Table of Contents

Usage

1. Install requirements:

```
pip install -r requirements.txt
```

2. Run `main.py` or go through the notebook `demo.ipynb` to see the example of usage.

Problem

ETU 2024 CompMath exponential sum fitting implementation

Given $(t_i, y_i)_{i=1}^n$, where X is 1-dimensional values and Y is 1-dimensional target.

Consider p is amount of exponential terms so we are going to fit:

$$f : X \to Y;\ f(t, \mathbf{p}) = \sum_{i=1}^{p} \lambda_i \alpha_i^t$$

Where $\mathbf{p} = (\lambda_1, \dots, \lambda_p, \alpha_1, \dots, \alpha_p)$

Assuming $\forall i\ \alpha_i > 0$ we can rewrite $f(t, \mathbf{p})$ as:

$$f(t, \mathbf{p}) = \sum_{i=1}^{p} \lambda_i \exp(\ln(\alpha_i)t) = \sum_{i=1}^{p} \lambda_i \exp(\omega_i t)$$

Finally, $\mathbf{p} = (\lambda_1, \dots, \lambda_p, \omega_1, \dots, \omega_p)$ – are parameters to fit

Loss function to optimize is MSE:

$$L(\mathbf{p}) = \sum_{i=1}^{n}(y_i - f(t_i, \mathbf{p}))^2$$

The method to optimize will be Levenberg-Marquardt algorith used in built-in curve_fit optimizer for scipy.

Levenberg-Marquardt algorithm (*LMA*)

Like other numeric minimization algorithms, the Levenberg–Marquardt algorithm is an iterative procedure. To start a minimization, the user has to provide an initial guess for the parameter vector $\mathbf{p}^T = \big(1, 1, \ldots, 1\big)$ will work fine; in cases with multiple minima, the algorithm converges to the global minimum only if the initial guess is already somewhat close to the final solution.

In each iteration step, the parameter vector $\mathbf{p}$ is replaced by a new estimate $\mathbf{p} + \boldsymbol{\Delta}$ To determine $\boldsymbol{\Delta}$ the function $f(t_i, \mathbf{p} + \boldsymbol{\Delta})$ is approximated by its linearization:

$$f(t_i, \mathbf{p} + \boldsymbol{\Delta}) \approx f(t_i, \mathbf{p}) + \mathbf{J}_i \boldsymbol{\Delta}$$

where

$$\mathbf{J}_i = \frac{\partial f\left(t_i, \mathbf{p}\right)}{\partial \mathbf{p}}$$

is the gradient of f with respect to $\mathbf{p}$.

So, for our problem $\forall j \leq p$:

$$\mathbf{J}_{ij} = \frac{\partial f\left(t_i, \mathbf{p}\right)}{\partial \lambda_{\mathbf{j}}} = \exp(\omega_j t_i),$$

$$\mathbf{J}_{ij+p} = \frac{\partial f\left(t_i, \mathbf{p}\right)}{\partial \omega_{\mathbf{j}}} = \lambda_j t_i \exp(\omega_j t_i).$$

The loss function has its minimum at a zero gradient with respect to $\mathbf{p}$. The above first-order approximation of $f\left(t_i, \mathbf{p} + \boldsymbol{\Delta}\right)$ gives

$$L\left(\mathbf{p} + \boldsymbol{\Delta}\right) \approx \sum_{i=1}^{p}\left[y_i - f\left(t_i, \mathbf{p}\right) - \mathbf{J}_i \boldsymbol{\Delta}\right]^2$$

or in vector notation,

$$\begin{aligned} L\left(\mathbf{p} + \boldsymbol{\Delta}\right) &\approx \left\|\mathbf{y} - \mathbf{f}\left(\mathbf{p}\right) - \mathbf{J}\boldsymbol{\Delta}\right\|_2^2 \\ &= \left[\mathbf{y} - \mathbf{f}\left(\mathbf{p}\right) - \mathbf{J}\boldsymbol{\Delta}\right]^{\mathrm{T}}\left[\mathbf{y} - \mathbf{f}\left(\mathbf{p}\right) - \mathbf{J}\boldsymbol{\Delta}\right] \\ &= \left[\mathbf{y} - \mathbf{f}\left(\mathbf{p}\right)\right]^{\mathrm{T}}\left[\mathbf{y} - \mathbf{f}\left(\mathbf{p}\right)\right] - \left[\mathbf{y} - \mathbf{f}\left(\mathbf{p}\right)\right]^{\mathrm{T}}\mathbf{J}\boldsymbol{\Delta} - \left(\mathbf{J}\boldsymbol{\Delta}\right)^{\mathrm{T}}\left[\mathbf{y} - \mathbf{f}\left(\mathbf{p}\right)\right] + \boldsymbol{\Delta}^{\mathrm{T}}\mathbf{J}^{\mathrm{T}}\mathbf{J}\boldsymbol{\Delta} \\ &= \left[\mathbf{y} - \mathbf{f}\left(\mathbf{p}\right)\right]^{\mathrm{T}}\left[\mathbf{y} - \mathbf{f}\left(\mathbf{p}\right)\right] - 2\left[\mathbf{y} - \mathbf{f}\left(\mathbf{p}\right)\right]^{\mathrm{T}}\mathbf{J}\boldsymbol{\Delta} + \boldsymbol{\Delta}^{\mathrm{T}}\mathbf{J}^{\mathrm{T}}\mathbf{J}\boldsymbol{\Delta}. \end{aligned}$$

Taking the derivative of this approximation of $L\left(\mathbf{p} + \boldsymbol{\Delta}\right)$ with respect to Δ and setting the result to zero gives

$$\left(\mathbf{J}^{\mathrm{T}}\mathbf{J}\right)\boldsymbol{\Delta} = \mathbf{J}^{\mathrm{T}}\left[\mathbf{y} - \mathbf{f}\left(\mathbf{p}\right)\right],$$

The above expression obtained for $\mathbf{p}$ comes under the Gauss–Newton method. The Jacobian matrix as defined above is not (in general) a square matrix, but a rectangular matrix of size $m \times n$, where n is the number of parameters (size of the vector $\mathbf{p}$). The matrix multiplication $\boldsymbol{J}^T\boldsymbol{J}$ yields the required $n \times n$ square matrix and the matrix-vector product on the right hand side yields a vector of size n. The result is a set of n linear equations, which can be solved for $\boldsymbol{\Delta}$.

Levenberg's contribution is to replace this equation by a "damped version":

$$\left(\mathbf{J}^{\mathrm{T}}\mathbf{J} + \lambda\mathbf{E}\right)\boldsymbol{\Delta} = \mathbf{J}^{\mathrm{T}}\left[\mathbf{y} - \mathbf{f}\left(\mathbf{p}\right)\right],$$

The (non-negative) damping factor λ is adjusted at each iteration. If reduction of L is rapid, a smaller value can be used, bringing the algorithm closer to the Gauss–Newton algorithm:

$$\boldsymbol{\Delta} \approx [\mathbf{J}^T\mathbf{J}]^{-1}\mathbf{J}^T[\mathbf{y} - \mathbf{f}\left(\mathbf{p}\right)]$$

whereas if an iteration gives insufficient reduction in the residual, λ can be increased, giving a step closer to the gradient-descent direction:

$$\boldsymbol{\Delta} \approx \lambda^{-1}\mathbf{J}^T[\mathbf{y} - \mathbf{f}\left(\mathbf{p}\right)]$$

To make the solution scale invariant Marquardt's algorithm solved a modified problem with each component of the gradient scaled according to the curvature. This provides larger movement along the directions where the gradient is smaller, which avoids slow convergence in the direction of small gradient. Fletcher in his 1971 paper *A modified Marquardt subroutine for non-linear least squares* simplified the form, replacing the identity matrix E with the diagonal matrix consisting of the diagonal elements of $\mathbf{J}^T\mathbf{J}$:

$$\left[\mathbf{J}^{\mathrm{T}}\mathbf{J} + \lambda\operatorname{diag}\left(\mathbf{J}^{\mathrm{T}}\mathbf{J}\right)\right]\boldsymbol{\Delta} = \mathbf{J}^{\mathrm{T}}\left[\mathbf{y} - \mathbf{f}\left(\boldsymbol{p}\right)\right].$$

Choice of damping parameter

An effective strategy for the control of the damping parameter, called delayed gratification, consists of increasing the parameter by a small factor for each uphill step, and decreasing by a large factor for each downhill step. The idea behind this strategy is to avoid moving downhill too fast in the beginning of optimization, therefore restricting the steps available in future iterations and therefore slowing down convergence.

Implementation details

As performance of implementation relied only on theoretical approaches was not good enough to use it as a solution, in resulting implementation few improvements were made.

Loss function

Loss function was changed to χ^2 loss since it is often used to curve-fitting problems. It is defined as:

$$\chi^2(\boldsymbol{p}) = \sum_{i=1}^{n} \left(\frac{y_i - f(t_i, \boldsymbol{p})}{\sigma_i} \right)^2 = [\mathbf{y} - \mathbf{f}(\mathbf{p})]^T \boldsymbol{W} [\mathbf{y} - \mathbf{f}(\mathbf{p})],$$

where $\boldsymbol{W} = \operatorname{diag}\left(\frac{1}{\sigma_1^2}, \ldots, \frac{1}{\sigma_n^2}\right)$ is a weight matrix: $\sigma_i^2 = \mathbb{D}[y_i]$. In practice, it is used to give more weight to the measurements with smaller errors.

The update formula for $\boldsymbol{\Delta}$ is adjusted accordingly to reflect the change in loss function::

$$\left(\mathbf{J}^{\mathrm{T}} \boldsymbol{W} \mathbf{J} + \lambda \mathbf{E}\right) \boldsymbol{\Delta} = \mathbf{J}^{\mathrm{T}} \boldsymbol{W} [\mathbf{y} - \mathbf{f}(\boldsymbol{p})].$$

Step acceptance

Previously, step was accepted if loss function decreased, else it was rejected and damping parameter was increased. Now, the step is accepted if the metric ρ is greater than a user-defined threshold $\epsilon_4 > 0$ (`step-acceptance` in code). This metric is a measure of actual reduction in χ^2 compared to the the improvement of an LMA step.

$$\begin{aligned} \rho &= \frac{\chi^2(\boldsymbol{p}) - \chi^2(\boldsymbol{p} + \boldsymbol{\Delta})}{|(\boldsymbol{y} - \hat{\boldsymbol{y}})^T \mathbf{W} (\boldsymbol{y} - \hat{\boldsymbol{y}}) - (\boldsymbol{y} - \hat{\boldsymbol{y}} - \mathbf{J}\boldsymbol{\Delta})^T \mathbf{W} (\boldsymbol{y} - \hat{\boldsymbol{y}} - \mathbf{J}\boldsymbol{\Delta})|} \\ &= \frac{\chi^2(\boldsymbol{p}) - \chi^2(\boldsymbol{p} + \boldsymbol{\Delta})}{|\boldsymbol{\Delta}^T (\lambda \boldsymbol{\Delta} + \mathbf{J}^T \mathbf{W} (\boldsymbol{y} - \hat{\boldsymbol{y}}))|} \end{aligned}$$

where $\hat{\boldsymbol{y}} = \mathbf{f}(\boldsymbol{p})$.

This metric of step acceptance was proposed by H.B. Nielson in his 1999 paper [3].

Chosen value for ϵ_4 is 10^{-1}.

Update strategy

Damping parameter and model parameters are updated according to the following rules:

If $\rho > \epsilon_4$: $\lambda = \max[\lambda / L_\downarrow,\ 10^{-7}],\ \mathbf{p} \leftarrow \mathbf{p} + \boldsymbol{\Delta}$
otherwise: $\lambda = \min[\lambda L_\uparrow,\ 10^{7}]$

where $L_\downarrow \approx 9$ and $L_\uparrow \approx 11$ are fixed constants (`REG_DECREASE_FACTOR` and `REG_INCREASE_FACTOR` in code). These values were chosen based on the paper [2].

Convergence criteria

The algorithm stops when *one* of the following conditions is satisfied:

- Convergence in the gradient norm: $\max |\mathbf{J}^T \mathbf{W} (\boldsymbol{y} - \hat{\boldsymbol{y}})| < \epsilon_1$ (`gradient_tol` in code)
- Convergence in coefficients: $\max |\boldsymbol{\Delta} / \mathbf{p}| < \epsilon_2$ (`coefficients_tol` in code)
- Convergence in (reduced) χ^2: $\chi^2_\nu = \chi^2 / (m - n) < \epsilon_3$ (`chi2_red_tol` in code)

where $\epsilon_1 = 10^{-3}$, $\epsilon_2 = 10^{-3}$, $\epsilon_3 = 10^{-1}$ are user-defined thresholds.

Initial guess

In nonlinear least squares problems the $\chi^2(\mathbf{p})$ loss function may have multiple local minima. In such cases, the LMA may converge to a poor fit. If this happens, the user can try to provide a better initial guess for the parameters, for example by random/grid search or data inspection.

References

1. Wikipedia contributors. *Levenberg–Marquardt algorithm*. Wikipedia, The Free Encyclopedia..
2. H.P. Gavin, *The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems*. 2020.
3. H.B. Nielson, *Damping Parameter in Marquardt's method*. 1999.