In [1]:
```julia
using CSV
using DataFrames
using GLM
using Optim
using Statistics
using ForwardDiff
using NLopt
using StatsFuns
using LinearAlgebra
```

```
[ Info: Precompiling GLM [38e38edf-8417-5370-95a0-9cbb8c7f171a]
└ @ Base loading.jl:1278
```

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:

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

- **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

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

What is the elasticity of C with respect to w_j?

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

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,

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

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

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

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 $\beta_i \geq 0, i = 1, \dots, g$.

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.

In [2]:
```
data = DataFrame(CSV.File("../data/nerlove.csv"))
first(data,6)
```

Out[2]: 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 [3]:
```
data = log.(data[:,[:cost,:output,:labor,:fuel,:capital]])
first(data,6)
```

Out[3]: 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]:
```
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)
```

Out[4]:

```
145×5 Array{Float64,2}:
 1.0  0.693147  0.737164  2.8848   5.20949
 1.0  1.09861   0.71784   3.5582   5.15906
 1.0  1.38629   0.71784   3.5582   5.14166
 1.0  1.38629   0.604316  3.47197  5.11199
 1.0  1.60944   0.751416  3.35341  5.45104
 1.0  2.19722   0.751416  3.35341  5.273
 1.0  2.3979    0.683097  3.56953  5.32788
 1.0  2.56495   0.71784   3.5582   5.01064
 1.0  2.56495   0.783902  3.37074  5.04343
 1.0  3.09104   0.542324  2.70805  5.23644
 1.0  3.21888   0.737164  2.8848   5.1358
 1.0  3.21888   0.518794  3.68135  5.11799
 1.0  3.55535   0.593327  3.11795  5.36129
 ⋮
 1.0  8.72193   0.652325  3.11352  5.07517
 1.0  8.88086   0.751416  3.35341  5.0876
 1.0  8.97284   0.476234  2.8792   5.18178
 1.0  9.03825   0.841567  3.46261  5.2933
 1.0  9.06439   0.806476  3.27714  5.20401
 1.0  9.08103   0.837248  3.51155  5.24702
 1.0  9.15736   0.746688  3.19458  5.10595
 1.0  9.20593   0.518794  3.36038  5.31321
 1.0  9.3481    0.806476  3.27714  5.01728
 1.0  9.37552   0.751416  3.35341  4.99721
 1.0  9.57213   0.837248  3.51155  5.35659
 1.0  9.7243    0.832909  3.16125  5.0876
```

In [5]:

```
inv(x'*x)*x'*y
```

Out[5]:

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

OLS

In [6]:

```
ols = lm(@formula(cost~output+labor+fuel+capital),data)
```

Out[6]:

```
StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Array{Float64,1}},GLM.DensePredC
hol{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

MLE

In [7]:
```
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[7]: fminunc (generic function with 1 method)

In [8]:
```
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[8]: normal (generic function with 1 method)

In [9]:
```
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[9]: mle (generic function with 1 method)

In [10]:
```
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[10]:
```
([-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 [11]:
```
thetahat
```

Out[11]:
```
6-element Array{Float64,1}:
 -3.5265029527984506
  0.7203941307911956
```

```
 0.4363410601373892
 0.4265167999256887
-0.21988830342608368
 0.3855314736645991
```

In [12]:
```
objvalue
```

Out[12]:
```
-0.4658061612351275
```

In [13]:
```
converged
```

Out[13]:
```
true
```

GMM

In [98]:
```
function gmm(moments, theta, weight)
    # average moments
    m = theta -> vec(mean(moments(theta),dims=1)) # 1Xg
    # moment contributions
    momentcontrib = theta -> moments(theta) # nXg
    # GMM criterion
    obj = theta -> ((m(theta))'weight*m(theta))
    # do minimization
    thetahat, objvalue, converged = fminunc(obj, theta)
    # derivative of average moments
    D = (ForwardDiff.jacobian(m, vec(thetahat)))'
    # moment contributions at estimate
    ms = momentcontrib(thetahat)
    return thetahat, objvalue, D, ms, converged
end
```

Out[98]:
```
gmm (generic function with 1 method)
```

In [99]:
```
weight = 1
theta = zeros(size(x,2))
moments = theta -> (y .- x*theta).*x
thetahat1, junk, junk, ms, junk = gmm(moments, theta, weight)
```

Out[99]:
```
([-3.526502843629834, 0.7203940758779137, 0.4363412007234413, 0.4265169530669924, -0.219
88835101027032], 2.0315912495823984e-25, [-1.0 -6.556651068379128 … -3.2088584232140143
-5.15677677573767; -6.556651068379128 -46.62321518989734 … -20.92418272643975 -33.792349
77799892; … ; -3.2088584232140143 -20.92418272643975 … -10.424693444784769 -16.552050639
682335; -5.15677677573767 -33.79234977799892 … -16.552050639682335 -26.60235530942714],
[0.11956154515690232 0.08287374792889741 … 0.3449112306976858 0.622854213907207; 1.62462
76007326357 1.7848358466742609 … 5.780751765522584 8.38154363280989; … ; 0.8830924883738
982 8.453078045568063 … 3.1010193996152986 4.730361102489577; 0.7151025229149148 6.95387
2233201762 … 2.260615499330251 3.6381529748973525], true)
```

In [100…
```
thetahat1
```

Out[100…
```
5-element Array{Float64,1}:
 -3.526502843629834
  0.7203940758779137
  0.4363412007234413
  0.4265169530669924
 -0.21988835101027032
```

In [101…
```
W = inv(cov(ms))
thetahat2, junk, junk, ms, junk = gmm(moments, theta, W)
```

Out[101… ([-3.526502845237955, 0.7203940758869178, 0.43634120077645283, 0.42651695306397513, -0.21988835071590623], 4.523502175795952e-22, [-1.0 -6.556651068379128 … -3.2088584232140143 -5.15677677573767; -6.556651068379128 -46.62321518989734 … -20.92418272643975 -33.79234977799892; … ; -3.2088584232140143 -20.92418272643975 … -10.424693444784769 -16.552050639682335; -5.15677677573767 -33.79234977799892 … -16.552050639682335 -26.60235530942714], [0.11956154519492257 0.08287374795525106 … 0.34491123080736663 0.6228542141052729; 1.6246276007849059 1.7848358467316856 … 5.7807517657085725 8.381543633079556; … ; 0.8830924882852536 8.453078044719545 … 3.1010193993040187 4.730361102014744; 0.7151025229032566 6.953872233088394 … 2.2606154992933964 3.63815297483804], true)

In [102…

```
thetahat2
```

Out[102…

```
5-element Array{Float64,1}:
 -3.526502845237955
  0.7203940758869178
  0.43634120077645283
  0.42651695306397513
 -0.21988835071590623
```

Restricted Nerlove

In [35]:

```
"""
    xopt, fopt, converged = fminunc(obj, startval)

Minimize the function obj, starting at startval.

fminunc() with no arguments will run an example, execute edit(fminunc,()) to see the co
fminunc() uses NLopt.jl  to do the actual minimization.

"""
function fminunc(obj, x; tol = 1e-10)
    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[35]: fminunc

In [37]:

```
# define the objective function and start value
obj = theta -> (y-x*theta)'*(y-x*theta)
startval = [-1e6, -1e6, 0., 0., 0.0]

# OLS
thetahat, objvalue = fminunc(obj, startval)
```

Out[37]: ([-3.526502845286827, 0.7203940758804234, 0.4363412008175227, 0.4265169530531963, -0.21988835069604426], 21.55200816418578, true)

In [42]:

```
"""
    xopt, fopt, converged = fmincon(obj, startval)

Minimize the function obj, starting at startval.

fminunc() with no arguments will run an example, execute edit(fminunc,()) to see the co
fminunc() uses NLopt.jl to do the actual minimization.
```

```julia
"""
function fmincon(obj, startval, R=[], r=[], lb=[], ub=[]; tol = 1e-25, 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[42]: fmincon

In [43]:

```julia
# 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

# restricted LS
thetahat, objvalue_r, flag = fmincon(obj, startval, R, r, lb, ub) # both lower and uppe
```

Out[43]: ([-6.048596329679246, 0.7206760132410218, 0.3250348901994794, 0.33416246746988226, 0.34080264233063823], 22.20823582015495, :XTOL_REACHED)

In [41]:

```julia
sum(thetahat[3:end])
```

Out[41]: 1.0000000000000002

Restricted OLS

For the review purpose, please refer to this notes for OLS in Matrix Form

The general formulation of linear equality restrictions is the model

$$y = X\beta + \varepsilon$$
$$R\beta = r$$

where R is a $Q \times K$ matrix, $Q < K$ and r is a $Q \times 1$ vector of constants.

Let's consider how to estimate β subject to the restrictions $R\beta = r$. The most obvious approach is to set up the Lagrangean

$$\min_{\beta,\lambda} s(\beta,\lambda) = \frac{1}{n}(y - X\beta)'(y - X\beta) + 2\lambda'(R\beta - r).$$

The Lagrange multipliers are scaled by 2, which makes things less messy. The fonc are

$$\begin{aligned} D_\beta s(\hat\beta,\hat\lambda) &= -2X'y + 2X'X\hat\beta_R + 2R'\hat\lambda \equiv 0 \\ D_\lambda s(\hat\beta,\hat\lambda) &= R\hat\beta_R - r \equiv 0, \end{aligned}$$

which can be written as

$$\begin{bmatrix} X'X & R' \\ R & 0 \end{bmatrix} \begin{bmatrix} \hat\beta_R \\ \hat\lambda \end{bmatrix} = \begin{bmatrix} X'y \\ r \end{bmatrix}.$$

Re-arragne:

$$\begin{bmatrix} \hat\beta_R \\ \hat\lambda \end{bmatrix} = \begin{bmatrix} X'X & R' \\ R & 0 \end{bmatrix}^{-1} \begin{bmatrix} X'y \\ r \end{bmatrix}.$$

and define that $P = R(X'X)^{-1}R'$:

$$\begin{aligned} \begin{bmatrix} \hat\beta_R \\ \hat\lambda \end{bmatrix} &= \begin{bmatrix} (X'X)^{-1} - (X'X)^{-1}R'P^{-1}R(X'X)^{-1} & (X'X)^{-1}R'P^{-1} \\ P^{-1}R(X'X)^{-1} & -P^{-1} \end{bmatrix} \begin{bmatrix} X'y \\ r \end{bmatrix} \\ &= \begin{bmatrix} \hat\beta - (X'X)^{-1}R'P^{-1}\left(R\hat\beta - r\right) \\ P^{-1}\left(R\hat\beta - r\right) \end{bmatrix} \\ &= \begin{bmatrix} (I_K - (X'X)^{-1}R'P^{-1}R) \\ P^{-1}R \end{bmatrix} \hat\beta + \begin{bmatrix} (X'X)^{-1}R'P^{-1}r \\ -P^{-1}r \end{bmatrix} \end{aligned}$$

The fact that $\hat\beta_R$ and $\hat\lambda$ are linear functions of $\hat\beta$ makes it easy to determine their distributions, since the distribution of $\hat\beta$ is already known. Recall that for x a random vector, and for A and b a matrix and vector of constants, respectively, $Var\left(Ax + b\right) = AVar(x)A'$.

In [27]:

```
function ols(y::Array{Float64}, x::Array{Float64,2}; R=[], r=[], vc="white", silent=fal

    # compute ols coefficients, fitted values, and errors
    function lsfit(y, x)
        beta = inv(x'*x)*x'*y
        fit = x*beta
        errors = y - fit
        return beta, fit, errors
    end

    n,k = size(x)
    b, fit, e = lsfit(y,x)
    df = n-k
    sigsq = (e'*e/df)[1,1]
    xx_inv = inv(x'*x)
    ess = (e' * e)[1,1]
```

```
    # Restricted LS
    if R !=[]
        q = size(R,1)
        P_inv = inv(R*xx_inv*R')
        b = b .- xx_inv*R'*P_inv*(R*b.-r)
        e = y-x*b;
        ess = (e' * e)[1,1]
        df = n-k-q
        sigsq = ess/df
        A = Matrix{Float64}(I, k, k) .- xx_inv*R'*P_inv*R;  # the matrix relating b and
    end

    xe = x.*e
    varb = xx_inv*xe'xe*xx_inv

    # restricted LS?
    if R !=[]
        varb = A*varb*A'
    end

    # common to both ordinary and restricted
    seb = sqrt.(diag(varb))
    seb = seb.*(seb.>1e-16) # round off to zero when there are restrictions
    t = b ./ seb
    tss = y .- mean(y)
    tss = (tss'*tss)[1,1]
    rsq = (1.0 - ess / tss)

    p = 2.0 .- 2.0*tdistcdf.(df, abs.(t))

    return b, seb, t, p
end
```

Out[27]: ols (generic function with 1 method)

Let's decompose the code:

In [6]:
```
function lsfit(y, x)
    beta = inv(x'*x)*x'*y
    fit = x*beta
    errors = y - fit
    return beta, fit, errors
end
```

Out[6]: lsfit (generic function with 1 method)

In [7]:
```
n,k = size(x)
b, fit, e = lsfit(y,x)
df = n-k
sigsq = (e'*e/df)[1,1]
xx_inv = inv(x'*x)
ess = (e' * e)[1,1]
```

Out[7]: 21.55200816418577

In [11]:
```
(e'*e/df)
```

Out[11]: 0.1539429154584698

In [12]:
```
# Set restrictions:
R = [0 0 1 1 1]
r = 1
```

Out[12]:
```
1
```

In [21]:
```
# If restricted:
q = size(R,1)
P_inv = inv(R*xx_inv*R')
b = b .- xx_inv*R'*P_inv*(R*b.-r)
e = y-x*b;
ess = (e' * e)[1,1]
df = n-k-q
sigsq = ess/df
A = Matrix{Float64}(I, k, k) .- xx_inv*R'*P_inv*R
```

Out[21]:
```
5×5 Array{Float64,2}:
 1.0  0.0   3.26103       3.26103       3.26103
 0.0  1.0  -0.000821913  -0.000821913  -0.000821913
 0.0  0.0   0.56147      -0.43853      -0.43853
 0.0  0.0   0.033738      1.03374       0.033738
 0.0  0.0  -0.595208     -0.595208      0.404792
```

In [18]:
```
xe = x.*e
varb = xx_inv*xe'xe*xx_inv
```

Out[18]:
```
5×5 Array{Float64,2}:
  2.79541     -0.0124238     -0.139666     0.0235275    -0.522476
 -0.0124238    0.00102379    -0.00145555  -0.000238609   0.00131859
 -0.139666    -0.00145555     0.060153    -0.00876526    0.0270589
  0.0235275   -0.000238609   -0.00876526   0.00560044   -0.00651251
 -0.522476     0.00131859     0.0270589   -0.00651251    0.100198
```

In [19]:
```
# If restricted:
varb = A*varb*A'
```

Out[19]:
```
5×5 Array{Float64,2}:
  0.645676     -0.0136316     0.125187     -0.00872577   -0.116461
 -0.0136316     0.00102454   -0.00128702   -0.000248581   0.0015356
  0.125187     -0.00128702    0.0277956    -0.0046787    -0.0231169
 -0.00872577   -0.000248581  -0.0046787     0.00516317   -0.000484464
 -0.116461      0.0015356    -0.0231169    -0.000484464   0.0236014
```

In [22]:
```
seb = sqrt.(diag(varb))
seb = seb.*(seb.>1e-16) # round off to zero when there are restrictions
t = b ./ seb
tss = y .- mean(y)
tss = (tss'*tss)[1,1]
rsq = (1.0 - ess / tss)
```

Out[22]:
```
0.9256517157418225
```

Finally, estimate with restricted OLS:

In [30]:
```
(b, seb, t, p) = ols(y, x, R=R, r=r)
```

Out[30]:
```
([-4.690789123000792, 0.7206875237539881, 0.592909608392978, 0.4144714553168333, -0.0073
81063709811425], [0.8035399132401216, 0.03200841460252987, 0.16672022985133889, 0.071855
18151252615, 0.15362745919807486], [-5.837655411647293, 22.515564507122658, 3.5563147251
036287, 5.768149861879906, -0.048045211112258755], [3.568130679809656e-8, 0.0, 0.0005147
311810049793, 4.981561319006289e-8, 0.9617491746445415])
```

In [31]:
```
b
```

Out[31]:
```
5-element Array{Float64,1}:
 -4.690789123000792
  0.7206875237539881
  0.592909608392978
  0.4144714553168333
 -0.007381063709811425
```

In [32]:
```
seb
```

Out[32]:
```
5-element Array{Float64,1}:
 0.8035399132401216
 0.03200841460252987
 0.16672022985133889
 0.07185518151252615
 0.15362745919807486
```

In [33]:
```
p
```

Out[33]:
```
5-element Array{Float64,1}:
 3.568130679809656e-8
 0.0
 0.0005147311810049793
 4.981561319006289e-8
 0.9617491746445415
```

In [34]:
```
sum(b[3:end])
```

Out[34]:
```
0.9999999999999999
```

Test Statistics: Linear Models with Restrictions

Wald Statistic

Recall that wald test statistic:

$$W \equiv n\mathbf{a}(\hat{\theta})'\left[\mathbf{A}(\hat{\theta})\hat{\mathbf{\Sigma}}^{-1}\mathbf{A}(\hat{\theta})'\right]^{-1}\mathbf{a}(\hat{\theta})$$

is asymptotically $\chi^2(r)$ under the null hypothesis.

The t and F tests require normality of the errors. The Wald test does not, but it is an asymptotic test - it is only approximately valid in finite samples.

The Wald principle is based on the idea that if a restriction is true, the unrestricted model should approximately'' satisfy the restriction. Given that the least squares estimator is asymptotically normally distributed:

$$\sqrt{n}\left(\hat{\beta}-\beta_0\right) \xrightarrow{d} N\left(0, \sigma_0^2 Q_X^{-1}\right)$$

then under $H_0: R\beta_0 = r$, we have

$$\sqrt{n}\left(R\hat{\beta}-r\right) \xrightarrow{d} N\left(0, \sigma_0^2 R Q_X^{-1} R'\right)$$

because if the n dimensional random vector $x \sim N(0, V)$, then $x'V^{-1}x \sim \chi^2(n)$.

$$n\left(R\hat{\beta} - r\right)'\left(\sigma_0^2 R Q_X^{-1} R'\right)^{-1}\left(R\hat{\beta} - r\right) \xrightarrow{d} \chi^2(q)$$

Note that Q_X^{-1} or σ_0^2 are not observable. The test statistic we use substitutes the consistent estimators. Use $(X'X/n)^{-1}$ as the consistent estimator of Q_X^{-1}. With this, there is a cancellation of $n's$, and the statistic to use is

$$\left(R\hat{\beta} - r\right)'\left(\widehat{\sigma_0^2} R(X'X)^{-1} R'\right)^{-1}\left(R\hat{\beta} - r\right) \xrightarrow{d} \chi^2(q)$$

Let's look at an example:

In [47]:
```
n,k = size(x)
q = size(R,1)
b = x\y
xx_inv = inv(x'*x)
P_inv = inv(R*xx_inv*R')
b_r = b .- xx_inv*R'*P_inv*(R*b.-r)
e = y - x*b
ess = (e'*e)[1]
e_r = y - x*b_r
ess_r = (e_r' * e_r)[1]
sigsqhat = ess/(n)
sigsqhat_r = ess_r/(n);
```

In [48]:
```
W = (R*b.-r)'*P_inv*(R*b.-r)/sigsqhat
```

Out[48]:
```
0.5941482584878846
```

In [50]:
```
chisqccdf(q,W)
```

Out[50]:
```
0.44081948121548903
```

Likelihood Ratio Multiplier (LR) Statistic

Reference Hayashi 7.4 for details on derivations.

$$\begin{aligned} LR &\equiv 2n\left[Q_n(\hat{\theta}) - Q_n(\tilde{\theta})\right] &(1)\\ &= -n(\hat{\theta} - \tilde{\theta})'\Psi(\hat{\theta} - \tilde{\theta}) + o_p &(2)\end{aligned}$$

is asymptotically $\chi^2(r)$ under the null hypothesis.

From this expression, deriving the asymptotic distribution of the LR:

$$\sqrt{n}(\hat{\theta} - \tilde{\theta}) = -\Psi^{-1}A_0'(A_0\Psi^{-1}A_0')^{-1}A_0'\Psi^{-1}\sqrt{n}\frac{\partial Q_n(\theta_0)}{\partial \theta} + o_p$$

Substituting the above equation to LR statistics and setting $\sqrt{n}\frac{\partial Q_n(\theta_0)}{\partial \theta} = g(\theta_0)$ and $\Psi^{-1} = \mathcal{I}(\theta_0)$

$$LR \stackrel{a}{=} n^{1/2}g(\theta_0)'\mathcal{I}(\theta_0)^{-1}R'\left(R\mathcal{I}(\theta_0)^{-1}R'\right)^{-1}R\mathcal{I}(\theta_0)^{-1}n^{1/2}g(\theta_0) \quad (3)$$

Under normality, we have seen that the likelihood function is

$$\ln L(\beta, \sigma) = -n \ln \sqrt{2\pi} - n \ln \sigma - \frac{1}{2} \frac{(y - X\beta)' (y - X\beta)}{\sigma^2}.$$

Using this,

$$\begin{aligned} g(\beta_0) &\equiv D_\beta \frac{1}{n} \ln L(\beta, \sigma) \\ &= \frac{X'(y - X\beta_0)}{n\sigma^2} \\ &= \frac{X'\varepsilon}{n\sigma^2} \end{aligned}$$

Also, by the information matrix equality:

$$\begin{aligned} \mathcal{I}(\theta_0) &= -H_\infty(\theta_0) \\ &= \lim -D_{\beta'} g(\beta_0) \\ &= \lim -D_{\beta'} \frac{X'(y - X\beta_0)}{n\sigma^2} \\ &= \lim \frac{X'X}{n\sigma^2} \\ &= \frac{Q_X}{\sigma^2} \end{aligned}$$

so

$$\mathcal{I}(\theta_0)^{-1} = \sigma^2 Q_X^{-1}$$

Substituting these last expressions:

$$LR \stackrel{a}{=} \varepsilon' X'(X'X)^{-1} R' \left(\sigma_0^2 R(X'X)^{-1} R'\right)^{-1} R(X'X)^{-1} X'\varepsilon$$

In [62]:

```
lnl = -n/2*log(2*pi) - n/2*log(sigsqhat) - ess/(2.0*sigsqhat)
lnl_r = -n/2*log(2*pi) - n/2*log(sigsqhat_r) - ess_r/(2.0*sigsqhat_r)
LR = 2.0*(lnl-lnl_r)
```

Out[62]: `0.5929342902877579`

In [63]:

```
chisqccdf.(q,LR)
```

Out[63]: `0.44128668478235494`

Lagrange Multiplier (LM) Statistic

$$LM \equiv n \left(\frac{\partial Q_n(\tilde{\theta})}{\partial \theta} \right)' \tilde{\Sigma}^{-1} \left(\frac{\partial Q_n(\tilde{\theta})}{\partial \theta} \right) \tag{4}$$

is asymptotically $\chi^2(r)$ under the nuull hypothesis.

We have seen that

$$\begin{aligned}\hat{\lambda} &= \left(R(X'X)^{-1}R'\right)^{-1}\left(R\hat{\beta} - r\right) \\ &= P^{-1}\left(R\hat{\beta} - r\right)\end{aligned}$$

so

$$\sqrt{n}\hat{P}\lambda = \sqrt{n}\left(R\hat{\beta} - r\right)$$

Given that

$$\sqrt{n}\left(R\hat{\beta} - r\right) \xrightarrow{d} N\left(0, \sigma_0^2 R Q_X^{-1} R'\right)$$

under the null hypothesis, we obtain

$$\sqrt{n}\hat{P}\lambda \xrightarrow{d} N\left(0, \sigma_0^2 R Q_X^{-1} R'\right)$$

So

$$\left(\sqrt{n}\hat{P}\lambda\right)'\left(\sigma_0^2 R Q_X^{-1} R'\right)^{-1}\left(\sqrt{n}\hat{P}\lambda\right) \xrightarrow{d} \chi^2(q)$$

Noting that $\lim nP = RQ_X^{-1}R'$, we obtain,

$$\hat{\lambda}'\left(\frac{R(X'X)^{-1}R'}{\sigma_0^2}\right)\hat{\lambda} \xrightarrow{d} \chi^2(q)$$

since the powers of n cancel. To get a usable test statistic substitute a consistent estimator of σ_0^2.

This makes it clear why the test is sometimes referred to as a Lagrange multiplier test. It may seem that one needs the actual Lagrange multipliers to calculate this. If we impose the restrictions by substitution, these are not available. Note that the test can be written as

$$\frac{\left(R'\hat{\lambda}\right)'(X'X)^{-1}R'\hat{\lambda}}{\sigma_0^2} \xrightarrow{d} \chi^2(q)$$

However, we can use the foc for the restricted estimator:

$$-X'y + X'X\hat{\beta}_R + R'\hat{\lambda}$$

to get that

$$\begin{aligned}R'\hat{\lambda} &= X'(y - X\hat{\beta}_R) \\ &= X'\hat{\varepsilon}_R\end{aligned}$$

Substituting this into the above, we get

$$\frac{\hat{\varepsilon}_R' X(X'X)^{-1}X'\hat{\varepsilon}_R}{\sigma_0^2} \xrightarrow{d} \chi^2(q)$$

but this is simply

$$\hat{\varepsilon}_R' \frac{P_X}{\sigma_0^2} \hat{\varepsilon}_R \xrightarrow{d} \chi^2(q).$$

In [51]:
```
P_x = x * xx_inv * x'
S = e_r' * P_x * e_r/(sigsqhat_r)
```

Out[51]: 0.5917236270218078

In [52]:
```
chisqccdf(q,S)
```

Out[52]: 0.4417533757921563

Combine all:

In [64]:
```
function TestStatistics(y, x, R, r; silent=false)
    n,k = size(x)
    q = size(R,1)
    b = x\y
    xx_inv = inv(x'*x)
    P_inv = inv(R*xx_inv*R')
    b_r = b .- xx_inv*R'*P_inv*(R*b.-r)
    e = y - x*b
    ess = (e'*e)[1]
    e_r = y - x*b_r
    ess_r = (e_r' * e_r)[1]
    sigsqhat = ess/(n)
    sigsqhat_r = ess_r/(n)
    # Wald test (uses unrestricted model's est. of sig^2)
    W = (R*b.-r)'*P_inv*(R*b.-r)/sigsqhat
    # LR test
    lnl = -n/2*log(2*pi) - n/2*log(sigsqhat) - ess/(2.0*sigsqhat)
    lnl_r = -n/2*log(2*pi) - n/2*log(sigsqhat_r) - ess_r/(2.0*sigsqhat_r)
    LR = 2.0*(lnl-lnl_r)
    # Score test (uses restricted model's est. of sig^2)
    P_x = x * xx_inv * x'
    S = e_r' * P_x * e_r/(sigsqhat_r)

    tests_label = ["Wald","LR","LM"]
    tests = [W[1], LR[1], S[1]]
    pvalues = chisqccdf.(q,tests)

    return tests_label, tests, pvalues
end
```

Out[64]: TestStatistics (generic function with 1 method)

In [65]:
```
TestStatistics(y, x, R, r)
```

Out[65]: (["Wald", "LR", "LM"], [0.5941482584878846, 0.5929342902877579, 0.5917236270218078], [0.44081948121548903, 0.44128668478235494, 0.4417533757921563])

Hence, $\beta_{labor} + \beta_{fuel} + \beta_{capital} = 1$ is not rejected at the usual significance level.