CS 240H Project Report:
Linear Least Squares Modeling in Haskell

Tri Dao
trid@stanford.edu

March 18, 2016

1 Introduction

Least squares are optimization problems that minimize the norm square of an affine expression, subject to linear constraints on the variables. They have the general form

$$\begin{array}{ll} \text{minimize} & \|Ax - b\|^2 \\ \text{subject to} & Cx = d \end{array} \tag{1}$$

where $x \in \mathbb{R}^n$ is a variable, $A \in \mathbb{R}^{m\times n}$ and $C \in \mathbb{R}^{p\times n}$ are matrices, and $b \in \mathbb{R}^m$ and $d \in \mathbb{R}^p$ are vectors.

These problems are routinely solved in the context of statistical estimation, machine learning, and engineering design. For example, linear regression, an approach for modeling the relationship between a scalar dependent variable y and one or more explanatory variables X, involves solving the problem minimize $\|X\beta - y\|^2$ to find β, where X is a given data matrix, y is a given vector, and β is the *parameter vector*.

In general, there can be many variables, the objective can contain many quadratic terms, and there can be multiple linear constraints. Typically one has to rewrite the problem in the standard form (1) and then use a general solver implemented by many numerical linear algebra packages. However, this is a tedious and error-prone process, especially for problems with many variables or several linear constraints.

We aim to develop a modeling language embedded in Haskell to solve these linearly constraints least squares problems. This modeling language (in Haskell) will allow user to specify the problem in a natural way that mirrors standard mathematical notation without being constrained by the standard form.

2 Background

2.1 Solution to the least squares problem

The (unconstrained) *least squares problem* has the standard form

$$\text{minimize} \quad \|Ax - b\|^2,$$

where $x \in \mathbb{R}^n$ is a variable, $A \in \mathbb{R}^{m\times n}$ is a matrix, and $b \in \mathbb{R}^m$ is a vector. Assuming that A is full rank, the least squares problem has the close-form solution

$$\hat{x} = (A^T A)^{-1} A^T b.$$

A slight generalization is the (linearly) *constrained least squares problem*:

$$\begin{array}{ll} \text{minimize} & \|Ax - b\|^2 \\ \text{subject to} & Cx = d \end{array}$$

where $x \in \mathbb{R}^n$ is a variable, $A \in \mathbb{R}^{m\times n}$ and $C \in \mathbb{R}^{p\times n}$ are matrices, and $b \in \mathbb{R}^m$ and $d \in \mathbb{R}^p$ are vectors. The solution of this constrained problem is

$$\begin{bmatrix} \hat{x} \\ z \end{bmatrix} = \begin{bmatrix} 2A^T A & C^T \\ C & 0 \end{bmatrix}^{-1} \begin{bmatrix} 2A^T b \\ d \end{bmatrix},$$

where z is the dual variable to the equality constraint. We assume that the matrix above is invertible. This occurs when the matrix C has independent rows, and the matrix $\begin{bmatrix} A \\ C \end{bmatrix}$ has independent columns.

2.2 Numerical linear algebra in Haskell

We use the package `hmatrix` [1] to handle vectors and matrices. This is a purely functional interface to linear algebra and other numerical algorithms, internally implemented using LAPACK, BLAS, and GSL.

The types provided (`Matrix` and `Vector`) are dense, immutable and strict in all elements (unboxed). We will only use the double precision real versions of these types, i.e. `Matrix R` and `Vector R` (`R=Double`). The matrix product is `(<>)`, the matrix-vector product is `(#>)`, the dot product is `(<.>)`, and the (conjugate) transpose is `tr`.

3 Modeling

The key data type is a linear expression `LinExpr`, which contains variables and constants that are combined using linear mathematical functions (addition, scaling, matrix-vector multiplication, etc.). A linear constraint (type `LinContr`) can be form as `LinExpr :==: LinExpr`. A quadratic expression `QuadExpr` can be formed by taking square norm (`SumSquares`) of linear expressions, or sum of many such square norms.

The package provides a function `minimize` to solve a least squares problem:

```
minimize :: QuadExpr -> [LinContr] -> M.Map VarName (Vector R)
```

The function takes a quadratic expression as objective, a list of linear constraints, and returns a map from variable names (type alias for `String`) to variable values.

For example, the classic problem of find the least norm solution to an under-determined system can be easily setup and solved with the following code:

```
import Data.Map.Strict ((!))
import Numeric.LinearAlgebra hiding ((!))
import LeastSquares

leastSquaresEstimate :: Vector R
leastSquaresEstimate = result ! "x"
  where result = minimize (SumSquares x) [c * x :==: d]
        c = Mat $ (2><3) [3.0, 7.0, -1.5, 2.8, 10.0, -5.3]
        d = Vec $ vector [-1.0, 3.5]
        x = Var "x" 3
```

4 Implementation

Given a problem with a quadratic expression as objective and a list of linear constraints, the system will analyze the abstract syntax tree to transform it to the standard form in (1), and then solve it using linear algebra subroutines.

The three steps of solving a least squares problem is given in Figure 1.

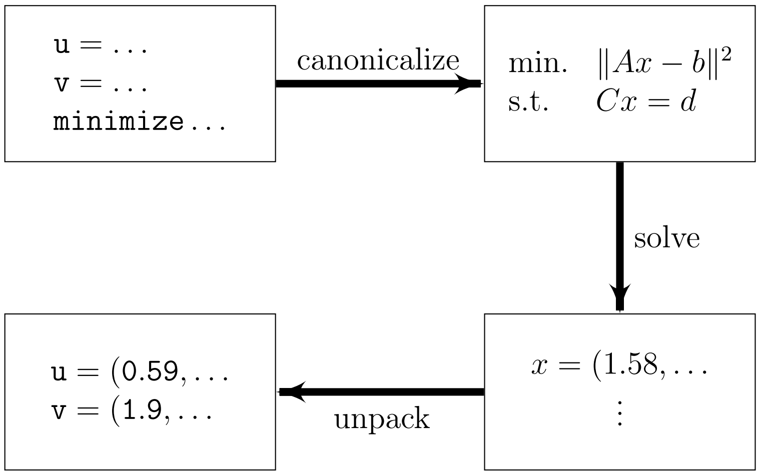


Figure 1: Three steps of solving a least squares problem.

In the canonicalization step, we stack all the variables in the problem as one big variable, and then stack the linear constraints. For example, given constraints $Cu + Dv = w$ and $Eu = z$, we produce the equivalent constraint

$$\begin{bmatrix} C & D \\ E & 0 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} w \\ z \end{bmatrix}.$$

Since the objective is a sum of square norms, we convert this sum to a single equivalent square norm. For example, if the objective is $\|Au - b\|^2 + 2\|v\|^2$, the equivalent square norm is

$$\left\| \begin{bmatrix} A & 0 \\ 0 & \sqrt{2}I \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\|^2,$$

where I is the identity matrix. We then form a new variable $x = \begin{bmatrix} u \\ v \end{bmatrix}$ and thus both the objective and the constraints are transformed to standard form.

In the solve step, we simply call the least squares solver provided by `hmatrix`. The unpacking step takes the resulting vector and assign the values to the original variables.

5 Example

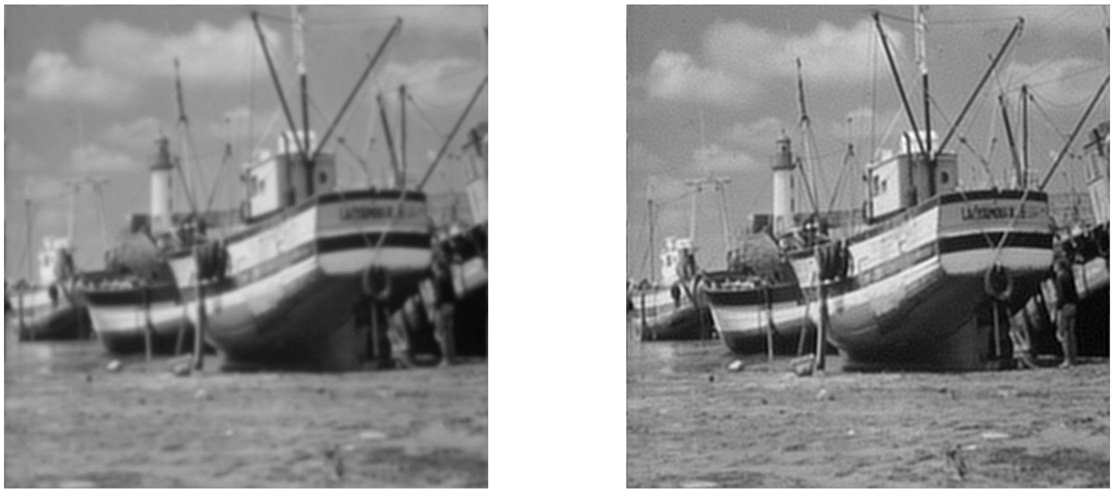
We consider an example of least squares modeling in image de-blurring. Suppose that x is an image, A is a blurring operator, and $y = Ax + v$ is a blurred, noisy image that we are given. We choose x to minimize

$$\|Ax - y\|^2 + \lambda(\|D_{\mathrm{v}}x\|^2 + \|D_{\mathrm{h}}x\|^2),$$

where D_{v}, D_{h} are vertical and horizontal differencing operations and the scalar parameter λ controls smoothing of de-blurred image. The problem can be expressed in this natural form:

```
minimize (SumSquares (a*x-b) + lambda * (SumSquares (dv*x) + SumSquares (dh*x))) []
```

We show the burred, noisy image and the recovered image with $\lambda = 0.007$ in Figure 2.



(a) Blurred, noisy image (b) Recovered image

Figure 2: Blurred, noisy image and regularized inversion with $\lambda = 0.007$.

6 Conclusion

We have designed and implemented a modeling language embedded in Haskell for linearly constrained least squares problem. There are still some work to be done to add more linear operators (vector indexing, convolution, Fourier transform, vector and matrix stacking, element wise multiplication, etc.). Moreover, the package only supports dense matrices due to lack of sparse matrix support in Haskell.

Overall, this modeling language will allow users to express their problems in a natural form while the system takes care of transforming it to the standard form. This might lead to faster prototyping and data analysis in statistical estimation, machine learning, and engineering design.

Acknowledgment

We thank Professor Stephen Boyd for introducing us to many examples of least squares modeling in control, statistical estimation, and engineering design. We also thank David Zeng whose Julia package `LinearLeastSquares` [2] inspired this project.

References

[1] Alberto Ruiz. hmatrix: Linear algebra and numerical computation. `http://hackage.haskell.org/package/hmatrix-0.17.0.1`, Sep 2015.

[2] David Zeng. LinearLeastSquares.jl: Least squares solver in Julia. `https://github.com/davidlizeng/LinearLeastSquares.jl`, Nov 2014.