Optimization and Least Squares Optimization in Pure Julia

Jiawei Li[1], Patrick K Mogensen[2]

[1]University of Birmingham
[2]University of Copenhagen

Abstract

`Optim.jl` and `LsqFit.jl` are parts of the `JuliaNLSolvers` family. Both packages provide a range of optimization and other capabilities written in the Julia programming language. While `Optim.jl` solves general optimization problems, `LsqFit.jl` solves the minimization problems in squared forms, such problems are also known as least squares, which makes it possible to perform non-linear regressions and assess the goodness of fit. This poster will demonstrate the usage of both packages as well as the future plans of `LsqFit.jl`.

Introduction

Optimization is the minimization or maximization of a function $f(x)$. The problem could be written as:

$$\min_{x} \quad f(x)$$

For least squares optimization, the problem has a special objective function. Assume the relationship between independent variable $\mathbf{x_i}$ and dependent variable Y_i follows:

$$Y_i = m(\mathbf{x_i}, \boldsymbol{\gamma}) + \epsilon_i$$

where m is a non-linear model function. We choose the parameter $\boldsymbol{\gamma}$ which minimizes the sum of squared residuals from our data:

$$\min_{\gamma} \quad s(\boldsymbol{\gamma}) = \sum_{i=1}^{n} [m(\mathbf{x_i}, \boldsymbol{\gamma}) - y_i]^2$$

Solving least squares problem does not require second condition check and parts of the Hessian matrix could be derived for free. Least squares algorithms exploit these features to achieve better performance and therefore it is reasonable to split these two types of optimization into two packages.

Optimization

`Optim.jl` solves optimization problems. The package supports optimization on manifolds, functions of complex numbers, and input types such as arbitrary precision vectors and matrices. Users can provide derivatives themselves, or request that they are calculated using automatic differentiation or finite difference methods. The main focus of the package has currently been on unconstrained optimization, however, box-constrained optimization is supported, and a more comprehensive support for constraints is under development.

Example

To solve the Hosaki's problem which has the form:

$$f(\mathbf{x}) = \left(1 - 8x_1 + 7x_1^2 - \frac{7}{3}x_1^3 + \frac{1}{4}x_1^4\right) x_2^2 e^{-x_2}$$

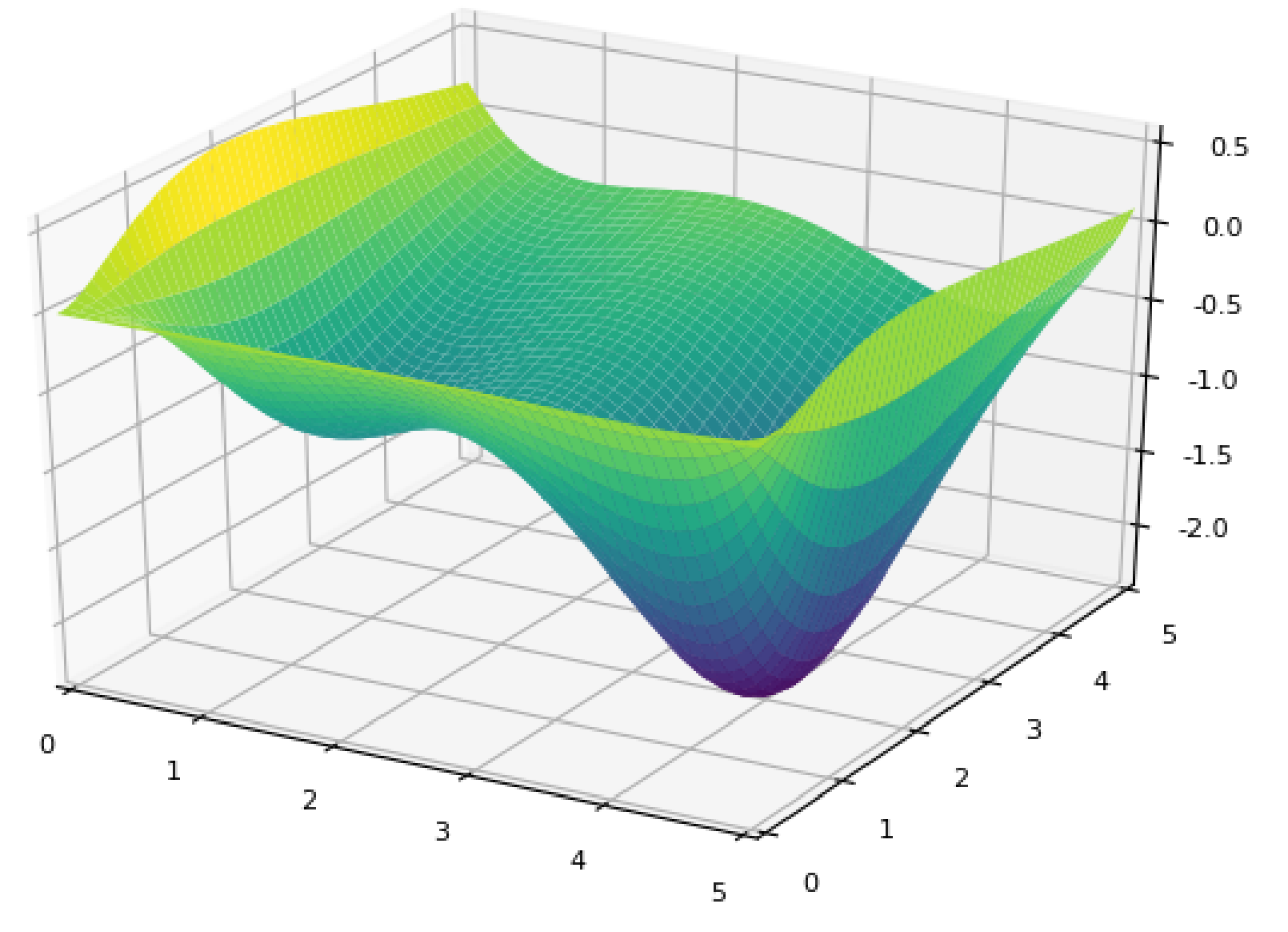

Figure 1: Hosaki's Problem

We first need define the function `f(x)` in code, then pass the function, initial points and method. `optimize(f, [0.0, 0.0], LBFGS())`.

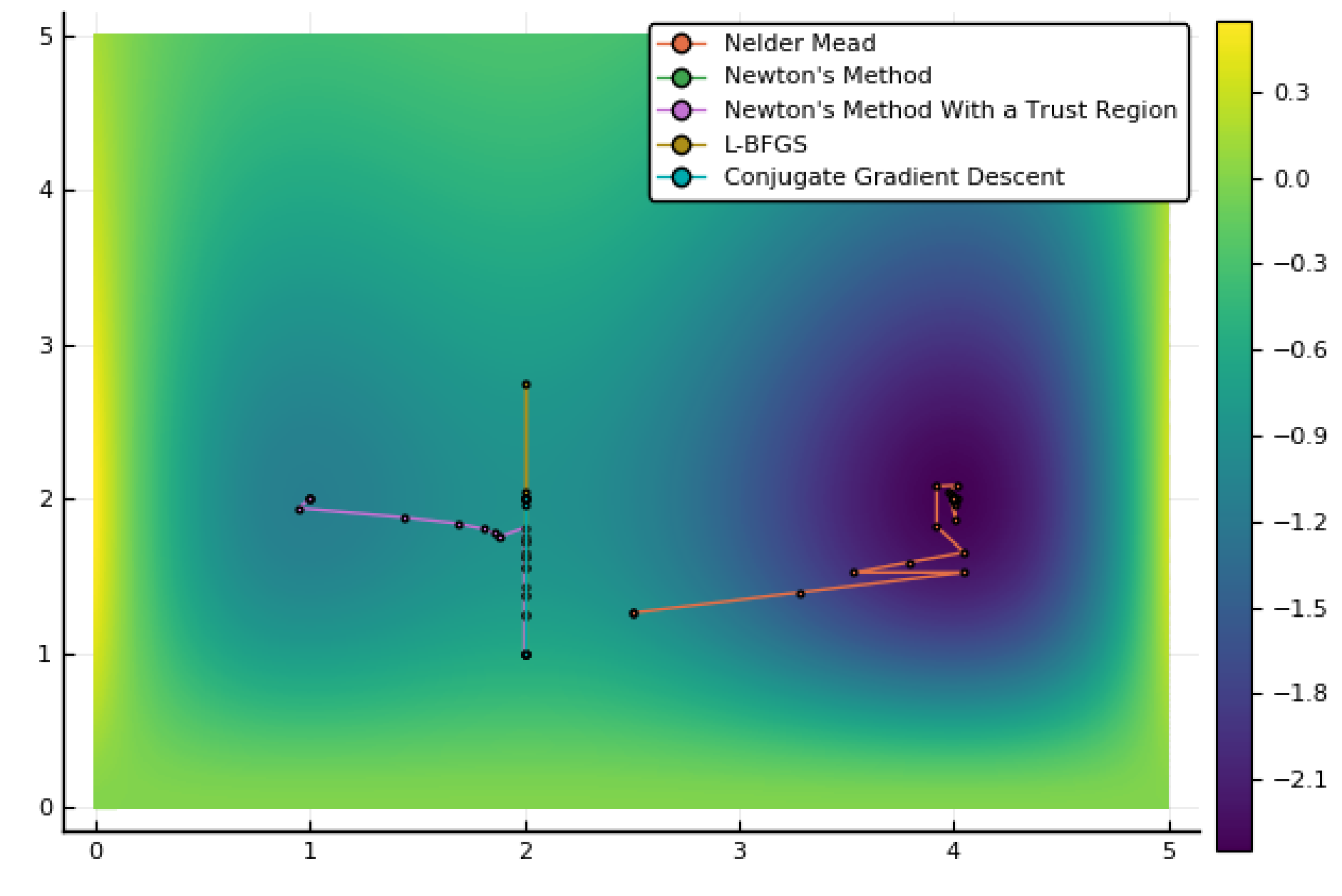

Figure 2: The Trace of Optimization

Least Squares Optimization

`LsqFit.jl` performs non-linear regression which requires least squares optimization to find the best fit. The package also supports Weighted and General Least Squares by providing weights. By approximating the non-linear function m as a linear function, the package could assess the goodness of fit, i.e. estimate covariance, standard errors and confidence intervals of each parameter. At this time, `LsqFit.jl` only utilizes the Levenberg-Marquardt algorithm.

Example

One non-linear model is the exponential decay model, which takes a one-element predictor variable t. The model function is:

$$m(t, \boldsymbol{\gamma}) = -\gamma_1 \exp(\gamma_2 t)$$

and the model becomes:

$$Y_i = -\gamma_1 \exp(\gamma_2 t_i) + \epsilon_i$$

To fit data using `LsqFit.jl`, pass the defined model function `m`, data and the initial parameter value to `curve_fit()`.

```
p_init = [0.5, 0.5]
m(t, p)= -p[1] * exp.(p[2] * t)
fit = curve_fit(m, tdata, ydata, p_init)
```

To exam the goodness of fit, `estimate_covar(fit)` and `standard_error(fit)` compute the estimated covariance matrix and standard errors respectively.

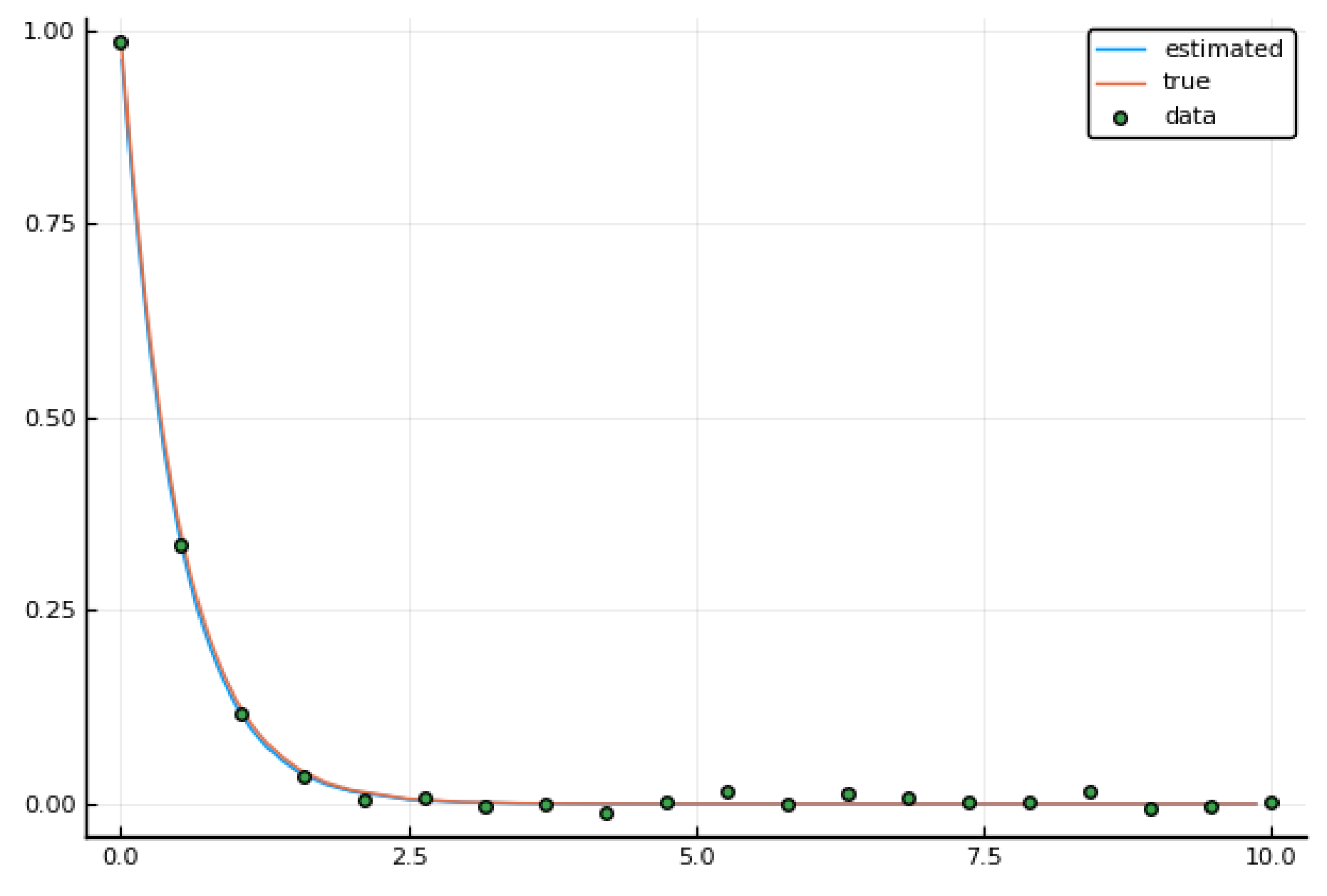

Figure 3: Least Squares Regression

Current Development of LsqFit

`LsqFit.jl` is currently under intense development and reconstruction. The main goal of the reconstruction is to keep the same interface as `Optim.jl` and involve functionalities from `NLSolversBase.jl` and `LineSearches.jl`. More solvers, for example, Gauss-Newton method will also be added.

There will be a new function `least_squares()` which behaves similar to `optimize()` but accept only least squares algorithms. `curve_fit()` will then keep the same interface. For example, to pass `x_tol`, the code will be:

```
curve_fit(m, tdata, ydata, p_init,
LsqFit.Options(x_tol = 1e-8))
```

To pass algorithm's parameters, the code will be:

```
curve_fit(model, tdata, ydata, p0,
LevenbergMarquardt(min_step_quality = 1e-3))
```

Conclusion

Optimization is key to many scientific problems. Our aim is to enable researchers, users, and other Julia packages to solve optimization problems without writing such algorithms themselves. While `Optim.jl` serves general optimization purpose, `LsqFit.jl` solves least squares and performs non-linear least squares regression. Current development of `LsqFit.jl` will build a similar interface to `Optim.jl` and take the package to a new level.

References

[1] Patrick K Mogensen and AsbjÃÿrn N Riseth.
Optim: A mathematical optimization package for julia.
3(24):615.

[2] Per Christian Hansen, V. Pereyra, and Godela Scherer.
Least squares data fitting with applications.

[3] Sanford Weisberg.
Applied linear regression.

[4] Jorge Nocedal and Stephen J. Wright.
Numerical optimization.