In [19]:

```julia
using CSV
using DataFrames
using GLM
using Optim
using Statistics
using ForwardDiff
using NLopt
```

In [12]:

```julia
function fminunc(obj, x; tol = 1e-08)
    results = Optim.optimize(obj, x, LBFGS(),
                            Optim.Options(
                            g_tol = tol,
                            x_tol=tol,
                            f_tol=tol))
    return results.minimizer, results.minimum, Optim.converged(results)
    #xopt, objvalue, flag = fmincon(obj, x, tol=tol)
    #return xopt, objvalue, flag
end
```

Out[12]: fminunc (generic function with 1 method)

In [20]:

```julia
function fmincon(obj, startval, R=[], r=[], lb=[], ub=[]; tol = 1e-10, iterlim=0)
    # the objective is an anonymous function
    function objective_function(x::Vector{Float64}, grad::Vector{Float64})
        obj_func_value = obj(x)[1,1]
        return(obj_func_value)
    end
    # impose the linear restrictions
    function constraint_function(x::Vector, grad::Vector, R, r)
        result = R*x .- r
        return result[1,1]
    end
    opt = Opt(:LN_COBYLA, size(startval,1))
    min_objective!(opt, objective_function)
    # impose lower and/or upper bounds
    if lb != [] lower_bounds!(opt, lb) end
    if ub != [] upper_bounds!(opt, ub) end
    # impose linear restrictions, by looping over the rows
    if R != []
        for i = 1:size(R,1)
            equality_constraint!(opt, (theta, g) -> constraint_function(theta, g, R[i:i
        end
    end
    xtol_rel!(opt, tol)
    ftol_rel!(opt, tol)
    maxeval!(opt, iterlim)
    (objvalue, xopt, flag) = NLopt.optimize(opt, startval)
    return xopt, objvalue, flag
end
```

Out[20]: fmincon (generic function with 5 methods)

The Nerlove Model

Theoretical Background

For a firm that takes input prices w and the output level q as given, the cost minimization problem is to choose the quantities of inputs x to solve the problem

$$\min_x w'x$$

subject to the restriction

$$f(x) = q.$$

The solution is the vector of factor demands $x(w,q)$. The cost function is obtained by substituting the factor demands into the criterion function:

$$Cw, q) = w'x(w,q).$$

- **Monotonicity** Increasing factor prices cannot decrease cost, so

  $$\frac{\partial C(w,q)}{\partial w} \geq 0$$

  Remember that these derivatives give the conditional factor demands (Shephard's Lemma).
- **Homogeneity** The cost function is homogeneous of degree 1 in input prices: $C(tw,q) = tC(w,q)$ where t is a scalar constant. This is because the factor demands are homogeneous of degree zero in factor prices - they only depend upon relative prices.
- **Returns to scale** The returns to scale parameter γ is defined as the inverse of the elasticity of cost with respect to output:

  $$\gamma = \left(\frac{\partial C(w,q)}{\partial q}\frac{q}{C(w,q)}\right)^{-1}$$

  Constant returns to scale is the case where increasing production q implies that cost increases in the proportion 1:1. If this is the case, then $\gamma = 1$.

Cobb-Douglas functional form

The Cobb-Douglas functional form is linear in the logarithms of the regressors and the dependent variable. For a cost function, if there are g factors, the Cobb-Douglas cost function has the form

$$C = Aw_1^{\beta_1} \ldots w_g^{\beta_g} q^{\beta_q} e^{\varepsilon}$$

What is the elasticity of C with respect to w_j?

$$\begin{aligned} e_{w_j}^C &= \left(\frac{\partial C}{\partial_{W_J}}\right)\left(\frac{w_j}{C}\right) \\ &= \beta_j A w_1^{\beta_1} . w_j^{\beta_j - 1} . . w_g^{\beta_g} q^{\beta_q} e^{\varepsilon} \frac{w_j}{Aw_1^{\beta_1} \ldots w_g^{\beta_g} q^{\beta_q} e^{\varepsilon}} \\ &= \beta_j \end{aligned}$$

This is one of the reasons the Cobb-Douglas form is popular - the coefficients are easy to interpret, since they are the elasticities of the dependent variable with respect to the explanatory variable. Not that in this case,

$$
\begin{aligned}
e^C_{w_j} &= \left(\frac{\partial C}{\partial W_J}\right)\left(\frac{w_j}{C}\right) \\
&= x_j(w,q)\frac{w_j}{C} \\
&\equiv s_j(w,q)
\end{aligned}
$$

the cost share of the j^{th} input. So with a Cobb-Douglas cost function, $\beta_j = s_j(w,q)$. The cost shares are constants.

Note that after a logarithmic transformation we obtain

$$\ln C = \alpha + \beta_1 \ln w_1 + \ldots + \beta_g \ln w_g + \beta_q \ln q + \epsilon$$

where $\alpha = \ln A$. So we see that the transformed model is linear in the logs of the data.

One can verify that the property of HOD1 implies that

$$\sum_{i=1}^{g} \beta_i = 1$$

In other words, the cost shares add up to 1.

The hypothesis that the technology exhibits CRTS implies that

$$\gamma = \frac{1}{\beta_q} = 1$$

so $\beta_q = 1$. Likewise, monotonicity implies that the coefficients $\beta_i \geq 0, i = 1, \ldots, g$.

The Nerlove Data

The file contains data on 145 electric utility companies' cost of production, output and input prices. The data are for the U.S., and were collected by M. Nerlove. The observations are by row, and the columns are

- COMPANYCOST (C)
- OUTPUT (Q)
- PRICE OF LABOR (P_L)
- PRICE OF FUEL (P_F)
- PRICE OF CAPITAL(P_K)

Note that the data are sorted by output level (the third column).

We will estimate the Cobb-Douglas model

$$\ln C = \beta_1 + \beta_Q \ln Q + \beta_L \ln P_L + \beta_F \ln P_F + \beta_K \ln P_K + \epsilon$$

by OLS.

OLS

In [7]:
```
data = DataFrame(CSV.File("../data/nerlove.csv"))
first(data,6)
```

Out[7]: 6 rows × 6 columns

	firm	cost	output	labor	fuel	capital
	Int64	Float64	Int64	Float64	Float64	Int64
1	101	0.082	2	2.09	17.9	183
2	102	0.661	3	2.05	35.1	174
3	103	0.99	4	2.05	35.1	171
4	104	0.315	4	1.83	32.2	166
5	105	0.197	5	2.12	28.6	233
6	106	0.098	9	2.12	28.6	195

In [8]:
```
data = log.(data[:,[:cost,:output,:labor,:fuel,:capital]])
first(data,6)
```

Out[8]: 6 rows × 5 columns

	cost	output	labor	fuel	capital
	Float64	Float64	Float64	Float64	Float64
1	-2.50104	0.693147	0.737164	2.8848	5.20949
2	-0.414001	1.09861	0.71784	3.5582	5.15906
3	-0.0100503	1.38629	0.71784	3.5582	5.14166
4	-1.15518	1.38629	0.604316	3.47197	5.11199
5	-1.62455	1.60944	0.751416	3.35341	5.45104
6	-2.32279	2.19722	0.751416	3.35341	5.273

In [4]:
```
ols = lm(@formula(cost~output+labor+fuel+capital),data)
```

Out[4]:
```
StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Array{Float64,1}},GLM.DensePredChol{Float64,LinearAlgebra.CholeskyPivoted{Float64,Array{Float64,2}}}},Array{Float64,2}}

cost ~ 1 + output + labor + fuel + capital

Coefficients:
──────────────────────────────────────────────────────────────────────────
                 Coef.  Std. Error      t  Pr(>|t|)  Lower 95%   Upper 95%
──────────────────────────────────────────────────────────────────────────
(Intercept)  -3.5265     1.77437    -1.99    0.0488  -7.03452   -0.0184845
output        0.720394   0.0174664  41.24    <1e-79   0.685862   0.754926
labor         0.436341   0.291048    1.50    0.1361  -0.139076   1.01176
fuel          0.426517   0.100369    4.25    <1e-4    0.228082   0.624952
capital      -0.219888   0.339429   -0.65    0.5182  -0.890957   0.45118
──────────────────────────────────────────────────────────────────────────
```

In [5]:
```
n = size(data,1)
y = data[:,1]
x = data[:,2:end]
```

```
x[!,:intercept]=ones(size(data,1))
x = x[!,[:intercept,:output,:labor,:fuel,:capital]]

y = convert(Array,y)
x = convert(Array,x)
inv(x'*x)*x'*y
```

Out[5]:
```
5-element Array{Float64,1}:
 -3.5265028449802216
  0.7203940758797012
  0.4363412007892406
  0.4265169530627446
 -0.2198883507567723
```

MLE

In [7]:
```
function normal(theta, y, x)
    b = theta[1:end-1]
    s = theta[end][1]
    e = (y - x*b)./s
    logdensity = -log.(sqrt.(2.0*pi)) .- 0.5*log(s.^2) .- 0.5*e.*e
end
```

Out[7]:
```
normal (generic function with 1 method)
```

In [13]:
```
function mle(model, θ)
    avg_obj = θ -> -mean(vec(model(θ))) # average log likelihood
    thetahat, objvalue, converged = fminunc(avg_obj, θ) # do the minimization of -logL
    objvalue = -objvalue
    obj = θ -> vec(model(θ)) # unaveraged log likelihood
    n = size(obj(θ),1) # how many observations?
    scorecontrib = ForwardDiff.jacobian(obj, vec(thetahat))
    I = cov(scorecontrib)
    J = ForwardDiff.hessian(avg_obj, vec(thetahat))
    Jinv = inv(J)
    V= Jinv*I*Jinv/n
    return thetahat, objvalue, V, converged
end
```

Out[13]:
```
mle (generic function with 1 method)
```

In [15]:
```
theta = [zeros(size(x,2)); 1.0] # start values for estimation
model = theta -> normal(theta, y, x)
thetahat, objvalue, V, converged = mle(model, theta)
```

Out[15]:
```
([-3.5265029527984506, 0.7203941307911956, 0.4363410601373892, 0.4265167999256887, -0.21
988830342608368, 0.3855314736645991], -0.4658061612351275, [2.871544188031368 -0.0133506
64827278202 … -0.5341833292561572 0.00758992898888696; -0.01335066482727622 0.0010330820
163605126 … 0.0014703347896175387 -0.000888618606090073; … ; -0.5341833292561352 0.00147
03347896179168 … 0.10194167672632491 -0.0009345831820626936; 0.00758992898888676 -0.0008
88618606090018 … -0.0009345831820626361 0.0017343512126735234], true)
```

In [16]:
```
thetahat
```

Out[16]:
```
6-element Array{Float64,1}:
 -3.5265029527984506
  0.7203941307911956
  0.4363410601373892
  0.426516799256887
```

```
-0.21988830342608368
 0.3855314736645991
```

In [17]:

```
converged
```

Out[17]:

```
true
```

Restricted Nerlove

In [16]:

```
# prepare the data
n = size(data,1)
y = data[:,1]
x = data[:,2:end]
x[!,:intercept]=ones(size(data,1))
x = x[!,[:intercept,:output,:labor,:fuel,:capital]]

y = convert(Array,y)
x = convert(Array,x)

# bounds and restriction
lb = [-1e6, -1e6, 0., 0., 0.0]
ub = [1e6, 1e6, 1., 1., 1.]
R = [0. 0. 1. 1. 1.]
r = 1.0

# define the objective function and start value
obj = theta -> (y-x*theta)'*(y-x*theta)
startval = (ub+lb)/2.0

# OLS
thetahat, objvalue = fminunc(obj, startval)
println("the OLS estimates: obj. value: ", round(objvalue,digits=5))
```

```
┌ Warning: `convert(::Type{Array}, df::AbstractDataFrame)` is deprecated, use `Matrix(d
f)` instead.
│   caller = top-level scope at In[16]:9
└ @ Core In[16]:9
the OLS estimates: obj. value: 21.55201
```

In [22]:

```
# restricted LS
thetahat, objvalue_r, flag = fmincon(obj, startval, R, r, lb, ub) # both lower and uppe
println("the restricted LS estimates: obj. value: ", round(objvalue_r,digits=5))
```

```
the restricted LS estimates: obj. value: 22.28018
```