Practical Empirical Likelihood Estimation with matElike

John Zedlewski (jzedlewski@hbs.edu)

June 1, 2008

1 Introduction

`matElike` is a Matlab package that simplifies the process of estimating moment condition models by Empirical Likelihood and related methods. It supports nonlinear moment conditions, the free specification of Cressie-Read parameters, and fast estimators for linear instrumental variables models. `matElike` uses a direct optimization approach combined with modern, efficient nonlinear solver techniques. This approach has proven to be robust and efficient for large models (with hundreds of thousands of observations) and models with difficult nonlinear moment conditions.

The `matElike` package requires only a standard version of Matlab, with no specialized toolboxes, although it can optionally take advantage of the external IPOPT solver library if it is available.

The reader in a hurry, who already has some knowledge of EL methods, should be able to use `matElike` after reading the example in section 2 then the sections on installation and basic usage (sections 4 and 4.3).

2 A simple example

Later sections will provide more detail on the models and methods used by `matElike`, but it may be helpful to start with an example. Consider a standard linear instrumental variables model, where we have N observations of x (a set of L possibly endogenous structural variables), z (a set of J exogenous instruments), and y (the dependent variable). Call the j-th instrument $z^{(j)}$. We have J moments with expectation 0, namely:

$$\mathbb{E}\left[(y - \beta' x)\, z^{(j)}\right] = 0 \qquad \text{for } j = 1 \ldots J \tag{1}$$

So we can define our J-th moment function to be:

$$\psi_i^{(j)}(\beta) = (y_i - \beta' x_i)\, z_i^{(j)} \tag{2}$$

Listing 1: Moment function for linear IV

```
function Mj = demoMoments(theta, elike, j, newTheta)
    global X y Z;
    resid = y - X * theta;
    Mj = Z(:,j) .* resid;    % Interact j-th instrument with residual
end
```

Listing 2: Solving the linear IV model

```
global X y Z;

% ... code to load the data matrices omitted ...

[N L] = size(X);
J = size(Z,2);

elike = elSetup(N, J, L, @demoMoments);
res = elSolve(elike, 'EL');
disp(res.theta)              % Show the estimated parameters
```

We would like to estimate β using empirical likelihood. With `matElike`, we start by writing a function that computes our functions $\psi_i^{(j)}(\beta)$ for a given β and j, returning a column vector like $(\psi_1^{(j)}(\beta), \ldots, \psi_N^{(j)}(\beta))'$. In this example, assume we have stacked the observations into the matrices X (dimension $N \times L$) and Z (dimension $N \times J$) and stored X, y, and Z in global variables. Then our moment function is simply Listing 1.

Once we've defined the moment function, we configure our model with `elSetup` and then pass this configuration to `elSolve`, which maximizes the empirical likelihood function and returns the estimated parameters. Listing 2 shows all the code needed to estimate the model.

And that's it! To run this example or view the complete code, look at `matelike/demos/runSimpleDemo.m`. In practice, `matElike` includes an optimized function `elLinearIV` that is faster for models of this type, but the process for estimating complex nonlinear models is identical to the example above.

3 Empirical likelihood theory and implementation

Empirical likelihood (EL) provides a method to estimate the parameters of a model defined by estimating equations like those in equation 1 above, without specifying the distribution of the disturbances. EL is applicable in many of the settings where the Generalized Method of Moments (GMM) has been applied,

but EL has attractive properties beyond those of two-step or iterated GMM. In particular, EL is invariant to the scaling of the data, and Newey and Smith [2004] have shown it to have higher-order bias properties that are superior to those of GMM. A full discussion of empirical likelihood theory is beyond the scope of this guide – in particular, regularity conditions may be omitted here, and we will only focus on models actually implemented in `matElike`. For a better introduction to the methods themselves, see Imbens [2002] or the book-length treatment by Owen [2001].

3.1 Basic EL model

Throughout this discussion[1], we assume that the data are a sample of N iid observations z_i. The empirical log likelihood is defined to be:

$$L(\pi) = \sum_{i=1}^{N} \ln \pi_i$$

where π is a vector of length N such that $0 < \pi_i < 1$ and $\sum_i \pi_i = 1$. Our goal is to estimate β, a K-dimensional vector whose true value is β_0. We know that $\mathbb{E}\psi(z, \beta) = 0$ for $\beta = \beta_0$, but $\mathbb{E}\psi(z, \beta') \neq 0$ for $\beta' \neq \beta_0$. Here, $\psi(z, \beta)$ is a vector-valued function of dimension J. We will denote the j-th element of ψ by $\psi^{(j)}$, as in the example above[2]. We estimate β by solving:

$$\hat{\beta} = \underset{\pi,\, \beta}{\operatorname{argmax}} \sum_{i=1}^{N} \ln \pi_i \tag{3}$$
$$\text{subject to } \sum_{i=1}^{N} \pi_i = 1\,, \quad \sum_{i=1}^{N} \pi_i\, \psi(z_i, \beta) = 0$$

Clearly, if we did not include the restriction that $\sum_{i=1}^{N} \pi_i\, \psi(z_i, \beta) = 0$, we would estimate that the unrestricted $\pi^{unres} = \iota/N$, where ι is a vector of all 1's. With this constraint included, our procedure is trying to find the $\hat{\pi}$ that is "closest" to $\frac{1}{N}$ in a particular sense, while still satisfying our moment conditions. By using different notions of "closeness" between the estimated $\hat{\pi}$ and ι/N, we can obtain different estimators. `matElike`, along with much of the EL literature, uses the Cressie-Read (CR) class of functions to express this notion of closeness. The CR family is parameterized by a scalar $\lambda \in [-2, 1]$. Given two discrete distributions with the same support over N points and probability weights $p = (p_1, \ldots, p_N)$ and $q = (q_1, \ldots, q_N)$, respectively, the CR discrepancy between them is:

$$I_\lambda(p, q) = \frac{1}{\lambda\,(1 + \lambda)} \sum_{i=1}^{N} p_i \left[(p_i/q_i)^\lambda - 1 \right]$$

[1] This section is loosely based on lecture notes from Guido Imbens for Harvard EC2144, Spring 2008

[2] Assume also that $\beta \in \beta$, where β is a compact subset of $\mathbb{R}^K$.

This leads to the natural definition of a more general class of estimators. We will define the estimator:

$$\hat{\beta}^{CR(\lambda)} = \underset{\pi,\beta}{\operatorname{argmax}} I_\lambda(\iota/N, \pi) \quad \text{subject to } \sum_{i=1}^{N} \pi_i = 1\,, \sum_{i=1}^{N} \pi_i\, \psi(z_i, \beta) = 0 \quad (4)$$

As a shorthand, we will refer to these parameterized estimators as $CR(\lambda)$. This class includes the most important Generalized Empirical Likelihood estimators that have been studied. $CR(0)$ is the empirical likelihood estimator described above[3]. $CR(-1)$ is the expontial tilting (ET) estimator studied by Imbens et al. [1998], and $CR(-2)$ is the continuously updating estimator (CUE) of Hansen et al. [1996]. Newey and Smith [2004] have analyzed the family of Generalized Empirical Likelihood (GEL) estimators, which are defined as the solution to a saddle point problem, in detail and shown that all of these $CR(\lambda)$ estimtors have equivalent GEL versions. As `matElike` works directly with the $CR(\lambda)$ formulation, we will not discuss other GEL estimators here[4].

Within `matElike` , changing from an EL estimator to a $CR(\lambda)$ estimator for an arbitrary λ simply requires changing a single function argument, so in the discussion below we will often describe EL estimation only. Unless otherwise specified, the methods apply equally well to other $CR(\lambda)$ estimators.

3.2 EL computation

The basic formulation of EL leads to a nonlinear programming problem with $N+K+M$ parameters, which can be a daunting problem for many optimization methods. For this reason, most approaches to computing EL estimators have focused on concentrating out some of the parameters. For instance, optimization can be performed instead over the $M + K$ space of Lagrange multipliers and structural parameters by substituting in the value of π_i implied by the first order conditions ($\pi_i = \exp(\mu - 1 + \lambda' \psi(z_i, \theta)$, where μ is the Lagrange multiplier on the constraint that $\sum_i \pi_i = 1$). Mittelhammer [2003] further reduce the dimensionality by concentrating out the Lagrange multipliers λ as well, although they need to introduce an additional inner layer optimization problem to perform the concentration. While these approaches lead to a smaller optimization problem, they can also make the optimization problem more difficult, leading to frustrating convergence problems.

The `matElike` package instead solves equation (3) directly[5]. The largest block of the Hessian of the Lagrangian is the $N \times N$ block corresponding to

[3] For EL and ET, we have to interpret this as a limit as $\lambda \to 0$ or $\lambda \to -1$.

[4] The terminology here can get confusing. Newey and Smith refer to the $CR(\lambda)$ estimators as MD (minimum discrepancy) estimators, but this seems too easy to confuse with minimum distance estimation. Note that authors following Newey and Smith often refer to a γ parmeterizing the GEL-dual estimator. This γ is *not* the same number as the λ for CR.

[5] Owen [2001] describes this "primal" approach to solving EL problems. At the time the book was written, leading optimization codes (such as `nlsol`, which he discusses) did not typically support sparse analytic Hessians and Jacobians, so they were much less effective for solving large primal EL problems.

$\frac{\partial^2}{\partial \pi \, \partial \pi} \mathcal{L}$. Off of its main diagonal, this derivative is zero, however, so the matrix is extremely sparse, with only N nonzero entries. Optimization codes that exploit a sparse Hessian can solve such problems efficiently without consuming excessive time or memory. Convergence is generally quite reliable with this method, although the scaling of the problem may still be important – if one moment condition has values or derivatives several orders of magnitude greater than another condition, it should be scaled down to ensure numerical stability.

Versions of Matab before releasee 7.6 (R2008b) do not include solvers that support sparse Hessians with nonlinear constraints. However, `matElike` includes a modern optimization code (`zipsolver`, developed by John Zedlewski) that does support these features. It also includes a link to the more robust optimization code IPOPT, developed by Wächter and Biegler. IPOPT, written in C++, can be difficult for the inexperienced user to install, but for large or difficult problems, it has proven to be more effective than `zipsolver`. See the Installation section for more details on configuring IPOPT. When using Matlab release 7.6 with the optimization toolbox version 4.0 or greater, `matElike` can also use the interior-point algorithm of Matlab's `fmincon` function. You can choose which solver to use for each problem by setting the `elike.solver` field.

Experience has shown that an exact Hessian and Jacobian are necessary to obtain fast and reliable convergence for EL problems. Because these derivatives can be difficult to calculate by hand, `matElike` includes code to compute them through automatic differentiation (AD), supported by the Intlab library (developed by Siegfried Rump, included along with the default `matElike` installation). The AD library replaces the standard Matlab vectors and matrices with special objects that track all mathematical operations applied to them and compute derivatives accordingly. In general, this AD process should be transparent to a user of `matElike`, but there are a few caveats. Iterative algorithms, where the number or sequence of iterations depend on θ also cannot be used with AD. For moment conditions that are differentiable, but not compatible with AD, users can code the derivatives directly. See the `elSetup` docmentation for more details.

4 Basic `matElike` installation and usage

Note that all `matElike` functions support Matlab's online help system, so you can type `help elSetup` at the command prompt to view documentation for the `elSetup` function, for instance.

4.1 Installation

To install `matElike`, unzip the `matelike12.zip` archive and place the resulting `melroot` directory wherever you like to store your Matlab code. Add the subdirectory `melroot/matelike` to your Matlab path with a command like:

```
addpath c:/matlab/melroot/matelike
```

where `c:/matlab` is replaced with the base directory in which matelike is stored. Edit the file `matelike/elLoad.m` so that the variable `BASEPATH` points to the directory where `matElike` is installed. Now the package is installed, but you must load it into Matlab's memory in each session before you can use it. Load the package with the command:

```
elLoad
```

you should see some output and then "Loaded matElike verson 0.1.4 package successfully". To avoid repeating these steps every time you run Matlab, you can add those commands to your 'startup.m' file (search the Matlab help docs for startup.m for information on how to do this).

4.2 Using IPOPT (optional)

If you want to use IPOPT instead of the default zipsolver, you must install the IPOPT library and its Matlab interface. See `http://www.coin-or.org/Ipopt/documentation/node9.html` for details on the basic installation and `https://projects.coin-or.org/Ipopt/wiki/MatlabInterface` for instructions on installing the Matlab interface. If you are not comfortable compiling C++ code and working with code libraries, this may be a frustrating process – you should ask a system administrator for help. To load `matElike` with Ipopt as the default solver, initialize the package with the commmand `elLoad(true)`. You may need to edit the variable IPOPTPATH in elLoad.m to point to the base directory of the IPOPT installation (the one that contains subdirectories called `matlab` and `lib`). Note that your operating system needs to know how to find the IPOPT libraries too. On Linux, this typically means that you have to set the `LD_LIBRARY_PATH` variable from the shell before starting Matlab, with something like:

```
export LD_LIBRARY_PATH=~/matlab/ipopt/lib
```

where ' /matlab/ipopt' is replaced with the path to your Ipopt directory. Assuming you're running this bash shell, you can add this line to the '.bashrc' file in your home directory to run the command every time you log in. In Windows, you should add this directory to the PATH environment variable.

For many users, the default `zipsolver` will be perfectly adequate. However, if you have convergence issues with your model, and you have checked that it is well-specified and well-scaled, Ipopt will often prove to be more effective.

4.3 Basic `matElike` usage

Trying out the basic example in section 2 of this document will be the best way to get up to speed with `matElike`. The directory `matelike/demos` contains that example (in `runSimpleDemo.m`), a dynamic panel data example from Imbens [2002] (in `dynamicPanel.m`) and a Poisson nonlinear instrumental variables model (in `demoPoissinst.m`). The Appendix to this document includes reference documentation for all signficant `matElike` functions.

In general, there are only a few steps to using `matElike` for an arbitrary model:

- Write a function (which we will call `momFunction`) that evaluates the moment functions for each observation. The function may have its own m-file, or it may be a nested function in the same file as the calling code. Nested functions make it simple to share data between the setup code and the moment function – see the Matlab documentation for details.
- Call `elLoad` to initialize `matElike` (only needs to be done once per Matlab session).
- Call `elike = elSetup(nObs, nMom, nTheta, @momFunction)`, specifying the number of observations, number of moment conditions, number of elements of θ, and the moment condition function to use. The moment function is passed as a Matlab "function handle" – simply the name of the function preceeded by the `@` symbol.
- Call `res = elSolve(elike, method)` to solve the model. You can display the results in more detail by then calling `elModelSumm(res)`.

For linear two-stage-least-squares models, the `elLinearIV` function is much more efficient than using `elSolve` directly. To use `elLinearIV` you can skip the `elSetup` step: just put your instruments (Z) and structural variables (X) in to matrices and pass them into `elLinearIV`.

5 Empirical and simulation results

Both asymptotic theory and Monte Carlo simulations indicate that empirical ilkelihood can deliver more accurate results than GMM in many situations. Below, we consider a dynamic panel data model based on simulated data and a real empirical example with a simple nonlinear model.

5.1 Dynamic panel model

Imbens (2002) discusses the following dynamic panel data model with fixed effects and no covariates. We observe N individuals for T time periods. An observation consists of only the dependent variable Y_{it}, which has both an individual-specific unobservable component (η_i) and a dependence on the previous value of Y_i. The exact model is:

$$Y_{it} = \eta_i + \theta\, Y_{i(t-1)} + \varepsilon_{it}$$

and our goal is to estimate the scalar θ. Lagged values of Y are not correlated with ε_{it}, but they affect Y_{it} through the dynamic part of the model, so even distance lags can be used as instruments. So we can generate many moment conditions of the form:

$$\psi_{1t}(Y,\theta) = \begin{pmatrix} Y_{i(t-2)} \cdot \left(Y_{it} - Y_{i(t-1)} - \theta\left(Y_{i(t-1)} - Y_{i(t-2)}\right)\right) \\ Y_{i(t-3)} \cdot \left(Y_{it} - Y_{i(t-1)} - \theta\left(Y_{i(t-1)} - Y_{i(t-2)}\right)\right) \\ \cdots \\ Y_{i1} \cdot \left(Y_{it} - Y_{i(t-1)} - \theta\left(Y_{i(t-1)} - Y_{i(t-2)}\right)\right) \end{pmatrix}$$

Using all possible ψ_{1t} conditions, we have $(T-1) \times (T-2)/2$ moments. If we assume that the data come from a long-run steady state of the model, we can add $(T-2)$ more moment conditions:

$$\psi_{2t}(Y,\theta) = \left(Y_{i(t-1} - Y_{i(t-2)}\right) \cdot \left(Y_{it} - \theta\, Y_{i(t-1)}\right)$$

The model is greatly overidentified, with $(T+1) \times (T-2)$ total instruments for a single parameter. For a small θ, the long lags contain little information and generate weak instruments. However, with a value of θ close to 1, the distant lags can significantly improve the precision of our estimate.

While this model can be formulated as a linear instrumental variables problem, given some careful stacking of lags and differences, the code in `dynamicPanel.m` uses the generic `matElike` formulation of the moment functions to keep the example code clear.

	$\theta = 0.50$		$\theta = 0.90$		$\theta = 0.95$	
	EL	GMM	EL	GMM	EL	GMM
Median bias	-0.0007	0.0006	-0.0002	-0.0668	-0.0001	-0.2562
Std. dev.	0.0093	0.0093	0.0126	0.0704	0.0154	0.1977
Root MSE	0.0093	0.0093	0.0126	0.1056	0.0154	0.3717
Average time	3.1 s	0.62 s	3.0 s	0.62 s	3.1 s	0.64 s

Table 1: Monte Carlo results for dynamic panel model: $N = 1500$, $T = 12$

Table 1 presents the results of 100 Monte Carlo simulations comparing the performance of Empirical Likelihood and 2-step GMM estimators for this model with simulated data. The dataset contains 1500 individuals with $T = 12$ time periods, and results for three different parameter values $\theta = \{0.5, 0.90, 0.95\}$ are presented. Note that the average time of the EL approach would decrease significantly if the problem were reformulated for use with `elLinearIV`[6]. For $\theta = 0.5$, GMM and EL have nearly-identical performance. However, as the autoregressive component of Y gets large, GMM breaks down and delivers unreliable results.

5.2 Poisson-IV model

Mullahy [1997] presents a nonlinear instrumental variables esimator for a Poisson regression model with an omitted variable that is correlated with the struc-

[6] In 3 of the 100 the simulations with $\theta = 0.95$, the EL estimator failed to converge. These simulations were dropped from both the GMM and EL results. The GMM estimator uses the identity matrix as its first step weighting matrix. Estimation times were recorded on a 2.4 ghz AMD Opteron processor.

tural variables (x) but not with the instruments (z). While the standard Poisson model uses the conditional mean specification $\mathbb{E}[y|x] = \exp(x\,\theta)$, Mullahy's model is:

$$\begin{aligned}\mathbb{E}[y \mid x, \eta] &= \exp(x\beta + \nu) \\ &= \exp(x\beta) \cdot \eta \\ \mathbb{E}[\eta \mid z] &= \tau\end{aligned}$$

where τ is normalized to 1. Given J instruments $\{z^{(1)}, z^{(2)}, \ldots, z^{(J)}\}$, we can consistently estimate β with EL or GMM, using the moment functions:

$$\psi^{(j)}(\beta) = z^{(j)} \cdot (\exp(-x\beta)\, y - 1)$$

This model is implemented in `demos/poissinst.m`. Mullahy estimates a model of cigarette demand, where the x variables include a measure of "habit stock," which is correlated with the unobserved η. As instruments, he includes lagged values of cigarette prices, lagged values of dummies for various statewide smoking restrictions, a cubic term in age and an interaction between age and education, leading to 15 instruments and 11 structural variables. (See `demos/demoPoissinst.m` for more details.) The dataset has 6160 total observations extracted from the 1979 National Health Interview Survey.

	EL		GMM		Rel. diff
Habit stock	0.0033	(0.0021)	0.0040	(0.0024)	0.3196
Price 79	-0.0086	(0.0044)	-0.0084	(0.0044)	0.0364
Rest res. 79	-0.0632	(0.0520)	-0.0590	(0.0512)	0.0806
Income	-0.0062	(0.0028)	-0.0070	(0.0028)	0.2543
Age	0.0896	(0.0392)	0.0776	(0.0439)	0.3069
Age^2	-0.0012	(0.0004)	-0.0011	(0.0004)	0.3090
Educ	0.1412	(0.0321)	0.1376	(0.0313)	0.1110
$Educ^2$	-0.0093	(0.0014)	-0.0092	(0.0013)	0.0949
Fam size	-0.0132	(0.0125)	-0.0127	(0.0125)	0.0365
White	-0.1026	(0.0687)	-0.0903	(0.0670)	0.1787
Intercept	-2.5436	(0.5663)	-2.4040	(0.6051)	0.2464

Table 2: Cigarette demand estimation results (standard error in parentheses)

Table 2 presents the results of the model as estimated by EL and 2-step GMM[7]. The relative difference columm is the absolute difference between the two estimators, divided by the EL standard errors. It appears that 2-step GMM has overestimated the impact of the endogenous habit stock variable by about 30% of a standard deviation in this procedure. Note that these GMM results are slightly different from those presented in Mullahy (1997) – they appear to be sensitive to the choice of the initial weighting matrix, which is a common issue in GMM estimation.

[7]The GMM estimator uses $(Z'Z/N)^{-1}$ as its initial weighting matrix.

6 Appendix: `matElike` function reference

This documentation is also available via Matlab's online help system.

---

elLoad

Loads ipopt, matelike, and intlab libraries

Usage:

```
elLoad
```

or:

```
elLoad(solver, forceReload)
```

Inputs:

`solver` is an optional string specifying the nonlinear solver to use.

Valid solver options are:

'zipsolver' -- matElike's default, built-in solver
'ipopt' -- the Ipopt solver, which require the Ipopt MEX interface and library to be installed
'fmincon' -- the fmincon solver from Matlab's optimization toolbox. This is only available with optimization toolbox version 4.0 (Matlab R2008a) or greater, as the older versions lack support for analytic Hessians with inequality constraints.

In general, Ipopt is the most robust solver, but 'zipsolver' works well for most problems as long as they're reasonably well scaled. 'fmincon' appears to be less reliable, but slightly faster than zipsolver.

IMPORTANT: You will need to edit BASEPATH below to point to the directory containing the 'matelike' and 'zipsolver' folders on your system.

---

elSetup

Initializes an EL (or other Cressie-Read) problem

Usage:

```
elike = elSetup(nObs, nMom, nTheta, momFunction, ...)
```

Inputs:

`nObs, nMom, nTheta` are the number of observations, number of moment conditions, and number of elements in theta

Extra arguments may follow 'momFunction' to set various options for the optimization process. Options are specified as pairs, where the first argument is the option name and the second is the option value, like:

```
elSetup(nobs,nmom,ntheta,func, 'verbose', false)
```

which sets the 'verbose' option to false, greatly reducing the output displayed.

`momFunction` is a function handle to compute the moment conditions for each observation, with the form:

```
M = momFunction(theta, elike, mnum, newTheta)
```

`mnum` indicates which moment to evaluate, `theta` is the point at which to evaluate it, and `elike` is the same structure that was returned from elSetup. If you need to pass additional paraneters to your function, you can stash them in `elike` as fields.

`newTheta` is true if this theta differs from the one that was passed in the most recent call. If you need to precompute some values only once for each theta, `newTheta` tells you when it's time to redo that computation.

The returned value M should be an nObs x 1 vector containing moment condition `mnum` evaluated at each observation.

*** IMPORTANT restrictions on momFunction ***

elSolve will use automatic differentiation to compute the derivatives of momFunction. For smooth functions computed analytically, this should work transparently. It will NOT work if momFunction uses some iterative algorithm internally, is non−differentiable, or uses some obscure external functions that are not understood by the differentiation library (Intlab).

The matrix returned by momFunction must be of the special AD type. This will happen by automatically if the result is produced by operations involving `theta`, but it will fail if you manually initialize M with something like `M = zeros(nObs,1)`. If you find this initialization convenient, you can use:
`M = zeros(nObs,1) * theta(1)`,
which will ensure that M has the correct type.

If automatic differentiation is not suitable for your model, you can provide functions in `elike.userCompJac` and `elike.userCompHessTheta` to compute them. (See elLinearIV for an example.)

Output:

`elike` is a structure that can be passed to elSolve

Requirements:

You must have Intlab installed to use elSetup. You also need to call `elLoad` sometime before elSetup to initialize the matelike library.

Author: John Zedlewski (jzedlewski@hbs.edu)

---

elSolve

Solves the empirical likelihood (or other CR−family) model in elike

Primary Usage:

```
res = elSolve(elike, method, thetaGuess, pguess, ...)
```

Inputs:

`elike` is a structure containing the problem definition. It must be created by: elike = elSetup(...); See `help elSetup`

`method` specifies which objective function to use. It can be 'EL' for empirical likelihood, 'ET' for exponential tilting, or a numerical value indicating the Cressie−Read lambda parmeter to use. If 'method' is 'GMM', a simple two−stage GMM estimator will be used (ignoring pguess and other EL−specific params).

`thetaGuess` is the starting value for the theta parameters.

`pguess` is an optional starting value for the pi parameters. If this argument is empty ([]) or omitted, pi=1/N will be used.

Outputs:

`res` is a structure containing:

```
res.theta   -- estimated value for theta
res.lagmult -- lagrange multipliers on moment constraints
res.p       -- estimated pi for each observation
res.numiter -- number of iterations taken
res.fval    -- value of objective function at optimum
res.flag    -- set to 0 if optimization succceeded, nonzero otherwise
```

Details:

elSolve will use the 'zipsolver' matlab package by default. If you have

Ipopt and its matlab interface installed, set `elike.solver = 'ipopt'` to use it (Ipopt is more robust and, for large problems, faster than zipsolver).

See 'help elSetup' for more info on configuring EL problems.

Author: John Zedlewski (jzedlewski@hbs.edu)

---

elLinearIV

Solves a linear instrumental variables problem with EL

Usage:

```
res = elLinearIV(X, Z, y, elMethod)
```

Inputs:

`X` is a matrix containing the structural variables
`Z` is a matrix containing the instruments
`y` is a vector of the dependent variable
`elMethod` specifies the method to use ('EL','ET', or a Cressie−Read lambda value). You can omit this argument to use the default 'EL'

The returned structure `res` is the same as that returned by elSolve. The field `res.theta` contains the estimates for the coefficients on X. Note that it contains a field `elike` with the problem setup. This may be useful to pass to `elEval(...)`, etc.

---

elModelSumm

Print some basic information about the EL model solution `res`

Usage:

```
[stderr,oidstat] = elModelSumm(res, names, [quiet])
```

Inputs:

`res` is a solution object returned by elSolve (...)
`names` is an optional vector of names for the elements of theta
`quiet` is an optional parameter −− if it's true, printing is suppressed

Outputs:

`stderr` is a column vector of computed standard errors.
`oidstat` is the chi−squared statistic from an LR test of overidentifying restrictions.

Details:

Reports the estimated theta from the model, standard errors, and a likelihood ratio−based test for the validity of the overidentifying restrictions .

The standard errors are based on the normal asymptotic approximation to the estimator of theta, using a variance matrix based on the moments weighted by the computed `pi` values.

---

elEval

Evaluates the empirical likelihood function at the given point

Usage:

res = elEval(elike, theta, p, lambda);

`elike` should be a structure obtained from elSetup (...)

`method` is 'EL','ET', or a CR constant parameter

`lambda` contains the Lagrange multipliers, but may be omitted if you don't want the Hessian.

`res` contains the following fields :

- `res.f` −− objective function
- `res.fgrad` −− gradient of objective
- `res.H` −− Hessian of the Lagrangian (empty if lambda missing)
- `res.c` −− constraints (sum(pi(i) ∗ m(i))
- `res.cJ` −− Jacobian of moments per observation
- `res.mom` −− Moment conditions for each observation: an (nObs x nMom) matrix with one column for each moment condition and one row for each observation.

---

elLRTest

Computes an empirical likelihood ratio test of the hypotheses `R*theta = b`

Usage:

```
testres = elLRTest(elres)

testres = elLRTest(elres, R, b, joint)
```

Inputs:

`elres` is a result structure returned by elSolve

`R` is a K x nTheta matrix of linear hypotheses to test, while `b` is a K x 1 vector of values to test R against. If `joint` is

omitted or false , then each row of the matrix is a separate linear hypothesis to be tested of the form:

H0: R(k,:) $*$ theta = b

If `joint` is true , then the vector hypothesis: H0: hyp $*$ theta = b is tested .

If `R` is omitted, by default elLRTest tests separately that each element of theta = 0. (So it is equivalent to `hyp = eye(nTheta)`)

Outputs:

`pval` is K x 1 vector for separate estimation or a scalar for joint estimation. It is the p−value for the LR test: the probability under the null that the LR statistic would be at least as large as the one obtained. It comes from the CDF of a chi−squared(dof) distribution, where `dof` is the rank of the hypothesis. If `joint` is false , separate p−values are computed for each hypothesis.

`lrstat` the LR statistic obtained from each hypothesis:

LR = $-2 *$ (L(theta_restricted) − L(theta_unres))

Notes:

Each test requires re−solving the model, so it may take some time for a large model.

---

elConfRegion

Computes an EL likelihood−ratio confidence interval and possibly plots it . Currently only supports univariate confidence intervals .

Primary Usage:

```
elConfRegion(elres, [whichVars], [confVals], [doPlot], [plotOpt])
```

Arguments in [ ] are optional .

Alternative Usage:

```
elConfRegion(elres, whichVars, confVals, 'precise')
```

Slightly more accurate, but much slower, optimization−based means of finding confidence regions .

Inputs:

`elres` is a result structure previously returned by elSolve (...)

`whichVars` specifies the indices of theta for which confidence regions are desired. Leave as '[]' or omit to compute CRs for all indices.

`confVals` specifies one or more confidence levels (e.g. [0.95,0.99]) to consider. Defaults to 0.95

`doPlot` specifies that the profile p−value should be plotted

`plotOpt` is an optional structure with fields specifying plotting options. Options include `plotOpt.numPoints` (number of points to sample), `plotOpt.varnames` (a cell array of variable names), `plotOpt.xciLabel` (whether to label the confidence interval on the x−axis), and `plotOpt.horizCR` (whether to draw a horizontal line for the confidence interval).

---

elValues

Computes the profile empirical likelihood , p−value, or LR test statistic at various points. Useful for plotting profile likelihoods .

The LR stat corresponds to the null hypothesis that:

```
theta(fixvars) = thetaList(fixvars,i)
```

The test has `length(fixvars)` degrees of freedom unless otherwise specified .

Usage:

```
[ll,lr,pv] = elValues(elres, thetaList, fixvars, [dof])
```

Inputs:

`elres` is a result structure previously returned by elSolve (...) solving the unconstrained problem.

`thetaList` is a matrix of size nFixed x nProfiles , where nFixed is the length of `fixvars` and nProfiles is the number of points at which to compute the profile likelihood . Each column is a single value of the parameter vector at which to compute the likelihood .

`fixvars` is an integer vector of length nFixed indicating which elements of theta should be held fixed . If fixvars is empty, it indicates that all elements of theta are fixed (only the p's are free). These all−theta−fixed problems can be solved faster.

`dof` specifies the number of degrees of freedom to use when computing the p−values. (Only needed if p−values are computed)

Outputs:

`ll` is a row vector containing the profile empirical log−likelihood at each point in thetaList. (This is the log likelihood obtained after maximizing over `p` and the theta elements not included in `fixvars`, but holding the other theta elements fixed.)

`lr` contains the profile empirical likelihood ratios (comparing the restricted and unrestricted models)

`pv` contains the profile empirical p−value comparing the restricted and unrestricted models(based on the chi−squared approximation to the distribution of the LR stat)

7 Appendix: Changes

- 0.1.2 – first public release
- 0.1.3 – added elConfRegion, fixed GMM standard errors, various other improvements
- 0.1.4 – support for fmincon, documentation improvements

8 References

References

L Hansen, J Heaton, and A Yaron. Finite-sample properties of some alternative gmm estimators. *Journal of Business & Economic Statistics*, Jan 1996. URL `http://links.jstor.org/sici?sici=0735-0015(199607)14%253A3%253C262%253AFPOSAG%253E2.0.CO%253B2-P`.

G Imbens. Generalized method of moments and empirical likelihood. *Journal of Business & Economic Statistics*, Jan 2002. URL `http://www.ingentaconnect.com/content/asa/jbes/2002/00000020/00000004/art00008`.

G Imbens, R Spady, and P Johnson. Information theoretic approaches to inference in moment condition models. *Econometrica*, Jan 1998. URL `http://links.jstor.org/sici?sici=0012-9682(199803)66%253A2%253C333%253AITATII%253E2.0.CO%253B2-M`.

R Mittelhammer. Empirical evidence concerning the finite sample performance of el-type structural equation .... *emlab.berkeley.edu*, Jan 2003. URL `http://emlab.berkeley.edu/symposia/nsf01/judge_p.pdf`.

J Mullahy. Instrumental-variable estimation of count data models: Applications to models of cigarette smoking .... *The Review of Economics and*

Statistics, Jan 1997. URL http://www.mitpressjournals.org/doi/pdf/10.1162/003465397557169.

W Newey and R Smith. Higher order properties of gmm and generalized empirical likelihood estimators. *Econometrica*, Jan 2004. URL http://www.blackwell-synergy.com/doi/abs/10.1111/j.1468-0262.2004.00482.x.

Art B Owen. Empirical likelihood. *Chapman & Hall/CRC*, 2001.