In [2]:
```
using Distributions
using LinearAlgebra
using Optim
using Plots
using ForwardDiff
using Statistics
```

Method of Moments

A **moment condition**, $\bar{m}_n(\theta) = \bar{m}_n(Z_n, \theta)$ is a vector-valued function of the data Z_n and the parameter θ that has mean zero, under the model, when evaluated at the true parameter value θ_0, and expectation differenet from zero when evaluated at other parameter values:

$$E\Big[\bar{m}_n(Z_n, \theta_0) = 0\Big]$$

$$E\Big[\bar{m}_n(Z_n, \theta) \neq 0, \quad \theta \neq \theta_0\Big]$$

The **method of moments principle** is to choose the estimator of the parameter to set the memonet condition equal to zero: $\bar{m}_n(\hat{\theta}) \equiv 0$.

Example (OLS)

The classical linear model. Let $\bar{m}_n(\beta) = \frac{1}{n}\sum_t x_t(y_t - x_t'\beta)$. So the moment contributions are $m_t(\beta) = x_t(y_t - x_t'\beta)$. When $\beta = \beta_0$, $y_t - x_t'\beta_0 = \epsilon_t$, and $m_t = x_t\epsilon_t$. We know that $E(x_t\epsilon_t) = 0$, by the weak exogeneity assumption. Thus, the moment contributions, and the moment condition, which is their average, have expectation zero when evaluated at the true parameter value.

Example (χ^2)

Suppose we draw a random sample of y_t from the $\chi^2(\theta_0)$ distribution. Here, θ_0 is the parameter of interest.

(Version: mean) If $Y \sim \chi^2(\theta_0)$, then the mean $E(Y) = \theta_0$. Let the moment contribution be

$$m_t(\theta) = y_t - \theta$$

Then

$$\bar{m}_n(\theta) = \frac{1}{n}\sum_{t=1}^{n} m_t(\theta) = \bar{y} - \theta$$

We know that the $E(\bar{y}) = \theta_0$.

- Thus, $E\bar{m}_n(\theta_0) = 0$.
- However, $E\bar{m}_n(\theta) = \theta_0 - \theta \neq$0 if $\theta \neq \theta_0$.

In [34]:
```
using Distributions
n = 10
theta = 3
```

```
# the distribution doesn't matter, what matters is
# that the moments are correctly specified
y = rand(Chisq(theta), n)
#y = rand(Normal(theta, sqrt(2*theta)), n)

# a MM estimator
thetahat = mean(y)
println("mm estimator based on mean: ", thetahat)
```

```
mm estimator based on mean: 2.9555207170453937
```

(Version: variance) The variance of a $\chi^2(\theta_0)$ r.v. is

$$V(Y) = E(Y - \theta_0)^2 = 2\theta_0$$

Let $m_t(\theta) = \frac{n}{n-1}(y_t - \bar{y})^2 - 2\theta$ Then

$$\bar{m}_n(\theta) = \frac{\sum_{t=1}^n (y_t - \bar{y})^2}{n-1} - 2\theta.$$

The first term is the unbiased formula for the sample variance, and thus has expectation equal to $2\theta_0$. So if we evaluate $\bar{m}_n(\theta)$ at θ_0, the expectation is zero.

The MM estimator using the variance would set

$$\bar{m}_n(\hat{\theta}) = \frac{\sum_{t=1}^n (y_t - \bar{y})^2}{n-1} - 2\hat{\theta} \equiv 0.$$

Solving for the estimator, it is half the sample variance:

$$\hat{\theta} = \frac{1}{2}\frac{\sum_{t=1}^n (y_t - \bar{y})^2}{n-1}.$$

Again, by the LLN, the sample variance is consistent for the true variance, that is,

$$\frac{\sum_{t=1}^n (y_t - \bar{y})^2}{n} \xrightarrow{p} 2\theta_0.$$

So, this MM is also consistent for θ_0.

In [37]:
```
n = 10000
theta = 3

# the distribution doesn't matter, what matters is
# that the moments are correctly specified
y = rand(Chisq(theta), n)
#y = rand(Normal(theta, sqrt(2*theta)), n)

thetahat = 0.5*var(y)
println("mm estimator based on variance: ", thetahat)
```

```
mm estimator based on variance: 2.912962793373761
```

- For the χ^2 example, we have two alternative moment conditions and only one parameter: we have **overidentification**, which means that we have more information than is strictly necessary for consistent estimation of the parameter.

To summarize a moment condition is a vector valued function which has expectation zero at the true parameter value. We have seen some examples of where we might get such functions, and more will follow. For now, let's take moment conditions as given, and work out the properties of the estimator

Generalized Method of Moments

For the purposes of this course, the following definition of the GMM estimator is sufficiently general: \begin{defn} \label{GMM estimator (defn)}The GMM estimator of the k -dimensional parameter vector θ_0,

$$\hat{\theta} \equiv \arg\min_{\Theta} \bar{m}_n(\theta)' W_n \bar{m}_n(\theta)$$

- where $\bar{m}_n(\theta) = \frac{1}{n}\sum_{t=1}^{n} m(Z_t, \theta)$ is a g-vector valued function, $g \geq k$, with $\mathcal{E}m(\theta_0) = 0,$
- and W_n converges almost surely to a finite $g \times g$ symmetric positive definite matrix W_∞.

```
In [5]:
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[5]: fminunc (generic function with 1 method)
```

Consistency

```
In [7]:
n = 10000
theta = 3

# the distribution doesn't matter, what matters is
# that the moments are correctly specified
y = rand(Chisq(theta), n)
#y = rand(Normal(theta, sqrt(2*theta)), n)
W = I(2)
obj = theta -> ([theta.-mean(y); theta.-0.5*var(y)]'W*[theta.-mean(y); theta.-0.5*var(y
thetahat, junk, junk = fminunc(obj, [2.0])
println("GMM estimator based on mean and variance: ", thetahat)
```

```
GMM estimator based on mean and variance: [3.001813776289966]
```

What's the reason for using GMM if MLE is asymptotically efficient?

- Robustness: GMM is based upon a limited set of moment conditions. For consistency, only these moment conditions need to be correctly specified, whereas MLE in effect requires correct specification of **every conceivable** moment condition. GMM is \emph{robust with respect to distributional misspecification.} The price for robustness is usually a loss of efficiency with respect to the MLE estimator. Keep in mind that the true distribution is \uline{not known} so if we erroneously specify a distribution and estimate by MLE, the estimator will be inconsistent in general (not always).

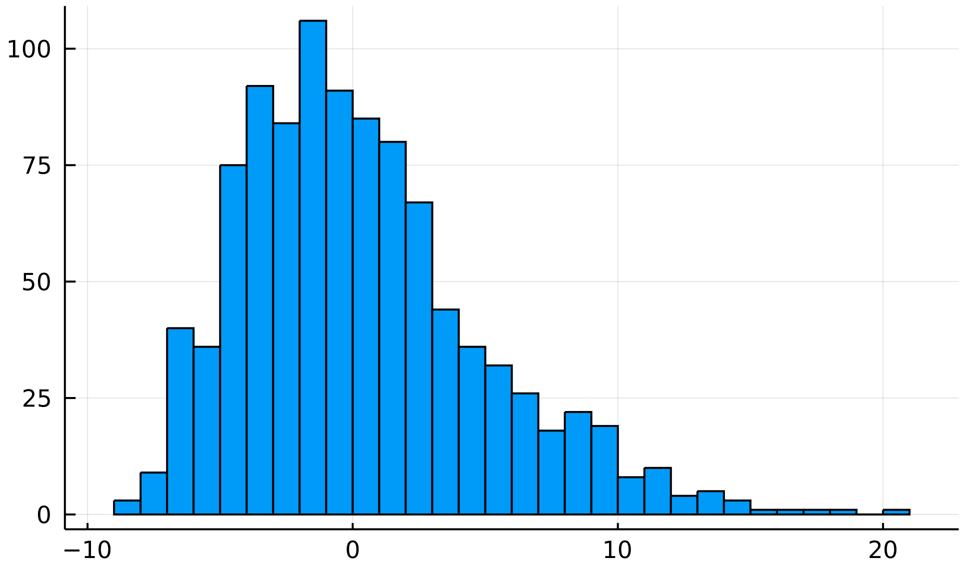
- Feasibility: in some cases the MLE estimator is not available, because we are not able to deduce or compute the likelihood function. More on this in the section on simulation-based estimation. The GMM estimator may still be feasible even though MLE is not available.

Asymptotic Normality

In [60]:

```
n = 30
theta = 3.0
reps = 1000
results = zeros(reps)
W = I(2)
for i = 1:reps
    y = rand(Chisq(theta), n)
    obj = theta -> ([theta.-mean(y); theta.-0.5*var(y)]'W*[theta.-mean(y); theta.-0.5*v
    thetahat, junk, junk = fminunc(obj, [5.0])
    results[i] = sqrt(n)*(thetahat[1]-theta)
end
histogram(results,nbins=50,legend=false)
plot!(size=(500,300))
```

Out[60]:



In [59]:

```
std(results)
```

Out[59]:

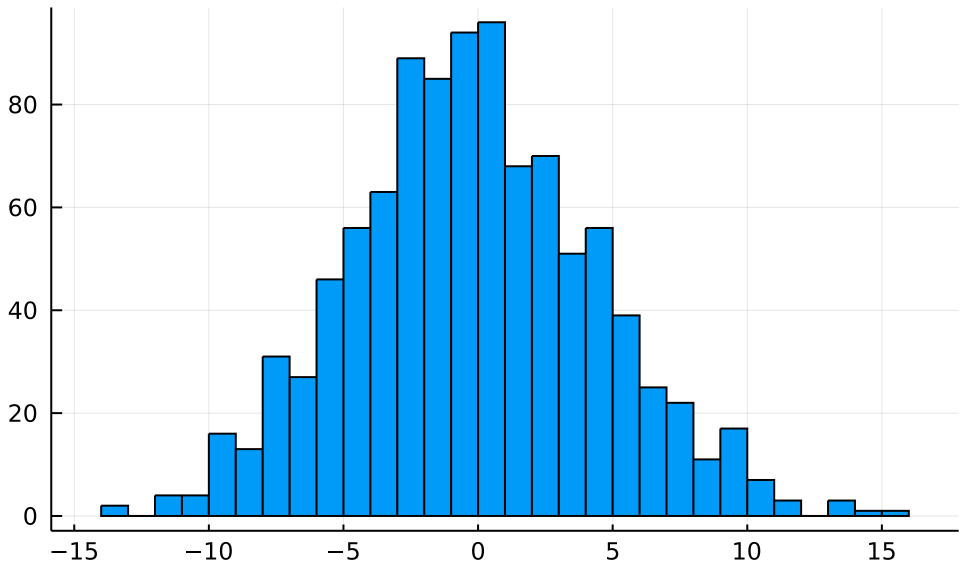
```
4.439578684072703
```

In [9]:

```
n = 10000
theta = 3.0
reps = 1000
results = zeros(reps)
W = I(2)
for i = 1:reps
    y = rand(Chisq(theta), n)
    obj = theta -> ([theta.-mean(y); theta.-0.5*var(y)]'W*[theta.-mean(y); theta.-0.5*v
    thetahat, junk, junk = fminunc(obj, [3.0])
    results[i] = sqrt(n)*(thetahat[1]-theta)
end
```

```
histogram(results,nbins=50,legend=false)
plot!(size=(500,300))
```

Out[9]:



Choosing the Weighting Matrix

W is a weighting matrix, which determines the relative importance of violations of the individual moment conditions. For example, if we are much more sure of the first moment condition, which is based upon the variance, than of the second, which is based upon the fourth moment, we could set

$$W = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$$

with a much larger than b. In this case, errors in the second moment condition have less weight in the objective function.

How to estimate efficient GMM estimator?

Two Step GMM estimator

The most common way to do efficient GMM estimation is the two step GMM estimator:

1. Set the weight matrix to some positive definite matrix. Most commonly, one uses an identity matrix. Obtain the GMM estimator that minimizes $Q_n(\theta) = m_n(\theta)'Wm_n(\theta)$
2. Based on this initial estimate, $\hat{\theta}$, say, compute the moment contributions $m_t(\hat{\theta}),\ t = 1, 2, \ldots, n$. Compute an estimate of Ω_∞ based on the moment contributions, say $\hat{\Omega}^{-1}$. The exact way to do this will depend upon the assumptions of the model. Given the estimate, compute the efficient GMM estimator which minimizes

$$Q_n(\theta) = m_n(\theta)'\hat{\Omega}^{-1}m_n(\theta).$$

Note that $\hat{\Omega}^{-1}$ is fixed while numeric minimization finds the second step estimator. The result is the two step estimator.

In [61]:
```
moments = theta -> [theta.-y  theta.-0.5.*(y .- mean(y)).^2.0]
```

```
Out[61]: #58 (generic function with 1 method)
```

```
In [63]: moments(theta)
```

```
Out[63]: 30×2 Array{Float64,2}:
          -0.312033    2.44138
           1.52427     2.69634
           2.57516     1.3252
           2.13139     2.03892
           0.276464    2.89025
           2.4311      1.57847
           2.84939     0.785711
          -1.24747     1.0151
           2.16011     1.99868
           2.45889     1.53124
           0.535292    2.97802
           2.1395      2.02763
           2.17425     1.97857
           ⋮
           0.96401     2.97601
           0.0498848   2.75843
          -2.93903    -3.7859
          -0.21141     2.54267
           2.09201     2.09274
           1.73282     2.51207
          -0.177703    2.57434
           1.34275     2.82132
           0.527676    2.97639
           2.7592      0.971422
           0.881976    2.99061
          -0.0375437   2.69384
```

```
In [67]: vec(mean(moments(theta),dims=1))
```

```
Out[67]: 2-element Array{Float64,1}:
          0.7449667668335332
          1.4913351161592463
```

```
In [10]: 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[10]: gmm (generic function with 1 method)
```

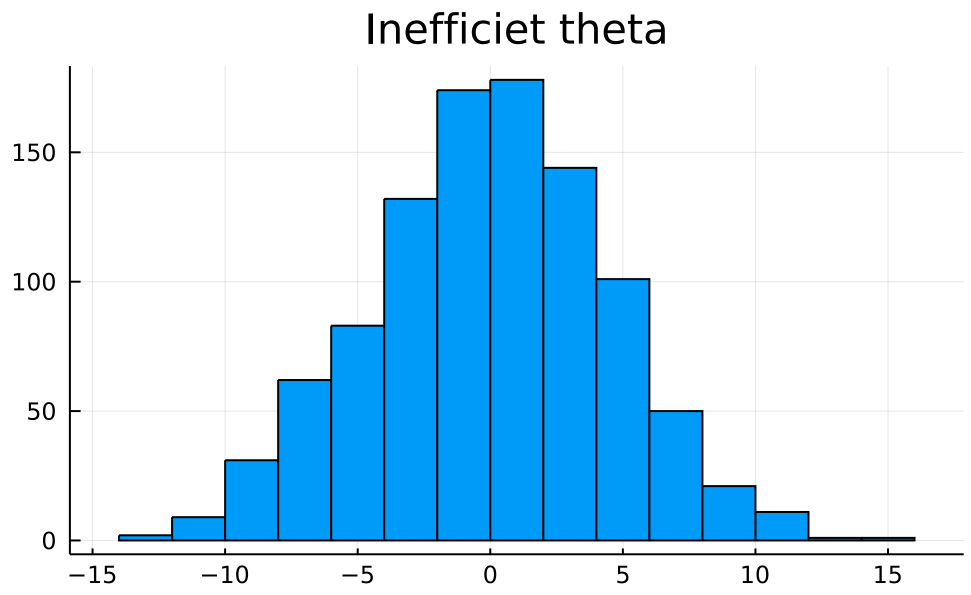
```
In [72]: n = 10000
         theta = 3.0
         reps = 1000
         results = zeros(reps,2)
         for i = 1:1000
```

```
    y = rand(Chisq(theta), n)
    moments = theta -> [theta.-y  theta.-0.5.*(y .- mean(y)).^2.0]
    thetahat, junk, junk, ms, junk = gmm(moments, [3.0], I(2))
    results[i,1] = sqrt(n)*(thetahat[1]-theta)
    W = inv(cov(ms))
    thetahat, junk, ms, junk = gmm(moments, thetahat, W)
    results[i,2] = sqrt(n)*(thetahat[1]-theta)
end
```

In [73]:

```
histogram(results[:,1],legend=false,title="Inefficiet theta")
plot!(size=(500,300))
```

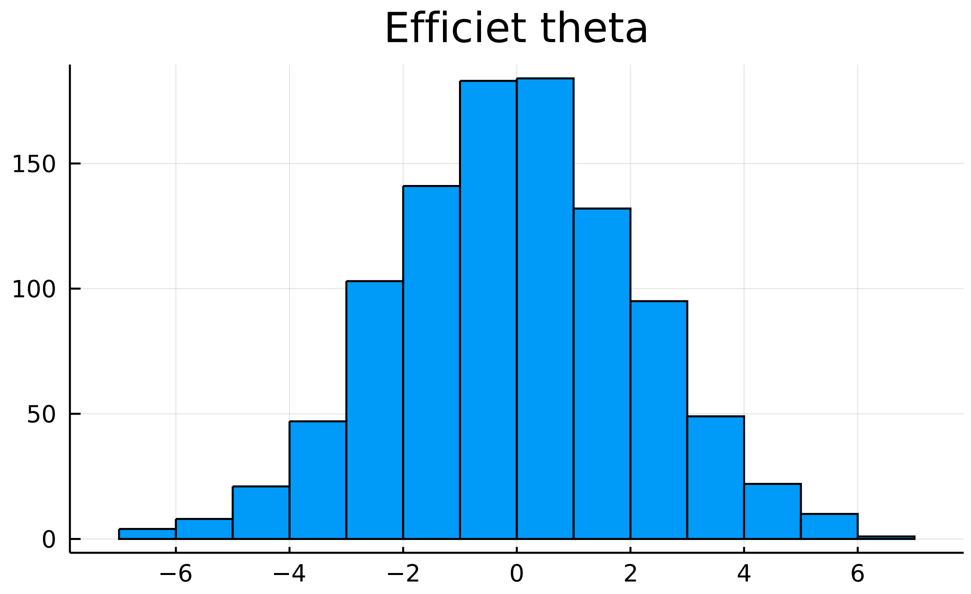
Out[73]:



In [74]:

```
histogram(results[:,2],legend=false,title="Efficiet theta")
plot!(size=(500,300))
```

Out[74]:



Continuously Updated GMM Estimator (CUE)

The continuously updated estimator solves a minimization problem where the efficient weight matrix is estimated at each iteration of the numeric optimization process. The CUE estimator solves the minimization problem

$$Q_n(\theta) = m_n(\theta)'\hat{\Omega}(\theta)^{-1}m_n(\theta).$$

- Note that the covariance of the moment conditions will be updated at each trial value of the objective function during the course of minimization.
- This estimator is equivalent to an iterated version of the two step estimator.
- The CUE estimator can be shown to have a smaller bias than does the two step estimator, which may have a large small sample bias.

Generalized instrumental variables estimator for linear models

The IV estimator may appear a bit unusual at first, but it will grow on you over time.

Let's look at the previous section's results in more detail, for the commonly encountered special case of a linear model with iid errors, but with correlation between regressors and errors:

$$\begin{aligned} y_t &= x_t'\theta + \varepsilon_t \\ \mathcal{E}(x_t\varepsilon_t) &\neq 0 \end{aligned}$$

- Let's assume, just to keep things simple, that the errors are iid
- The model in matrix form is $y = X\theta + \epsilon$

We have seen some cases where this problem arises:

- measurement error of regressors
- lagged dependent variable and autocorrelated errors
- simultaneous equations

Let $K = dim(x_t)$. Consider some vector z_t of dimension $G \times 1$, where $G \geq K$. Assume that $E(z_t\epsilon_t) = 0$. The variables z_t are **instrumental variables**.

Consider the moment conditions

$$\begin{aligned} m_t(\theta) &= z_t\epsilon_t \\ &= z_t\left(y_t - x_t'\theta\right) \end{aligned}$$

We can arrange the instruments in the $n \times G$ matrix

$$Z = \begin{bmatrix} z_1' \\ z_2' \\ \vdots \\ z_n' \end{bmatrix}$$

The average moment conditions are

$$\begin{aligned} \bar{m}_n(\theta) &= \frac{1}{n}Z'\epsilon \\ &= \frac{1}{n}(Z'y - Z'X\theta) \end{aligned}$$

Exercise: Derive the below representation of IV estimator from previous class, or Creel.

$$\hat{\theta}_{IV} = \left(X'Z(Z'Z)^{-1}Z'X\right)^{-1}X'Z(Z'Z)^{-1}Z'y \tag{1}$$

As an exercise, you can verify GIV estimator is

1. Consistent and
2. Asymptotically normally distributed

Example (GIV)

Suppose the model is

$$\begin{aligned} y_t^* &= \alpha + \rho y_{t-1}^* + \beta x_t + \epsilon_t \\ y_t &= y_t^* + \upsilon_t \end{aligned}$$

where ϵ_t and υ_t are independent Gaussian white noise errors. Suppose that y_t^* is not observed, and instead we observe y_t. If we estimate the equation

$$y_t = \alpha + \rho y_{t-1} + \beta x_t + \nu_t$$

this the estimator is biased and inconsistent.

What about using the GIV estimator?

Consider using as instruments $Z = [1\, x_t\, x_{t-1}\, x_{t-2}]$. The lags of x_t are correlated with y_{t-1} as long as β is different from zero, and by assumption x_t and its lags are uncorrelated with ϵ_t and υ_t (and thus they're also uncorrelated with ν_t). Thus, these are legitimate instruments. As we have 4 instruments and 3 parameters, this is an overidentified situation.

In [75]:
```
function  lags(x::Array{Float64,2},p)
        n, k = size(x)
        lagged_x = zeros(eltype(x),n,p*k)
        for i = 1:p
                lagged_x[:,i*k-k+1:i*k] = lag(x,i)
        end
    return lagged_x
end
```

Out[75]:
```
lags (generic function with 2 methods)
```

In [76]:
```
function lag(x::Array{Float64,2},p::Int64)
        n,k = size(x)
        lagged_x = [ones(p,k); x[1:n-p,:]]
end
```

Out[76]:
```
lag (generic function with 2 methods)
```

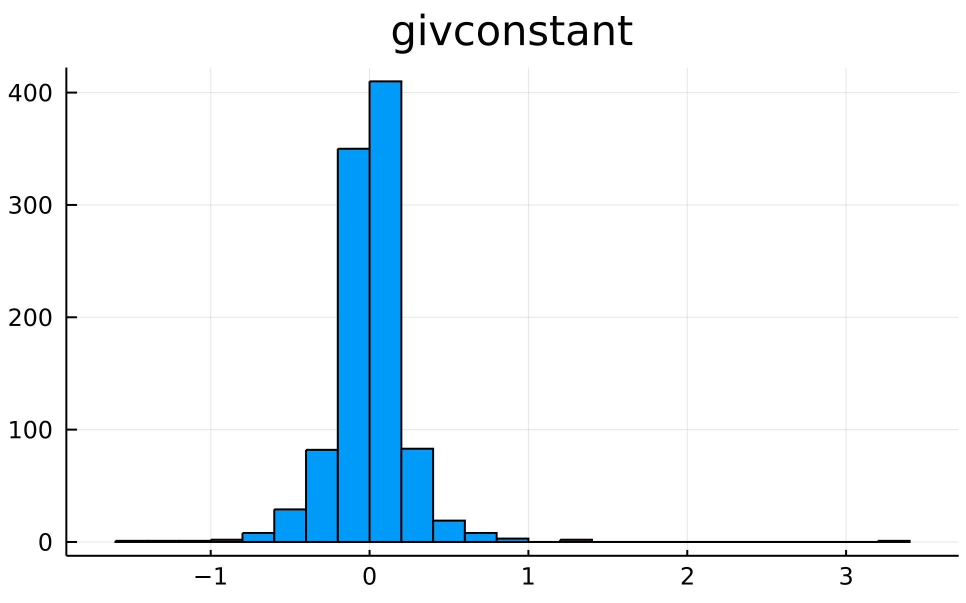
In [83]:
```
function GIVmoments(theta, data)
        data = [data lags(data,2)]
    data = data[3:end,:] # get rid of missings
        n = size(data,1)
        y = data[:,1]
        ylag = data[:,2]
        x = data[:,3]
        xlag = data[:,6]
        xlag2 = data[:,9]
        X = [ones(n,1) ylag x]
        e = y - X*theta
        Z = [ones(n,1) x xlag xlag2]
        m = e.*Z
end


# do the Monte Carlo
n = 100
sig = 1
reps = 1000
results = zeros(reps,3)
for rep = 1:reps
        x = randn(n) # an exogenous regressor
        e = randn(n) # the error term
        ystar = zeros(n)
        # generate the dep var
        for t = 2:n
          ystar[t] = 0.0 + 0.9*ystar[t-1] + 1.0*x[t] + e[t]
        end
    # add measurement error
        y = ystar + sig*randn(n)
        # now do GMM, using the data with meas. error in both dep. var. and regressor
        ylag = lag(y,1)
        data = [y ylag x];
    data = data[2:end,:] # drop first obs, missing due to lag
        theta = [0, 0.9, 1]
        weight = 1
    # note to self: this is very slow, because it uses the general GMM method, instead
    # also, the weight matrix is not optimal
    moments = theta-> GIVmoments(theta, data)
        b, junk, junk, junk, junk = gmm(moments, theta, weight)
        b = b - [0.0, 0.9, 1.0] # subtract true values, so mean should be approx. zero
    results[rep,:] = b
end
```

In [85]:

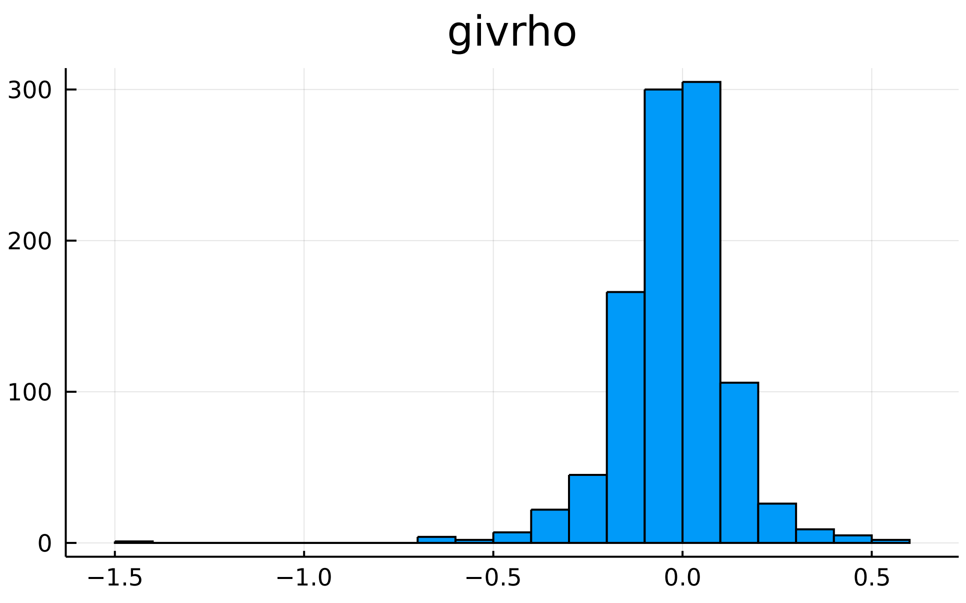
```
histogram(results[:,1],nbins = 30,title="givconstant",legend=false)
plot!(size=(500,300))
```

Out[85]:



In [86]:
```
histogram(results[:,2],nbins = 30,title="givrho",legend=false)
plot!(size=(500,300))
```

Out[86]:



In [87]:
```
results
```

Out[87]:
```
1000×3 Array{Float64,2}:
  0.0361209   0.070782    -0.0670949
 -0.420143   -0.173636    -0.198024
  0.0160008   0.0853514    0.0951837
  0.0374344  -0.259086     0.027451
 -0.0888301  -0.0703276    0.563424
  0.0430656   0.0172      -0.0653093
  0.0283554  -0.184548    -0.0444603
  0.0531046   0.0688273   -0.0186385
 -0.394061   -0.117586    -0.169036
  0.0111533  -0.166998     0.0560165
  0.0850043   0.0397519   -0.245819
 -0.0532908   0.024126    -0.127344
  0.532533   -0.162968     0.0819763
  ⋮
  0.133765    0.00384949  -0.0849784
 -0.0909113  -0.331881    -0.251216
 -0.0575435   0.120515     0.157166
 -0.0361674   0.110857     0.226054
 -0.218999    0.155639    -0.113875
  0.720517    0.405889     0.476985
 -0.111555    0.200226    -0.215096
  0.0978214   0.0101492    0.292513
 -0.0179431   0.088332     0.109964
 -0.598198   -0.57408     -0.367348
  0.173642   -0.234145    -0.128681
 -0.0248432  -0.0528914    0.234389
```

In [30]:
```
mean(results,dims=1)
```

Out[30]:
```
1×3 Array{Float64,2}:
 0.00155478  -0.020019  -0.00456686
```

In [31]:
```
std(results,dims=1)
```

Out[31]:
```
1×3 Array{Float64,2}:
 0.250703  0.149116  0.180644
```