 & d & 0 \\ 0 & 1 & 0 & 0 & 0 & c & 0 & 0 & 0 & d \end{pmatrix}.$$

The state vector is defined as:

$$Z(t) = \left(x, y, \tfrac{\partial x}{\partial a}, \tfrac{\partial x}{\partial b}, \tfrac{\partial x}{\partial c}, \tfrac{\partial x}{\partial d}, \tfrac{\partial y}{\partial a}, \tfrac{\partial y}{\partial b}, \tfrac{\partial y}{\partial c}, \tfrac{\partial y}{\partial d}\right)^T.$$

If X_0 is the initial condition for the 2×2 system, then the initial condition for the 10×10 matrix is:

$$Z_0 = \left(x_0, y_0, 0, 0, 0, 0, 0, 0, 0, 0\right)^T.$$

The sensitivity system's differential equation is:

$$\frac{dZ}{dt} = MZ(t), \tag{4.3.5}$$

and the solution of (4.3.5) is given by:

$$Z(t) = \exp(Mt)Z_o. \tag{4.3.6}$$

Then using this solution we can extract the Jacobian matrix define in (3.2.6).

4.4 Results and Discussion

you have to say here that it is a complicated method for this simple case, but unavoidable when you do not know explicitely the solution of the ODE in the general case.

$$F(A) \in M_{2\times 2}(\mathbb{R}) \longrightarrow \mathbb{R}^{N\times 2}$$
$$A \longrightarrow \begin{pmatrix} e^{At_1}X_0 \\ \vdots \\ e^{At_N}X_0 \end{pmatrix}$$

$$F(A+h) = F(A) + JF(A)h + o(\|h\|)$$

should be simpler to compute.

5. Levenberg Marquardt method

The Levenberg-Marquardt method is an iterative process used to locate the minimum of a function expressed as the sum of squares of nonlinear functions. It has become a common method for nonlinear least squares problems(reference) and can be described as a combination of gradient descent and the Gauss-Newton method. It is often referred to as a globalization of the Gauss-Newton method(reference) and is an advanced extension of it.

5.1 Optimization problem

Considering the same problem in (3.2.3), we want to find the optimal parameters that fit our model as best as possible. Levenberg-Marquardt is a modification of the Gauss-Newton algorithm [4, 5] which is based on the Taylor approximation of the predicted function $g(\mathbf{q}, t)$ with $\mathbf{q} \in \mathbb{R}^N$, about an initial solution $\mathbf{q}_0$ as follows:

$$g(\mathbf{q}, t) = g(\mathbf{q}_0, t) + J_g(\mathbf{q})(q - q_0). \tag{5.1.1}$$

Let's $\delta(\mathbf{q}) = (\mathbf{q} - \mathbf{q}_0)$, then (5.1.2) become:

$$g(\mathbf{q}) = g(\mathbf{q}_0, t) + J_g(\mathbf{q})\delta(\mathbf{q}), \tag{5.1.2}$$

where $J_g(\mathbf{q}) = \dfrac{\partial g(\mathbf{q}, t)}{\partial q}$ represent the jacobian matrix of $g(\mathbf{q}, t)$ respect to $\mathbf{q}$.

Using (5.1.2) in (3.1.1) we have the following expression:

$$f(\mathbf{q}) = g(\mathbf{q}_0, t) + J_g(\mathbf{q})\delta(\mathbf{q}) - b, \tag{5.1.3}$$

here, $f(\mathbf{q})$ represents the residuals function, b are observed data, $g(\mathbf{q}_0, t)$ are the function values at the initial guess $\mathbf{q}_0$, $J_g(\mathbf{q})$ is the Jacobian matrix of $g(\mathbf{q})$ and $\delta(\mathbf{q})$ are the unknowns to be determined.

The equation (5.1.3) can be written as follows:

$$f(\mathbf{q}) = K + J_g(\mathbf{q}, t)\delta(\mathbf{q}), \tag{5.1.4}$$

where K is defined by:

$$K = g(\mathbf{q}, t) - b.$$

We want to minimize the sum of squared residuals:

$$G(\mathbf{q}) = \sum_{i=1}^{N} f_i^2(\mathbf{q}) = f^T(\mathbf{q}) f(\mathbf{q}). \tag{5.1.5}$$

Expanding the objective function and substituting $f(\mathbf{q}) = K + J_g(\mathbf{q})\delta\mathbf{q}$ in (5.1.5) we have:

$$G(\mathbf{q}) = (K + J_g(\mathbf{q})\delta(\mathbf{q}))^T (K + J_g(\mathbf{q})\delta(\mathbf{q})).$$

Expanding the product:

$$G(\delta_q) = K^T K + 2K^T J_g(\mathbf{q})\delta\mathbf{q} + (\delta\mathbf{q})^T J_g(\mathbf{q})^T J_g(\mathbf{q})\delta\mathbf{q}.$$

Since $\mathbf{q} = \mathbf{q}_0 + \delta(\mathbf{q})$, to minimize $G(\delta\mathbf{q})$, we apply formula (3.2.4), where $\nabla G(\delta\mathbf{q})$ is the derivative of $G(\delta\mathbf{q})$ with respect to $\delta\mathbf{q}$ and set it to zero:

$$\nabla G = \frac{\partial G(\mathbf{q})}{\partial \delta\mathbf{q}} = 2J_g(\mathbf{q})^T K + 2J_g(\mathbf{q})^T J_g(\mathbf{q})\delta\mathbf{q} = 0.$$

Solving for $\delta\mathbf{q}$, we have:

$$J_g(\mathbf{q})^T J_g(\mathbf{q})\delta\mathbf{q} = J_g(\mathbf{q})^T r, \tag{5.1.6}$$

with

$$r(\mathbf{q}) = -K = b - g(\mathbf{q}, t). \tag{5.1.7}$$

Now, it is necessary to show that the $\delta\mathbf{q}$ obtained from (5.1.6) minimizes G. To demonstrate this, we need to examine the Hessian matrix of G, denoted as H_G, which contains the second derivatives of G with respect to the components of $\delta\mathbf{q}$.

$$H_G = \frac{\partial^2 G(\delta\mathbf{q})}{\partial \delta^2} = \left(J_g(\mathbf{q})^T J_g(\mathbf{q})\right), \tag{5.1.8}$$

$$\delta\mathbf{q} = (J_g(\mathbf{q})^T J_g(\mathbf{q}))^{-1} J_g(\mathbf{q})^T r(\mathbf{q}). \tag{5.1.9}$$

In summary, $\delta\mathbf{q}$ that minimizes the sum of squared residuals can be found using the normal equation:

$$\delta\mathbf{q} = (J_g(\mathbf{q})^T J_g(\mathbf{q}))^{-1} J_g(\mathbf{q})^T r(\mathbf{q}). \tag{5.1.10}$$

Consider that at each current iterate $q(k)$, then we have:

$$\delta\mathbf{q}_k = (J_g(\mathbf{q}_k)^T J_g(\mathbf{q}_k))^{-1} J_g(\mathbf{q}_k)^T r(\mathbf{q}_k), \tag{5.1.11}$$

then to compute the updated estimate at k iteration, we use the formula:

$$\mathbf{q}_{(k+1)} = \mathbf{q}_{(k)} + \delta\mathbf{q}_{(k)}. \tag{5.1.12}$$

In this formula:

- $\mathbf{q}_{(k+1)}$ is the updated estimate $\mathbf{q}$.

- $\mathbf{q}_{(k)}$ is the current estimate of $\mathbf{q}$.

- $\delta\mathbf{q}_{(k)}$ is the computed adjustment from the previous step

- $r(\mathbf{q_k})$ is the new residual function.

The update process iteratively improves the estimate of $\mathbf{q}$ by adding the adjustment $\delta(\mathbf{q})$ to the current estimate $\mathbf{q}_{(k)}$. Gradient The iterative method for minimizing the residuals, using $\delta\mathbf{q}$ from

(5.1.12) and calculated via (5.1.11), is known as the Gauss-Newton method. This method can also be derived using other approaches [8]. However, as mentioned at the beginning, one issue with this method is that If the matrix J^TJ is poorly conditioned, $\delta\mathbf{q}_{(k)}$ can become very large when the initial guess $\mathbf{q}_0$ is not close to the true minimum. As a result, the Gauss-Newton method might take large and uncontrolled steps. This means that the successive parameter updates can be too large, leading to divergence rather than convergence to the optimal solution. This can cause the iterative process to fail to converge. To address the limitations of the Gauss-Newton method, Levenberg and Marquardt proposed damping the J^TJ matrix using a diagonal adjustment [4], [5]. Consequently, the Levenberg-Marquardt algorithm updates its steps as follows:

$$\delta\mathbf{q} = (J_g(\mathbf{q})^T J_g(\mathbf{q}) + \lambda I)^{-1} J_g(\mathbf{q})^T r(\mathbf{q}), \tag{5.1.13}$$

where λ is a damping parameter ($\lambda > 0$) and I the identity matrix. Consider that at each current iterate $\mathbf{q}(k)$, then we have:

$$\delta\mathbf{q}_k = (J_g(\mathbf{q}_k)^T J_g(\mathbf{q}_k) + \lambda_k I)^{-1} J_g(\mathbf{q}_k)^T r(\mathbf{q}_k), \tag{5.1.14}$$

then to compute the updated estimate at k iteration, we use the formula:

$$\mathbf{q}_{(k+1)} = \mathbf{q}_{(k)} + \delta\mathbf{q}_{(k)}. \tag{5.1.15}$$

Now, let's consider equation (5.1.13), where $\delta\mathbf{q}$ and c from equation (5.1.6), and suppose $\delta\mathbf{q}$ approaches infinity (it is too large). In this scenario, the first term on the left side of the equation becomes insignificant, leading to the solution.

$$\delta\mathbf{q}_g \approx \frac{1}{\lambda_k}\mathbf{J_g}(\mathbf{q_k})^T\mathbf{r}(\mathbf{q_k}) = -\frac{1}{2\lambda_k}\mathbf{J_g}(\mathbf{q_k})^T\mathbf{r}(\mathbf{q_k}) \to 0 \quad \text{as} \quad \lambda_k \to \infty. \tag{5.1.16}$$

Thus, in this limiting case, the damped least-squares solution $\delta\mathbf{q}_g$ aligns with the negative gradient direction. Moreover, $\delta\mathbf{q}_g$ tends to zero as λ_k decreases. This emphasizes the direction.

The vector $\mathbf{J_g}(\mathbf{q_k})^T\mathbf{r}(\mathbf{q_k})$ forms the basis of the steepest-descent method for minimization, which is one of the oldest optimization methods[8]. The introduction to the steepest-descent method is as follows, we seek an iterative process such that

$$G_{(k+1)}(\mathbf{q}) \leq G_{(k)}(\mathbf{q}). \tag{5.1.17}$$

To achieve this, we use the fact that ∇G points in the direction of the steepest ascent. This is a well-known calculus result(see [8] and will be demonstrated below in a more general context. Hence, the initial estimate for the $(k+1)^{th}$ iteration is calculated using

$$\mathbf{q}_{(k+1)} = \mathbf{q}_{(k)} - \alpha_{(k)}\nabla G\left(\mathbf{q}_{(k)}\right), \tag{5.1.18}$$

where $\alpha_{(k)} = \dfrac{1}{\lambda_k}$ is a step size parameter. Here, $\alpha_{(k)}$ is a scalar that ensures equation (5.1.17) is satisfied. The question arises on how to choose the value of $\lambda_{(k)}$. Due to this, various strategies for selecting it have been developed [10].

If λ_k is small, then $\delta\mathbf{q}_k$ satisfied the Gauss-Newton step define by (5.1.10), Therefore, the value of λ_k influences both the direction and magnitude of the step $\delta\mathbf{q}_k$. When $\mathbf{q}$ is near the solution, the faster convergence of the Gauss-Newton method is desirable. However, when $\mathbf{q}$ is far from $\mathbf{q}^*$, the robustness of the steepest descent method is preferred. Levenberg Algorithm is given as follows:

Algorithm 3: Levenberg-Marquardt algortihm

1: **Step 1** Initialise $\mathbf{q} = \mathbf{q}_0$, $\lambda = \lambda_0$, tolerance $\varepsilon > 0$, and $k = 0$.
2: **Step 2** Compute the function $r_i(\mathbf{q}_k),\ i = 0, \ldots, N$ from (5.1.7).
3: **Step 3 while**$(k < k_{max})$
Calculate the Jacobian matrices $J_r(\mathbf{q}_k)$, $H_k = J_r(\mathbf{q}_k)^T J_r(\mathbf{q}_k)$, and $S_k = \left(J_r(\mathbf{q}_k)^T r(\mathbf{q}_k)\right)$.
4: **Step 4** Calculate d_k by solving $(H_k + \lambda \mathbf{I})d_k = -S_k$.
5: **Step 5**update parameter $\mathbf{q}_{k+1} = \mathbf{q}_k + d_k$
6: **Step 6** If $\|d_k\| < \varepsilon$ and $\|r(\mathbf{q_{k+1}}\| < \|r(\mathbf{q_k})\|$, set $\lambda = \lambda * \mu$,
if $\|r(\mathbf{q}_{k+1})\| > \|r(\mathbf{q_k})\|$ set $\lambda = \dfrac{\lambda}{\mu}$
7: **Step 7** Update $\mathbf{q}_{k+1} = \mathbf{q}_k + d_k$. Set $k = k + 1$

5.2 Choice of damping parameter

A critical aspect of the Levenberg-Marquardt algorithm is selecting the damping parameter, λ. This parameter controls the blend between the gradient descent method and the Gauss-Newton method. The selection of λ can be done in two different methods.

5.2.1 Direct Method

In this method the value of λ is dynamically made based on the current behavior of the algorithm. If a proposed step leads to an increase in the cost function, it indicates that the algorithm is likely moving away from the optimal solution. Therefore, to avoid taking excessively large steps that could lead to divergence or instability, the value of λ is increased. This increase in λ ensures that subsequent steps will be smaller, allowing the algorithm to gradually approach the optimal solution.

Numerous strategies have emerged for efficiently adjusting λ or δ. The straightforward approach originally proposed by Marquardt tends to suffice.

- If $\|\mathbf{f}_{k+1}\|^2 < \|\mathbf{f}_k\|^2$: a step is accepted, $\lambda = \lambda * \mu$ is decreased by a fixed factor $\mu \geq 1$, typically $\mu = 10$.
- Conversely, if $\|\mathbf{f}_{k+1}\|^2 > \|\mathbf{f}_k\|^2$: a step is rejected, $\lambda = \dfrac{\lambda}{\mu}$ is appropriately increased by a factor of $\mu = 10$.

As detailed in [9], the damping term's qualitative impact is to ensure that the eigenvalues of the matrix $\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I}$ are at least λ. Often, the eigenvalues of $J^T J$ exhibit logarithmic spacing. Thus, it's natural to adjust λ by a factor comparable to the spacing of these eigenvalues. In practice, a factor of 2 or 3 yields better results in most scenarios. Moreover, lowering λ by a larger factor than raising it often proves more advantageous. This approach known as delayed gratification, is motivated further in [9]

5.3 Indirect Method

The fundamental strategy to ensure that the eigenvalues of the matrix $\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I}$ can also be extended to an indirect method by systematically adjusting δ by a multiplicative factor. In this method, we firstly selects an acceptable step size and then finds a corresponding λ that ensures the step size is within the desired bounds. This method leverages the relationship between step size and λ. Here λ is defined as a maximal eigenvalue of the Hessian matrix (J^TJ) and the section of λ uses the same principle defined in the section above. The largest diagonal element of the initial matrix $\mathbf{J}_r^\top\mathbf{J}_r$ is of the same order of magnitude as $\max\{\text{eigenvalues}(\mathbf{J}_r^\top\mathbf{J}_r)\}$, then we chosen it as follows:

$$\lambda = \Gamma \max\left\{(\mathbf{J}_r(\mathbf{q})^T\mathbf{J}_r(\mathbf{q}))_{ii}\right\}, \tag{5.3.1}$$

where Γ values($\Gamma > 0$) varies between $[10^{-8}, 1]$.

This method is generally effective due to its flexibility and ability to adapt to varying eigenvalue spacings. It offers a more sophisticated approach to step size adjustment, which is crucial for the complex landscapes encountered in nonlinear optimization. It results in more efficient, and reliable convergence compared to the direct method [10].

The convergence is influenced by the initial parameter guess. If the starting point is close to the true minimum, the algorithm converges rapidly. However, if the initial guess is far from the minimum, the algorithm might converge slowly or get stuck in a local minimum.

5.4 Avantages and disadvantages of Levenberg-Marquardt algorithm

Despite the Levenberg-Marquart algorithm can resolve a larger optimization problem than the Gauss-Newton, it deals with some limitations. Let's give the advantages and disadvantages of the Levenberg-Marquardt Algorithm.

Advantages of the Levenberg-Marquardt Algorithm

- The combination of gradient descent and Gauss-Newton methods makes the Levenberg-Marquardt algorithm robust against poor initial guesses.
- For problems where the residuals are well-approximated by a linear function, Levenberg-Marquardt can converge very quickly.
- The algorithm's ability to adjust between gradient descent and Gauss-Newton allows it to handle a wide range of optimization problems.

Disadvantages of the Levenberg-Marquardt Algorithm

- While robust, the performance can still be sensitive to initial parameter guesses.
- For large problems, the evaluation of the Jacobian matrix can be computationally expensive.
- Like many optimization algorithms, The Levenberg-Marquardt algorithm, despite its robustness, can sometimes converge to local minima instead of finding the global minimum, especially in complex, multi-modal landscapes.

In summary, the Levenberg-Marquardt algorithm is a powerful tool for nonlinear least squares problems, offering a balance between speed and robustness. However, its effectiveness heavily relies on the appropriate selection of the damping parameter and good initial guesses. Despite its computational cost, especially for large problems, the algorithm's adaptability makes it a preferred choice in many scientific and engineering applications.

6. Non-linear test case

6.1 Exact solution calculation

In this section, we will apply the Gauss-Newton method to a two-dimensional non-linear damped oscillator to identify the governing equations from noisy data.

Now we deal with an example that is not linear like the previous example $x'(t) = Ax(t)$ with A a constant matrix. This system is defined as follows:

$$\begin{cases} \dfrac{dx}{dt} = ax^3 + by^3 \\ \dfrac{dy}{dt} = cx^3 + dy^3. \end{cases} \tag{6.1.1}$$

Here it's $x'(t) = F(X(t), A)$, with $X = (x, y)$ and $A = (a, b, c, d)$, $X \in \mathbb{R}^N$. So $F : \mathbb{R}^{2N} \to \mathbb{R}^2$. When we derive X with respect to A, we obtain, by noting $z_{x1,j}$ and $z_{y2,j}$ the derivatives of x, y respectively with respect to the parameters a, b, c, d and $z(t) = (x, y, z_{x1,1}, \ldots, z_{x1,N}, z_{y2,1}, \ldots, z_{y2,N})$.

$$\begin{gathered} \frac{dz(t)}{dt} = G(z(t), u) \\ \text{with} \quad G : \mathbb{R}^{2+2N} \times \mathbb{R} \to \mathbb{R}^{2+2N}. \end{gathered} \tag{6.1.2}$$

The first two components of the G function are F and the next $2N$ are the second members you calculated for $\dfrac{dz_x}{dt}$ and $\dfrac{dz_y}{dt}$. Using the same process as in the case of the linear system we have:

$$\begin{aligned} \frac{d}{dt}\frac{\partial x}{\partial a} = \frac{\partial}{\partial a}\frac{dx}{dt} &= x^3 + 3ax^2\frac{\partial x}{\partial a} + 3by^2\frac{\partial y}{\partial a} \\ \frac{d}{dt}\frac{\partial y}{\partial a} = \frac{\partial}{\partial a}\frac{dy}{dt} &= y^3 + 3ax^2\frac{\partial x}{\partial b} + 3bx^2\frac{\partial y}{\partial b}. \end{aligned} \tag{6.1.3}$$

So applied the same process to find all those derivatives and replace them in (6.1.2), we obtain:

$$\frac{dz(t)}{dt} = \begin{pmatrix} ax^3 + by^3 \\ cx^3 + dy^3 \\ x^3 + 3ax^2\frac{\partial x}{\partial a} + 3bx^2\frac{\partial y}{\partial a} \\ y^3 + 3ax^2\frac{\partial x}{\partial b} + 3bx^2\frac{\partial y}{\partial b} \\ 3ax^2\frac{\partial x}{\partial c} + 3bx^2\frac{\partial y}{\partial c} \\ 3ax^2\frac{\partial x}{\partial d} + 3bx^2\frac{\partial y}{\partial d} \\ 3cx^2\frac{\partial x}{\partial a} + 3dx^2\frac{\partial y}{\partial a} \\ 3cx^2\frac{\partial x}{\partial b} + 3dx^2\frac{\partial y}{\partial b} \\ x^3 + 3cx^2\frac{\partial x}{\partial c} + 3dx^2\frac{\partial y}{\partial c} \\ y^3 + 3cx^2\frac{\partial x}{\partial d} + 3dx^2\frac{\partial y}{\partial d} \end{pmatrix}. \tag{6.1.4}$$

Once you have your G function well defined, you can solve the differential system $\frac{z(t)}{dt} = G(z(t), A)$ with a solver in Python to get

$$z(t) = (x, y, \frac{\partial x}{\partial a}, \frac{\partial x}{\partial b}, \frac{\partial x}{\partial c}, \frac{\partial x}{\partial d}, \frac{\partial y}{\partial a}, \frac{\partial y}{\partial b}, \frac{\partial y}{\partial c}, \frac{\partial y}{\partial d}).$$

6.2 Application of Gauss Newton

References

[1] Peter Deuflhard and Susanna Röblitz. *A guide to numerical modelling in systems biology*, volume 12. Springer, 2015.

[2] Jerry Eriksson. *Optimization and regularization of nonlinear least squares problems*. Verlag nicht ermittelbar Jerusalem, 1996.

[3] Serge Gratton, Amos S Lawless, and Nancy K Nichols. Approximate gauss–newton methods for nonlinear least squares problems. *SIAM Journal on Optimization*, 18(1):106–132, 2007.

[4] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

[5] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[6] Jorge Nocedal and Stephen J Wright. Theory of constrained optimization. *Numerical optimization*, pages 304–354, 2006.

[7] James M Ortega and Werner C Rheinboldt. *Iterative solution of nonlinear equations in several variables*. SIAM, 2000.

[8] Jose Pujol. The solution of nonlinear inverse problems and the levenberg-marquardt method. *Geophysics*, 72(4):W1–W16, 2007.

[9] Mark K Transtrum, Benjamin B Machta, and James P Sethna. Geometry of nonlinear least squares with applications to sloppy models and optimization. *Physical Review E*, 83(3):036701, 2011.

[10] Mark K Transtrum and James P Sethna. Improvements to the levenberg-marquardt algorithm for nonlinear least-squares minimization. *arXiv preprint arXiv:1201.5885*, 2012.