Maximum Likelihood Estimation of Utility Functions Using *Stata*

by

Glenn W. Harrison[†]

May 2008

Working Paper 06-12, Department of Economics,
College of Business Administration, University of Central Florida, 2006

Abstract. Economists in a wide range of fields are now developing customized likelihood functions to correspond to specific models of decision-making processes. The demand for customization derives partly from the need to consider a variety of parametric functional forms to account for experimental behavior under uncertainty. It is also generated by the fact that these models often specify decision rules that have to be "written out by hand." Thus it is becoming common to see user-written maximum likelihood estimates in behavioral econometrics, and less use of pre-packaged model specifications. These pedagogic notes document the manner in which one can estimate maximum likelihood models of utility functions within *Stata*. We can quickly go beyond "utility functions," in the narrow sense, and consider a wide range of decision-making processes.

[†] Department of Economics, College of Business Administration, University of Central Florida, USA. E-mail: GHARRISON@RESEARCH.BUS.UCF.EDU. Thanks to the U.S. National Science Foundation for research support under grants NSF/HSD 0527675 and NSF/SES 0616746, and to the Danish Social Science Research Council for research support under project #24-02-0124. I am grateful to Morten Lau, Lisa Rutström and Nat Wilcox for helpful comments, and particularly to Steffen Andersen and John Hey for comments and specific contributions to the exposition. Software and data to replicate all results can be obtained from *ExLab* at http://exlab.bus.ucf.edu.

Economists in a wide range of fields are now developing customized likelihood functions to correspond to specific models of decision-making processes. The demand for customization derives partly from the need to consider a variety of parametric functional forms to account for experimental behavior under uncertainty. It is also generated by the fact that these models often specify decision rules that have to be "written out by hand." Thus it is becoming common to see user-written maximum likelihood estimates in behavioral econometrics, and less use of pre-packaged model specifications.

These pedagogic notes document the manner in which one can estimate maximum likelihood models of utility functions within *Stata*. It provides a programming complement to the exposition of economic and modeling issues in Andersen, Harrison, Lau and Rutström [2008], Harrison, Lau and Rutström [2007] and Harrison and Rutström [2005][2008], as well as replications of specific classes of maximum likelihood models considered in important early studies by Camerer and Ho [1994; §6.1], Hey and Orme [1994] and Holt and Laury [2002]. We can quickly go beyond "utility functions," in the narrow sense, and consider a wide range of decision-making processes. We start with a standard Constant Relative Risk Aversion (CRRA) utility function and binary choice data over two lotteries, assuming expected utility theory (EUT). This step illustrates the basic economic and statistical logic, and introduces the core *Stata* syntax. We then consider an extension to consider loss aversion and probability weighting from Prospect Theory (PT), the inclusion of structural "stochastic errors," the estimation of utility numbers themselves to avoid any parametric assumption about the utility function, tests for non-nested model specifications, a simple mixture model in which some choices are EUT-consistent and some are PT-consistent, and the method of maximum simulated likelihood for estimating random coefficients. Once the basic syntax is defined from the first example, you will be delighted to see how quickly you can jump to other likelihood functions using different data and specifications. Of course, this is just a reflection of the "extensible power" of a package such as *Stata*, once one understands the basic syntax.

The exposition is deliberately transparent to economists. Most of the exposition in section A would be redundant for those familiar with Gould, Pitblado and Sribney [2006] or even Rabe-Hesketh and Everitt [2004; ch.13]. It is easy to find expositions of maximum likelihood in *Stata* that are more general and elegant for their purpose, but for those trying to learn the basic tools for the first time that elegance can just appear to be needlessly cryptic coding, and actually act as an impediment to comprehension. There are good reasons that one wants to build more flexible and computationally efficient models, but ease of comprehension is rarely one of them. In addition, many of the programming steps are specific to the issues that derive from behavioral models of choice under uncertainty.

Several methodological lessons emerge, quite apart from the econometric toolkit *per se*. First, it is feasible to undertake structural estimation of all of the parameters of a given model. This might seem self-evident to many, but the behavioral literature has developed a tendency to use "piecemeal estimation" methods in which some tasks are used to estimate or calibrate one parameter from a model, and then other tasks are used to estimate or calibrate another parameter. Joint estimation is essential for inference about the model as a whole, and the tools presented here should facilitate the wider use of joint estimation.

Second, dogmatic priors dominate the debates over models and data. Priors are invaluable for inference, and will likely play a greater role in behavioral econometrics, but they should not be used to dismiss results out of hand. We will see several instances in which the *a priori* implausibility of certain parameter estimates, from the perspective of the received literature, simply means that one has to think more about the economics and econometrics. Of course, such specification searches can become slippery slopes, in which priors are integrated with data in an *ad hoc* manner (Leamer [1978]). But we will see that a healthy interplay between theory and estimation will provide insights into behavior, so that specification searches of this kind can be kept to a minimum and properly documented.

Third, behavior should not be viewed as generated by any one theoretical model, since that can lead to invalid inferences.[1]

[1] It does not always lead to invalid inferences. If you need an estimate of an elasticity of substitution for a simulation model that has the maintained assumption that utility is Constant Elasticity of Substitution (CES), the econometrician's job is to estimate that specification and deliver the estimate (including standard errors, for sensitivity analysis in the simulation model). One might question why CES is a maintained assumption, and politely offer some generalizations, but that is a separate inferential task. To return to the present focus on EUT and choice under uncertainty, one can almost always "fit" an EUT model to observed data if some minimal stochastic assumptions are allowed. The issue is whether the fit is good or bad, in a sense that can only be correctly defined once the inferential objective is explicit.

Experimental economists are remarkably quick to close off alternative specifications, and intolerant to the possibility that one model might work well for some task domain and some subjects, while another model might work well for other task domains and other subjects. So stated, the notion that behavior is subject-specific and/or domain-specific receives widespread, blurred agreement, but when it comes to translating that agreement into formal inference the heavy hand of received econometric tools cuts off exploration. The tools presented here allow one to write out all of the models in a statistically consistent manner and estimate them using maximum likelihood (sections A though E), and then offers a constructive bridge to allow the consistent comparison of models (sections F and G).

A. Estimating a CRRA Utility Function

We assume that utility of income is defined by

$$U(x) = x^r \tag{1}$$

where x is the lottery prize and r is a parameter to be estimated. Under EUT the probabilities for each outcome k, p_k, are those that are induced by the experimenter, so expected utility is simply the probability weighted utility of each outcome in each lottery i:

$$EU_i = \sum_{k=1,n} [\, p_k \times u_k \,] \tag{2}$$

for n outcomes. In our examples n=4 or n=2, as explained below, but nothing essential hinges on that. The EU for each lottery pair is calculated for a candidate estimate of r, and the index

$$\nabla EU = EU_R - EU_L \tag{3}$$

calculated, where EU_L is the "left" lottery and EU_R is the "right" lottery in some display given to subjects. This latent index, based on latent preferences, is then linked to the observed choices using a standard cumulative normal distribution function $\Phi(\nabla EU)$. This "probit" function takes any argument between $\pm\infty$ and transforms it into a number between 0 and 1 using the function shown in Figure 1. The logistic function is very similar, as shown in Figure 1, and leads instead to the "logit" specification.

The latent index (3) could have been written in a ratio form:

$$\nabla EU = EU_R / (EU_R + EU_L) \tag{3'}$$

and then the latent index would already be in the form of a probability between 0 and 1, so we would not need to take the probit or logit transformation. We will see that this specification has also been used, with some modifications we discuss later, by Holt and Laury [2002].

The code listed below is written for version 9 of *Stata*, documented in StataCorp [2005]. With one or two minor exceptions it will run on earlier versions of *Stata*. It is generally written with many comment statements, and in a deliberately verbose manner in which all data and interim calculations are spelled out to ease comprehension. After the basics are understood, it is easy to write more compact code if one really wants to.

The following program defines the model:

```
program define ML_eut0

    * specify the arguments of this program
    args lnf r

    * declare the temporary variables to be used
    tempvar choice prob0l prob1l prob2l prob3l prob0r prob1r prob2r prob3r endow m0 m1 m2 m3 y0 y1 y2 y3 euL euR euDiff

    * please do not display all these steps on the screen, for every ML iteration!
    quietly {

        * initialize the data
        generate int `choice' = $ML_y1

        generate double `prob0l' = $ML_y2
        generate double `prob1l' = $ML_y3
        generate double `prob2l' = $ML_y4
        generate double `prob3l' = $ML_y5

        generate double `prob0r' = $ML_y6
        generate double `prob1r' = $ML_y7
        generate double `prob2r' = $ML_y8
        generate double `prob3r' = $ML_y9

        generate double `m0' = $ML_y10
        generate double `m1' = $ML_y11
        generate double `m2' = $ML_y12
        generate double `m3' = $ML_y13
        generate double `endow' = $ML_y14

        * construct the argument of the utility function
        replace `m0' = `endow'+`m0'
        replace `m1' = `endow'+`m1'
        replace `m2' = `endow'+`m2'
        replace `m3' = `endow'+`m3'

        * evaluate the utility function
        generate double `y0' = `m0'^`r'
        generate double `y1' = `m1'^`r'
        generate double `y2' = `m2'^`r'
        generate double `y3' = `m3'^`r'

        * calculate EU of each lottery
        generate double `euL' = (`prob0l'*`y0')+(`prob1l'*`y1')+(`prob2l'*`y2')+(`prob3l'*`y3')
        generate double `euR' = (`prob0r'*`y0')+(`prob1r'*`y1')+(`prob2r'*`y2')+(`prob3r'*`y3')

        * get the Fechner index
        generate double `euDiff' = `euR' - `euL'

        * evaluate the likelihood
        replace `lnf' = ln(normal( `euDiff')) if `choice'==1
        replace `lnf' = ln(normal(-`euDiff')) if `choice'==0

        * save the calculated likelihood in an external storage variable (for post-processing, explained later)
        replace ll = `lnf'

    }
end
```

This program is written to evaluate the lottery choices of Harrison and Rutström [2005], which is a replication of the experimental tasks of Hey and Orme [1994] but with some prizes being negative, and with information on individual demographic characteristics being collected.[2] It is a simple matter to modify it for other tasks, in which there might be more or less than 4 prizes per lottery.

This program makes more sense when one sees the command line invoking it, and supplying it with values for all variables. The

[2] We restrict the data that is passed to only include expressions of strict preference. The response of indifference was allowed in these experiments. One could easily modify the likelihood function to handle indifference, as explained in Andersen, Harrison, Lau and Rutström [2008] and Harrison and Rutström [2008].

simplest case is where there are no explanatory variables for the CRRA coefficient, which we come to shortly:

```
ml model lf ML_eut0 (r: Choices P0left P1left P2left P3left P0right P1right P2right P3right prize0 prize1 prize2 prize3
       stake = ), cluster(id) technique(nr) maximize
```

The "ml model" part invokes the *Stata* maximum likelihood model specification routine, which essentially reads in the ML_eut0 program defined above and makes sure that it does not violate any syntax rules. The "lf " part of "lf ML_eut0" tells this routine that this is a particular type of likelihood specification: specifically, that the routine ML_eut0 does not calculate analytical derivatives, so those must be calculated numerically. The part in brackets defines the equation for the CRRA coefficient r. The "r:" part just labels this equation, for output display purposes and to help reference initial values if they are specified for recalcitrant models. There is no need for the "r:" here to match the "r" inside the ML_eut0 program; instead, we could have referred to "rEUT:" in the "ml model" command. We use the same "r" to help see the connection, but it is not essential.

The "`Choices P0left P1left P2left P3left P0right P1right P2right P3right prize0 prize1 prize2 prize3 stake`" part tells the program what observed values and data to use. This allows one to pass parameter values as well as data to the likelihood evaluator defined in ML_eut0. Each item in this list translates into a $ML_y* variable referenced in the ML_eut0 program, where * denotes the order in which it appears in this list. Thus the data in variable Choices, which consists of 0's and 1's for choices, is passed to the ML_eut0 program as variable $ML_y1. Variable p0left, which holds the probabilities of the first prize of the lottery presented to subjects on the left of their screen, is passed as $ML_y2, and so on. Finally, variable stake, holding the values of the initial endowments provided to subjects, gets passed as variable $ML_y14. It is good programming practice to then define these in some less cryptic manner, as we do just after the "quietly" line in ML_eut0. This does not significantly slow down execution, and helps avoid cryptic code. There is no error if some variable that is passed to ML_eut0 is not referenced in ML_eut0.

Once the data is passed to ML_eut0 the likelihood function can be evaluated. By default, it assumes a constant term, so when we have "=)" in the above command line, this is saying that there are no other explanatory variables. We add some below, but for now this model is just assuming that one CRRA coefficient characterizes all choices by all subjects. That is, it assumes that everyone has the same risk preference.

Returning to the ML_eut0 program, the "args" line defines some arguments for this program. When it is called, by the Newton-Raphson optimization routine within *Stata*, it accepts arguments in the "r" array and returns a value for the log-likelihood in the "lnf" scalar. In this case "r" is the vector of coefficient values being evaluated. The optimization algorithm can be set using the "technique" option, and *Stata* defaults to Newton-Raphson. The major alternatives are Davidson-Fletcher-Powell or Broyden-Fletcher-Goldfarb-Shanno, selected using acronyms dfp or bfgs instead of acronym nr.[3]

The "tempvar" lines create temporary variables for use in the program. These are temporary in the sense that they are only local to this program, and hence can be the same as variables in the main calling program. Once defined they are referred to within the ML_eut0 program by adding the funny left single-quote mark ` and the regular right single-quote mark '. Thus temporary variable euL, to hold the expected utility of the left lottery, is referred to as `euL' in the program. Note that this is `euL' and not 'euL': beginning *Stata* users make this mistake a lot.

The "quietly" line defines a block of code that is to be processed without the display of messages. This avoids needless display of warning messages, such as when some evaluation returns a missing value. Errors are not skipped, just display messages. Since the ML_eut0

[3] A fourth algorithm is Berndt-Hall-Hall-Hausman (bhhh), but is not recommended for the class of problems considered here.

program is called many, many times to evaluate Jacobians and the like, these warning messages can clutter the screen display needlessly. During debugging, however, one normally likes to have things displayed, so the command "quietly" would be changed to "noisily" for debugging. Actually, we use the "ml check" option for initial debugging, as explained later, and never change this to "noisily." We do sometimes add a command within this block to "noisily" display some interim output. Thus, if you were concerned that the EU values were not evaluating properly, you might add a debugging statement such as the last line in this fragment:

```
 * get the Fechner index
 generate double `euDiff' = `euR' - `euL'
noisily summ `euR' `euL' `euDiff'
```

This will cause a steady stream of descriptive statistics to be sent to the display and your output log file, which can then be evaluated to see what the problem is, or lead you to also examine the probabilities or the prizes, to see if they had been read in correctly. When the debugging of this part is done change the "noisily summ" to "quietly summ" or delete this debugging line.[4]

The remaining lines should make sense to any economist from the comment statements. The program simply builds up the expected utility of each lottery, using the CRRA specification for the utility of the prizes. Then it uses the probit index function to define the likelihood values. The actual responses, stored in variable Choices (which is internal variable $ML_y1, that is read into local variable `choice'), are used at the very end to define which side of the probit index function this choice happens to be. The logit index specification is just as easy to code up: you replace "normal" with "invlogit".

The "cluster(id)" command at the end tells *Stata* to treat the residuals from the same person as potentially correlated. It then corrects for this fact when calculating standard errors of estimates.[5] We later consider maximum simulated likelihood methods for allowing

[4] Later we introduce the concept of a "global" in *Stata*, which allows you to write a program with debugging statements that are turned on or off with minimal effort.

[5] More generally, *Stata* allows the specification of complex, stratified sampling designs, which will improve estimates when used. These are common in surveys of a target population, and have been used in the estimation of preference parameters from behavior observed in field experiments (e.g., Harrison, Lau and

for unobserved heterogeneity.

Invoking the above command line, with the ", maximize" option at the end to tell *Stata* to actually proceed with the optimization, generates this output:

```
initial:       log pseudolikelihood =  -10201.74
alternative:   log pseudolikelihood = -10058.855
rescale:       log pseudolikelihood = -10058.855
Iteration 0:   log pseudolikelihood = -10058.855  (not concave)
Iteration 1:   log pseudolikelihood = -9617.4367
Iteration 2:   log pseudolikelihood = -9512.9457
Iteration 3:   log pseudolikelihood = -9512.8186
Iteration 4:   log pseudolikelihood = -9512.8186

. ml display

                                                  Number of obs   =      14718
                                                  Wald chi2(0)    =          .
Log pseudolikelihood = -9512.8186                 Prob > chi2     =          .

                                   (Std. Err. adjusted for 240 clusters in id)
------------------------------------------------------------------------------
             |               Robust
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       _cons |    .791697    .016104    49.16   0.000     .7601336    .8232603
------------------------------------------------------------------------------
```

So we see that the optimization routine converged nicely, with no error messages or warnings about numerical irregularities at the end. The interim warning message is nothing to worry about: only worry if there is an error message of any kind at the end of the iterations. Of course, lots of error message, particularly about derivatives being hard to calculate, usually flag convergence problems. The "ml display" command allows us to view the standard output, and is given after the "ml model" command.[6] For our purposes the critical thing is the

Williams [2002], Harrison, Lau, Rutström and Sullivan [2005], and Harrison, Lau and Rutström [2007]). Gould, Pitblado and Sribney [2006; ch.11] explain the methods for these extensions.

[6] The missing values for the Wald test just reflect the fact that we have no covariates in the model, so that is no cause for concern here.

"_cons" line, which displays the maximum-likelihood estimate and it's standard error. Thus we have estimated that $\hat{r} = 0.792$. This is the maximum likelihood CRRA coefficient in this case. This indicates that these subjects are risk averse.

Before your program runs nicely it may have some syntax errors. The easiest way to check these is to issue the command

```
ml model lf ML_eut0 (r: Choices P0left P1left P2left P3left P0right P1right P2right P3right prize0 prize1 prize2 prize3
        stake = )
```

which is the same as before except that it drops off the material after the comma, which tells *Stata* to maximize the likelihood and how to handle the errors. This command simply tells *Stata* to read in the model and be ready to process it, but not to begin processing it. You would then issue the command

```
        ml check
```

and *Stata* will provide some diagnostics. These are really quite informative if you use them, particularly for syntax errors.

The real power of this approach becomes evident when we allow the CRRA coefficient to be determined by individual or treatment characteristics. To illustrate, consider the effect of allowing the CRRA coefficient to differ depending on the individual demographic characteristics of the subject. Here is a list and sample statistics:

```
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
      Female |       261    .4444444    .4978587          0          1
       Black |       261     .091954    .2895162          0          1
    Hispanic |       261    .1302682     .337245          0          1
         Age |       261     20.4751    3.695677         17         47
    Business |       261    .4597701    .4993364          0          1
      GPAlow |       261    .4750958    .5003388          0          1
```

So if we only included the first two characteristics, to illustrate, the estimate of r, $\hat{r}$, would actually be

$$\hat{r} = \hat{r}_0 + (\hat{r}_{FEMALE} \times \text{FEMALE}) + (\hat{r}_{BLACK} \times \text{BLACK}), \tag{5}$$

where $\hat{r}_0$ is the estimate of the constant. If we collapse this specification by dropping all individual characteristics, we would simply be estimating the constant terms for each of r. The earlier command line is changed slightly at the “=)” part to read “= Female Black Hispanic Age Business GPAlow)”, and no changes are made to ML_eut0. The results are as follows:

```
. ml model lf ML_eut0 (r: Choices P0left P1left P2left P3left P0right P1right P2right P3right prize0 prize1 prize2
prize3 stake = $demog ), cluster(id) technique(nr) maximize

initial:       log pseudolikelihood =  -10201.74
alternative:   log pseudolikelihood = -10058.855
rescale:       log pseudolikelihood = -10058.855
Iteration 0:   log pseudolikelihood = -10058.855  (not concave)
Iteration 1:   log pseudolikelihood = -9840.8942
Iteration 2:   log pseudolikelihood =  -9662.873
Iteration 3:   log pseudolikelihood = -9437.1581
Iteration 4:   log pseudolikelihood = -9314.3859
Iteration 5:   log pseudolikelihood = -9311.6301
Iteration 6:   log pseudolikelihood =  -9311.596
Iteration 7:   log pseudolikelihood =  -9311.596
                                                  Number of obs   =      14718
                                                  Wald chi2(6)    =      53.91
Log pseudolikelihood =  -9311.596                 Prob > chi2     =     0.0000
                                   (Std. Err. adjusted for 240 clusters in id)
------------------------------------------------------------------------------
             |               Robust
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      Female |  -.0843659   .0348788    -2.42   0.016    -.1527271   -.0160046
       Black |  -.1492568   .0678122    -2.20   0.028    -.2821663   -.0163473
    Hispanic |  -.1948952   .0765465    -2.55   0.011    -.3449235   -.0448669
         Age |   .0193893   .0035911     5.40   0.000     .0123508    .0264277
    Business |  -.0126035   .0330866    -0.38   0.703     -.077452    .0522451
      GPAlow |   .0337201   .0309452     1.09   0.276    -.0269314    .0943715
       _cons |   .4536359   .0811465     5.59   0.000     .2945916    .6126802
------------------------------------------------------------------------------
```

So we see that the CRRA coefficient changes from $\hat{r}$ = 0.792 to $\hat{r}$ = 0.454 - 0.084×Female - 0.149×Black ... and so on. We can quickly find out what the average value of $\hat{r}$ is when we evaluate this model using the actual characteristics of each subject and the estimated coefficients:

```
. predictnl r=xb(r)
. summ r if task==1
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
           r |       261    .7842491    .1169373    .510778   1.267962
```

So the average value is 0.784, close to the earlier estimate of 0.792. Thus we have a richer characterization of risk attitudes around roughly the same mean.

B. Extending the Economic Specification to Include Loss Aversion and Probability Weighting

It is a simple matter to specify different economic models. To illustrate, consider the extension to allow for loss aversion and probability weighting, two components of Prospect Theory. Tversky and Kahneman [1992] assume a power utility function defined separately over gains and losses:

$$U(x) = x^{\alpha} \text{ if } x \geq 0 \tag{6a}$$

$$U(x) = -\lambda(-x)^{\beta} \text{ if } x < 0. \tag{6b}$$

So α and β are the parameters controlling the curvature[7] of the utility functions, and λ is the coefficient of loss aversion.

There are two variants of prospect theory, depending on the manner in which the probability weighting function is combined with utilities. The original version proposed by Kahneman and Tversky [1979] posits some weighting function which is separable in outcomes, and has been usefully termed Separable Prospect Theory (SPT) by Camerer and Ho [1994; p. 185]. The alternative version is Cumulative Prospect Theory (CPT), proposed by Tversky and Kahneman [1992]. CPT posits a weighting function defined over the cumulative

[7] It is tempting, but false, to refer to these as "risk aversion parameters." It is true that under EUT the curvature of the utility function defines risk attitudes, but this is not generally the case for non-EUT specifications. The reason under Prospect Theory is that risk aversion depends on the curvature of the utility functions, the loss aversion parameter, and the extent of probability weighting.

probability distributions so that the decision weights on each outcome are rank-dependent. In either case, the weighting function proposed by Tversky and Kahneman [1992] has been widely used: it assumes weights

$$w(p) = p^\gamma / [\, p^\gamma + (1-p)^\gamma \,]^{1/\gamma}. \tag{7}$$

Alternative functional forms for the probability weighting function are available: in fact, there is a veritable "menagerie" of functional forms, to borrow the felicitous expression of Stott [2006], who provides a convenient taxonomy.

Assuming that SPT is the true model for the moment, prospective utility PU is defined in much the same manner as when EUT is assumed to be the true model. The PT utility function is used instead of the EUT utility function, and w(p) is used instead of p, but the steps are otherwise identical.[8] The same error process is assumed to apply when the subject forms the preference for one lottery over the other. Thus the difference in prospective utilities is defined similarly as $\nabla PU = PU_R - PU_L$. The likelihood, conditional on the SPT model being true, depends on the estimates of α, β, λ and γ given the above specification and the observed choices. Here is the code that implements this specification. Note how much of it is similar to ML_eut0, and the few differences:

```
* define KT 1992 SPT specification with no errors
program define MLkt0

    args lnf alpha beta lambda gamma
    tempvar choice prob0l prob1l prob2l prob3l prob0r prob1r prob2r prob3r
    tempvar m0 m1 m2 m3 y0 y1 y2 y3 euL euR euDiff euRatio tmp

    quietly {

        generate int `choice' = $ML_y1

        generate double `tmp'    = (($ML_y2^`gamma')+($ML_y3^`gamma')+($ML_y4^`gamma')+($ML_y5^`gamma'))^(1/`gamma')
        generate double `prob0l' = ($ML_y2^`gamma')/`tmp'
        generate double `prob1l' = ($ML_y3^`gamma')/`tmp'
        generate double `prob2l' = ($ML_y4^`gamma')/`tmp'
```

[8] This is assuming that there are more than two non-zero prizes, or that there are some positive and negative prizes in the same lottery. If there are only two non-zero prizes, and all prizes are gains or all are losses, then one of the editing processes proposed by Kahneman and Tversky [1979; p.276, eq. (2)] means that SPT is the same as CPT.

```
        generate double `prob3l' = ($ML_y5^`gamma')/`tmp'

        replace `tmp'            = (($ML_y6^`gamma')+($ML_y7^`gamma')+($ML_y8^`gamma')+($ML_y9^`gamma'))^(1/`gamma')
        generate double `prob0r' = ($ML_y6^`gamma')/`tmp'
        generate double `prob1r' = ($ML_y7^`gamma')/`tmp'
        generate double `prob2r' = ($ML_y8^`gamma')/`tmp'
        generate double `prob3r' = ($ML_y9^`gamma')/`tmp'

        generate double `m0' = $ML_y10
        generate double `m1' = $ML_y11
        generate double `m2' = $ML_y12
        generate double `m3' = $ML_y13

        generate double `y0' = .
        replace `y0' =           ( `m0')^(`alpha') if `m0'>=0
        replace `y0' = -`lambda'*(-`m0')^(`beta')  if `m0'<0

        generate double `y1' = .
        replace `y1' =           ( `m1')^(`alpha') if `m1'>=0
        replace `y1' = -`lambda'*(-`m1')^(`beta')  if `m1'<0

        generate double `y2' = .
        replace `y2' =           ( `m2')^(`alpha') if `m2'>=0
        replace `y2' = -`lambda'*(-`m2')^(`beta')  if `m2'<0

        generate double `y3' = .
        replace `y3' =           ( `m3')^(`alpha') if `m3'>=0
        replace `y3' = -`lambda'*(-`m3')^(`beta')  if `m3'<0

        generate double `euL' = (`prob0l'*`y0')+(`prob1l'*`y1')+(`prob2l'*`y2')+(`prob3l'*`y3')
        generate double `euR' = (`prob0r'*`y0')+(`prob1r'*`y1')+(`prob2r'*`y2')+(`prob3r'*`y3')

        generate double `euDiff' = `euR' - `euL'

        replace `lnf' = ln(normal( `euDiff')) if `choice'==1
        replace `lnf' = ln(normal(-`euDiff')) if `choice'==0
    }
end
```

The first thing to notice is that the initial line "`args lnf alpha beta lambda gamma`" has more parameters than ML_eut0. The "lnf" parameter is the same, since it is the one used to return the value of the likelihood function for trial values of the other parameters. But we now have four parameters instead of just one.

The body of the program runs through the same logic, just with some extra steps. This would be an excellent exercise to work through, since it is supposed to match the economic specification presented above.

When we estimate this model we get this output:

```
. ml model lf MLkt0 (alpha: Choices P0left P1left P2left P3left P0right P1right P2right P3right prize0 prize1 prize2
prize3 = ) (beta: ) (lambda: ) (gamma: ), cluster(id) technique(nr) maximize

                                                  Number of obs   =      14718
                                                  Wald chi2(0)    =          .
Log pseudolikelihood = -9199.9139                 Prob > chi2     =          .

                                   (Std. Err. adjusted for 240 clusters in id)
------------------------------------------------------------------------------
             |               Robust
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
alpha        |
       _cons |   .6983474   .0213405    32.72   0.000     .6565207     .740174
-------------+----------------------------------------------------------------
beta         |
       _cons |   .7928385   .0579259    13.69   0.000     .6793058    .9063711
-------------+----------------------------------------------------------------
lambda       |
       _cons |   .6313889   .0994708     6.35   0.000     .4364297     .826348
-------------+----------------------------------------------------------------
gamma        |
       _cons |   .9194815    .028761    31.97   0.000     .8631111     .975852
------------------------------------------------------------------------------
```

So we get estimates for all four parameters. *Stata* used the variable "_cons" for the constant, and since there are no characteristics here, that is the only variable to be estimated. It would be a simple extension to add demographic or other characteristics to any or all of these four parameters. We see that the utility curvature coefficients α and β are similar, and indicate concavity in the gain domain and convexity in the loss domain. The loss aversion parameter λ is less than 1, which is a blow for PT devotees since "loss aversion" calls for λ>1.[9] And γ is

[9] At this point we often encounter suspicious questions from behavioralists, convinced that something must be wrong with the code, the logic of estimation, or some of the functional forms assumed, since loss aversion *has* to be positive. But our job here is just to report the facts, to which one is then free to confront with dogmatic priors if one truly entertains such things. A more interesting prior from theory is that the loss aversion parameter λ should be greater than or equal to 1: we admit loss aversion, loss neutrality, but not loss seeking. We discuss how to incorporate such a prior into the estimation later. A formal Bayesian analysis would call for the systematic pooling of priors and likelihoods, and is an attractive option in experimental economics (e.g., Harrison [1990]), so we do not want to discourage the formation and evaluation of priors, just to be aware that they are a prior belief and not a logical truth.

very close to 1, which is the value that implies that w(p)=p for all p, the EUT case. We should formally test these hypotheses, and here is how we do it:

```
. test [alpha]_cons=[beta]_cons

 ( 1)  [alpha]_cons - [beta]_cons = 0

           chi2(  1) =    2.64
         Prob > chi2 =    0.1043

. test [lambda]_cons=1

 ( 1)  [lambda]_cons = 1

           chi2(  1) =   13.73
         Prob > chi2 =    0.0002

. test [gamma]_cons=1

 ( 1)  [gamma]_cons = 1

           chi2(  1) =    7.84
         Prob > chi2 =    0.0051
```

So we see that PT is not doing so well here in relation to the *a priori* beliefs about loss aversion it comes packaged with, and that the deviation in λ is indeed statistically significant. But γ is less than 1, so things are not so bad in terms of those priors.

We turn now to the rank dependent specification of CPT. The idea of rank-dependent preferences had two important precursors.[10] In economics Quiggin [1982] had formally presented the general case in which one allowed for subjective probability weighting in a rank-dependent manner and allowed non-linear utility functions. This branch of the family tree of choice models has become known as Rank-Dependent Utility (RDU). The Yaari [1987] model can be seen as a pedagogically important special case in which utility is linear in money, and can be called Rank-Dependent Expected Value (RDEV). The other precursor, in psychology, is Lopes [1984]. Her concern was

[10] Of course, many others recognized the basic point that the distribution of outcomes mattered for choice in some holistic sense. Allais [1979; p.54] was quite clear about this, in a translation of his original 1952 article in French. Similarly, in psychology it is easy to find citations to kindred work in the 1960's and 1970's by Lichtenstein, Coombs and Payne, inter alia.

motivated by clear preferences that her experimental subjects exhibited for lotteries with the same expected value but alternative *shapes* of probabilities, as well as the verbal protocols those subjects provided as a possible indicator of their latent decision processes. The notion of rank-dependent decision weights was incorporated into the sign-dependent SPT initially by Starmer and Sugden [1989], and then by Luce and Fishburn [1991] and Tversky and Kahneman [1992].

From the perspective of writing out a ML program, adding rank-dependent decision weights is a little tedious from a coding perspective, but not conceptually difficult. Formally, to calculate decision weights under RDU for a lottery of n prizes one replaces expected utility

$$EU_i = \sum_{k=1,n} [p_k \times u_k] \tag{2}$$

with RDU

$$RDU_i = \sum_{k=1,n} [w_k \times u_k] \tag{8}$$

where

$$w_i = \omega(p_i + ... + p_n) - \omega(p_{i+1} + ... + p_n) \tag{9a}$$

for i=1,... , n-1, and

$$w_i = \omega(p_i) \tag{9b}$$

for i=n, where the subscript indicates outcomes ranked from worst to best, and $\omega(p)$ is some probability weighting function. So the probability weighting function $\omega(.)$ is applied to the aggregated probabilities, and the decision weights w_i then derived by the differences in these transformed aggregate probabilities.

Our code for the RDU model exploits the fact we know that we have up to 4 outcomes in each lottery in these data, and that the prizes are already ordered. The former would just entail the use of more general, and perhaps less transparent code, to allow for varying numbers of outcomes (the number of outcomes is not the issue, but when there are more than 4 outcomes it becomes inefficient to write the expressions out “long hand”). The latter could be handled by sorting the data after reading it in. But the basic logic, apart from the mess of some of the computational steps to get to the decision weights, is the same as earlier examples:

```
* define RDEU specification with no errors
program define MLrdeu0
    args lnf alpha gamma

    tempvar  choice m0 m1 m2 m3 y0 y1 y2 y3 euL euR euDiff euRatio tmp endow
    tempvar  a_prob0l a_prob1l a_prob2l a_prob3l a_prob0r a_prob1r a_prob2r a_prob3r
    tempvar  pw_prob0l pw_prob1l pw_prob2l pw_prob3l pw_prob0r pw_prob1r pw_prob2r pw_prob3r
    tempvar  dw_prob0l dw_prob1l dw_prob2l dw_prob3l dw_prob0r dw_prob1r dw_prob2r dw_prob3r
    tempvar  dw_check_l dw_check_r

    quietly {

        * read in data
        generate int `choice' = $ML_y1

        * generate the aggregate probabilities
        generate double `a_prob3l' = $ML_y5
        generate double `a_prob2l' = $ML_y4
        replace         `a_prob2l' = `a_prob2l' + `a_prob3l' if `a_prob2l'>0
        generate double `a_prob1l' = $ML_y3
        replace         `a_prob1l' = `a_prob1l' + `a_prob2l' if `a_prob1l'>0
        replace         `a_prob1l' = `a_prob1l' + `a_prob3l' if `a_prob1l'>0 & `a_prob2l'==0
        generate double `a_prob0l' = $ML_y2
        replace         `a_prob0l' = `a_prob0l' + `a_prob1l' if `a_prob0l'>0
        replace         `a_prob0l' = `a_prob0l' + `a_prob2l' if `a_prob0l'>0 & `a_prob1l'==0
        replace         `a_prob0l' = `a_prob0l' + `a_prob3l' if `a_prob0l'>0 & `a_prob1l'==0 & `a_prob2l'==0

        * generate the weighted probabilities
        generate double `pw_prob0l' = (`a_prob0l'^`gamma')/(`a_prob0l'^`gamma' + (1-`a_prob0l')^`gamma')^(1/`gamma')
        generate double `pw_prob1l' = (`a_prob1l'^`gamma')/(`a_prob1l'^`gamma' + (1-`a_prob1l')^`gamma')^(1/`gamma')
        generate double `pw_prob2l' = (`a_prob2l'^`gamma')/(`a_prob2l'^`gamma' + (1-`a_prob2l')^`gamma')^(1/`gamma')
        generate double `pw_prob3l' = (`a_prob3l'^`gamma')/(`a_prob3l'^`gamma' + (1-`a_prob3l')^`gamma')^(1/`gamma')

        * Generate the decision weights
        generate double `dw_prob3l' = 0
        replace         `dw_prob3l' = `pw_prob3l' if `pw_prob3l' > 0
        generate double `dw_prob2l' = 0
        replace         `dw_prob2l' = `pw_prob2l' if `pw_prob2l' > 0
        replace         `dw_prob2l' = `pw_prob2l'-`pw_prob3l' if `dw_prob3l' > 0 & `pw_prob2l' > 0
        generate double `dw_prob1l' = 0
        replace         `dw_prob1l' = `pw_prob1l' if `pw_prob1l' > 0
        replace         `dw_prob1l' = `pw_prob1l'-`pw_prob2l' if `dw_prob2l' > 0 & `pw_prob1l' > 0
        replace         `dw_prob1l' = `pw_prob1l'-`pw_prob3l' if `dw_prob3l' > 0 & `pw_prob2l'== 0 & `pw_prob1l' > 0
        generate double `dw_prob0l' = 0
        replace         `dw_prob0l' = `pw_prob0l' if `pw_prob0l' > 0
        replace         `dw_prob0l' = `pw_prob0l'-`pw_prob1l' if `dw_prob1l' > 0 & `pw_prob0l' > 0
        replace         `dw_prob0l' = `pw_prob0l'-`pw_prob2l' if `dw_prob2l' > 0 & `pw_prob1l'== 0 & `pw_prob0l' > 0
        replace         `dw_prob0l' = `pw_prob0l'-`pw_prob3l' if `dw_prob3l' > 0 & `pw_prob2l'== 0 & `pw_prob1l'== 0 &
`pw_prob0l' > 0

        * Check Decision weights
        generate double `dw_check_l'  = 0
        replace `dw_check_l' = `dw_prob3l' + `dw_prob2l' + `dw_prob1l' + `dw_prob0l'
        replace dw_check_l = `dw_check_l'

        * Now do the other lottery
        * generate the aggregate probabilities
        generate double `a_prob3r' = $ML_y9
        generate double `a_prob2r' = $ML_y8
        replace         `a_prob2r' = `a_prob2r' + `a_prob3r' if `a_prob2r'>0
        generate double `a_prob1r' = $ML_y7
        replace         `a_prob1r' = `a_prob1r' + `a_prob2r' if `a_prob1r'>0
        replace         `a_prob1r' = `a_prob1r' + `a_prob3r' if `a_prob1r'>0 & `a_prob2r'==0
        generate double `a_prob0r' = $ML_y6
        replace         `a_prob0r' = `a_prob0r' + `a_prob1r' if `a_prob0r'>0
        replace         `a_prob0r' = `a_prob0r' + `a_prob2r' if `a_prob0r'>0 & `a_prob1r'==0
        replace         `a_prob0r' = `a_prob0r' + `a_prob3r' if `a_prob0r'>0 & `a_prob1r'==0 & `a_prob2r'==0

        * generate the weighted probabilities
        generate double `pw_prob0r' = (`a_prob0r'^`gamma')/(`a_prob0r'^`gamma' + (1-`a_prob0r')^`gamma')^(1/`gamma')
        generate double `pw_prob1r' = (`a_prob1r'^`gamma')/(`a_prob1r'^`gamma' + (1-`a_prob1r')^`gamma')^(1/`gamma')
        generate double `pw_prob2r' = (`a_prob2r'^`gamma')/(`a_prob2r'^`gamma' + (1-`a_prob2r')^`gamma')^(1/`gamma')
        generate double `pw_prob3r' = (`a_prob3r'^`gamma')/(`a_prob3r'^`gamma' + (1-`a_prob3r')^`gamma')^(1/`gamma')

        * Generate the decision weights
        generate double `dw_prob3r' = 0
        replace         `dw_prob3r' = `pw_prob3r' if `pw_prob3r' > 0
        generate double `dw_prob2r' = 0
        replace         `dw_prob2r' = `pw_prob2r' if `pw_prob2r' > 0
        replace         `dw_prob2r' = `dw_prob2r'-`pw_prob3r' if `dw_prob3r' > 0 & `pw_prob2r' > 0
        generate double `dw_prob1r' = 0
        replace         `dw_prob1r' = `pw_prob1r' if `pw_prob1r' > 0
        replace         `dw_prob1r' = `pw_prob1r'-`pw_prob2r' if `dw_prob2r' > 0 & `pw_prob1r' > 0
        replace         `dw_prob1r' = `pw_prob1r'-`pw_prob3r' if `dw_prob3r' > 0 & `pw_prob2r'== 0 & `pw_prob1r' > 0
        generate double `dw_prob0r' = 0
        replace         `dw_prob0r' = `pw_prob0r' if `pw_prob0r' > 0
        replace         `dw_prob0r' = `pw_prob0r'-`pw_prob1r' if `dw_prob1r' > 0 & `pw_prob0r' > 0
        replace         `dw_prob0r' = `pw_prob0r'-`pw_prob2r' if `dw_prob2r' > 0 & `pw_prob1r'== 0 & `pw_prob0r' > 0
        replace         `dw_prob0r' = `pw_prob0r'-`pw_prob3r' if `dw_prob3r' > 0 & `pw_prob2r'== 0 & `pw_prob1r'== 0 &
`pw_prob0r' > 0

        * Check Decision weights
        generate double `dw_check_r'  = 0
        replace `dw_check_r' = `dw_prob3r' + `dw_prob2r' + `dw_prob1r' + `dw_prob0r'
        replace dw_check_r = `dw_check_r'

        foreach X in l r {
        foreach y in 0 1 2 3 {
            replace RDa_prob`y'`X' = `a_prob`y'`X''
            replace RDpw_prob`y'`X' = `pw_prob`y'`X''
            replace RDdw_prob`y'`X' = `dw_prob`y'`X''
        }
        }

        * Calculate utility values
        * construct the argument of the utility function
        generate double `endow' = $ML_y14
        generate double `m0' = $ML_y10
        generate double `m1' = $ML_y11
        generate double `m2' = $ML_y12
        generate double `m3' = $ML_y13
        replace `m0' = `endow'+`m0'
        replace `m1' = `endow'+`m1'
        replace `m2' = `endow'+`m2'
        replace `m3' = `endow'+`m3'

        generate double `y0' = (`m0')^(`alpha')
        generate double `y1' = (`m1')^(`alpha')
        generate double `y2' = (`m2')^(`alpha')
        generate double `y3' = (`m3')^(`alpha')

        * calculate expected utility values and EU indices
        generate double `euL' = (`dw_prob0l'*`y0')+(`dw_prob1l'*`y1')+(`dw_prob2l'*`y2')+(`dw_prob3l'*`y3')
        generate double `euR' = (`dw_prob0r'*`y0')+(`dw_prob1r'*`y1')+(`dw_prob2r'*`y2')+(`dw_prob3r'*`y3')

        generate double `euDiff' = `euR' - `euL'

        * likelihood
        replace `lnf' = ln(normal( `euDiff')) if `choice'==1
        replace `lnf' = ln(normal(-`euDiff')) if `choice'==0

        replace ll=`lnf'

    }
end
```

The estimates from this model, focusing solely on the gain-frame lotteries in which all outcomes were non-negative, are as follows:

```
. ml model lf MLrdeu0 (alpha: Choices P0left P1left P2left P3left P0right P1right P2right P3right prize0 prize1 prize2
prize3 = ) (gamma: ), cluster(id) technique(nr) init(0.5 1, copy) maximize
```

```
                                                  Number of obs   =       6191
                                                  Wald chi2(0)    =          .
Log pseudolikelihood = -4090.6375                 Prob > chi2     =          .

                                   (Std. Err. adjusted for 120 clusters in id)
------------------------------------------------------------------------------
             |               Robust
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
alpha        |
       _cons |    .615686   .0364702    16.88   0.000     .5442058    .6871662
-------------+----------------------------------------------------------------
gamma        |
       _cons |   .9499579    .063464    14.97   0.000     .8255707    1.074345
------------------------------------------------------------------------------
```

So we see evidence of concave utility (α<1) and probability weighting ($\gamma \neq 1$). The evidence for probability weighting is slight, and is not statistically significant: a Wald test of the null hypothesis that γ=1 has a *p*-value of 0.43, so we cannot reject the null. Nonetheless, Figure 2 displays the estimated probability weighting function in the left panel, and examples of the implied decision weights in the right panel. The decision weights are calculated assuming a lottery with equal probabilities of each outcome, to see the pure effect of the probability transformation. One example assumes 2 prizes, and the other example assumes 5 prizes. Given the estimated probability weighting function, convex for low probabilities and concave for higher probabilities, we infer a concave decision weight function, with extreme prizes receiving slightly less weight than the interior prizes. Specifically, for the 2-prize case, the weights are 0.499 and 0.501 instead of 0.5 and 0.5 under EUT; for the 5-prize case, the weights are 0.211, 0.193, 0.190, 0.193 and 0.212 instead of 0.2, 0.2, 0.2, 0.2 and 0.2, respectively. The decision weights under RDU always sum to 1.

The probability weighting function estimated here does have the shape that is normally assumed, since γ<1. But again we do not want dogmatic priors to get in the way at this stage, since there are several other factors to consider in the statistical specification of utility functions, such as stochastic errors in the latent choice process. We return to consider these estimates when we have extended the estimation methods to consider such errors, in the next section.

The extension to consider both rank-dependent *and* sign-dependent preferences using CPT is tedious to code, but conceptually straightforward given the templates provided by our SPT and RDU models. The tedium comes from keeping track of gains and losses, and the ranking of each. We include a template to estimate the CPT model in the software accompanying these notes, and it provides the following estimates:

```
. ml model lf MLcpt0 (alpha: Choices P0left P1left P2left P3left P0right P1right P2right P3right prize0 prize1 prize2
prize3 stake= ) (beta: ) (lambda: ) (gamma: ), cluster(id) technique(nr) maximize

                                                  Number of obs   =      14718
                                                  Wald chi2(0)    =          .
Log pseudolikelihood = -9204.3636                 Prob > chi2     =          .
                                   (Std. Err. adjusted for 240 clusters in id)
------------------------------------------------------------------------------
             |               Robust
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
alpha        |
       _cons |   .6971163   .0211916    32.90   0.000     .6555815    .7386512
-------------+----------------------------------------------------------------
beta         |
       _cons |   .7967744   .0567211    14.05   0.000     .6856031    .9079456
-------------+----------------------------------------------------------------
lambda       |
       _cons |   .6119571   .0952587     6.42   0.000     .4252534    .7986607
-------------+----------------------------------------------------------------
gamma        |
       _cons |   .9307044   .0285199    32.63   0.000     .8748064    .9866025
------------------------------------------------------------------------------
```

So we find estimates for CPT that are virtually the same as those obtained with SPT.

C. Adding Stochastic Errors

One popular feature of structural models of choice uncertainty is to allow for subjects to make a stochastic error. There are various specifications in use: some place the error at the stage at which the subject forms the expected utility, some at the stage when the subject compares the expected utilities, and some at the stage when the subject makes a choice. Here we illustrate one method for adding a stochastic error at the second stage, originally due to Luce and popularized by Holt and Laury [2002]. We return to base camp, the ML_eut0 program, and simply make two changes. We augment the arguments by one parameter, μ, to be estimated:

```
        args lnf r mu
```

and then we revise the line defining the EU difference from

```
        generate double `euDiff' = `euR' - `euL'
```

to

```
        replace `euDiff' = (`euR'^(1/`mu'))/((`euR'^(1/`mu'))+(`euL'^(1/`mu')))
```

So this changes the latent preference index from being the difference in the EU of each lottery to the ratio of the EU of one lottery to the sum of the EU values of all lotteries, which is just the change from (3) to (3') noted earlier. But it is also adds the $1/\mu$ exponent to each expected utility. Thus as $\mu \rightarrow 0$ this specification collapses to the deterministic choice EUT model, where the choice is strictly determined by the EU of the two lotteries; but as μ gets larger and larger the choice essentially becomes random.[11]

Apart from this change in the program, there is nothing extra that is needed. You just add one more parameter in the "ml model" stage, as we did for the PT extensions. In fact, Holt and Laury [2002] cleverly exploit the fact that the latent preference index defined above is already in the form of a cumulative probability density function, since it ranges from 0 to 1, and is equal to ½ when the subject is indifferent between the two lotteries. Thus, instead of defining the likelihood contribution by

```
         replace `lnf' = ln(normal( `euDiff')) if `choice'==1
         replace `lnf' = ln(normal(-`euDiff')) if `choice'==0
```

we can use

[11] A useful exercise for students is to check this in a spreadsheet: as μ gets larger, the value of `euDiff' approaches ½ for *any* values of the expected utilities.

```
    replace `lnf' = ln(`euDiff') if `choice'==1

    replace `lnf' = ln(1-`euDiff') if `choice'==0
```

instead. We illustrate this error specification later, when we replicate the Holt and Laury [2002] estimation.

There is one other important error specification, due originally to Fechner and popularized by Hey and Orme [1994]. This ends up being simple to code, but involves some extra steps in the economics.[12] Again we add an error term "noise" to the arguments of the program, as above, and now we have the latent index

```
   generate double `euDiff' = (`euR' - `euL')/`noise'
```

instead of the original

```
   generate double `euDiff' = `euR' - `euL'
```

Here are the results:

```
. ml model lf ML_eut (r: Choices P0left P1left P2left P3left P0right P1right P2right P3right prize0 prize1 prize2 prize3
stake = ) (noise: ), cluster(id) technique(nr) maximize

                                                  Number of obs   =      14718
                                                  Wald chi2(0)    =          .
Log pseudolikelihood = -9512.3711                 Prob > chi2     =          .

                                   (Std. Err. adjusted for 240 clusters in id)
------------------------------------------------------------------------------
             |               Robust
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
r            |
```

[12] Let euA and euB denote the EU of lotteries A and B, respectively. If the subject does not make mistakes then A is chosen if euA-euB > 0, and otherwise B is chosen. If the subject makes measurement errors, denoted by ε, then the decision is made on the basis of the value of euA-euB+ε. That is, A is chosen if euA-euB+ε > 0, and otherwise B is chosen. If ε is random then the probability that A is chosen = P(euA-euB+ε>0) = P(ε>-(euA-euB)). Now suppose that ε is normally distributed with mean 0 and standard deviation σ, then it follows that Z=ε/σ is normally distributed with mean 0 and standard deviation 1: in other words, Z has a unit normal distribution. Hence the probability that A is chosen is P(ε > -(euA-euB)) = P(ε/σ > -(euA-euB)/σ). Using *Stata* notation it follows that the probability that A is chosen is 1-normal(-(euA-euB)/σ) = normal((euA-euB)/σ), since the distribution is symmetrical about 0. Hence the probability that B is chosen is given by normal(-(euA-euB)/σ) = 1-normal((euA-euB)/σ). Implicit here is the assumption that the probability that the subject is indifferent is zero. Hence, if we denote by y the decision of the subject with y=0 indicating that A was chosen and y=1 indicating that B was chosen, then the likelihood is normal((euA-euB)/σ) if y=0 and 1-normal((euA-euB)/σ) if y=1.

```
       _cons |   .8020972   .0335822    23.88   0.000     .7362774     .867917
-------------+----------------------------------------------------------------
noise        |
       _cons |   1.061093    .127369     8.33   0.000     .8114544    1.310732
------------------------------------------------------------------------------
```

So the CRRA coefficient increases very slightly, and the noise term is estimated as a normal distribution with mean 0 and standard deviation of 1.06.

Using the Fechner latent error story, we can re-consider the estimates of the RDU model. Making the obvious modifications, and again using only the gain-frame responses, we estimate

```
. ml model lf MLrdeu (alpha: Choices P0left P1left P2left P3left P0right P1right P2right P3right prize0 prize1 prize2
prize3 = ) (gamma: ) (mu: ), cluster(id) technique(nr) init(0.5 1 1, copy) maximize

                                                  Number of obs   =       6191
                                                  Wald chi2(0)    =          .
Log pseudolikelihood = -3998.8062                 Prob > chi2     =          .

                                   (Std. Err. adjusted for 120 clusters in id)
------------------------------------------------------------------------------
             |               Robust
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
alpha        |
       _cons |   .5358161   .0264942    20.22   0.000     .4838885    .5877437
-------------+----------------------------------------------------------------
gamma        |
       _cons |   .9814947   .0066665   147.23   0.000     .9684287    .9945607
-------------+----------------------------------------------------------------
mu           |
       _cons |   .3965074   .0362864    10.93   0.000     .3253874    .4676275
------------------------------------------------------------------------------

. test [gamma]_cons=1

 ( 1)  [gamma]_cons = 1

           chi2(  1) =    7.71
         Prob > chi2 =    0.0055
```

Again, the estimated $\gamma<1$, as expected by behavioralists, but it is more precisely estimated than when we assumed no Fechner error story.

We can test if it is significantly different from the EUT null hypothesis, and it is with a two-sided *p*-value of 0.0055. There is evidence of greater concavity of the utility function, with the estimate of α dropping from 0.61 to 0.53. Figure 3 shows the estimated probability weighting function and implied decision weights, and is not likely to be more comforting to behavioralists than Figure 2, since extreme prizes receive only slightly greater weight. The level of probability weighting is not as large as some might expect, even if the deviation from the EUT assumption that γ=1 is statistically significant. We come back to this issue later, when we consider mixture processes that might be at work in these data.

It is a simple matter to add point constraints on parameters to be estimated. For example, the RDEV model is a special case of the RDU model estimated here in which α=1. So we can use the same ML code to estimate RDEV, by just adding the constraint

```
constraint define 1 [alpha]_cons=1
```

and then estimating the same model with the constraint included (in bold in the command line):

```
. ml model lf MLrdeu (alpha: Choices P0left P1left P2left P3left P0right P1right P2right P3right prize0 prize1 prize2
prize3 = ) (gamma: ) (mu: ), cluster(id) technique(nr) init(0.5 1 1, copy) maximize constraints(1)

                                                  Number of obs   =       6191
                                                  Wald chi2(0)    =          .
Log pseudolikelihood = -4247.1438                 Prob > chi2     =          .

 ( 1)  [alpha]_cons = 1
                                  (Std. Err. adjusted for 120 clusters in id)
------------------------------------------------------------------------------
             |               Robust
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
alpha        |
       _cons |          1          .        .       .            .           .
-------------+----------------------------------------------------------------
gamma        |
       _cons |   1.909885   1.518126     1.26   0.208    -1.065588    4.885358
-------------+----------------------------------------------------------------
mu           |
       _cons |   7.743446   1.235278     6.27   0.000     5.322347    10.16455
------------------------------------------------------------------------------

. test [gamma]_cons=1
```

```
 ( 1)  [gamma]_cons = 1

           chi2(  1) =    0.36
         Prob > chi2 =    0.5489
```

Thus we see some evidence of probability weighting, since concavity of the utility function is ruled out *a priori*. O n hte other hand, one cannot reject the null hypothesis that γ=1 at conventional significance levels. Figure 4 shows the implied probability weighting function and decision weights, using the point estimates for γ. The log-likelihood is much larger than the log-likelihood for the RDU model, suggesting that the RDEV constraint on RDU would be easily rejected.

D. Non-Parametric Estimation of the EUT Model

It is possible to estimate the EUT model without assuming a functional form for utility, following Hey and Orme [1994]. We also use the Fechner noise specification just introduced, and restrict attention to the experimental tasks in the "gain frame" in order to keep the number of parameters to be estimated reasonably small. In these tasks there were only four monetary prizes of $0, $5, $10 and $15. We normalize to u($0)=0 and u($15)=1, and estimate u($5), u($10) and the noise parameter. As explained by Hey and Orme [1994], one could normalize the noise parameter to some fixed value and then estimate u($15), but our choice of normalization seems the most natural. The likelihood function is evaluated as follows:

```
* define Original Recipe EUT with Fechner errors: non-parametric
program define ML_eut_np

    args lnf u5_ u10_ LNmu

    tempvar prob0l prob1l prob2l prob3l prob0r prob1r prob2r prob3r y0 y1 y2 y3 euL euR euDiff euRatio
    tempvar tmp lnf_eut lnf_pt p1 p2 f1 f2 u0 u15 mu u5 u10

    quietly {

        * define parameters
        generate double `mu'  = exp(`LNmu')
        generate double `u5'  =  1/(1+exp(`u5_'))
        generate double `u10' =  1/(1+exp(`u10_'))

        * data
        generate int `choice' = $ML_y1
        generate double `prob0l' = $ML_y2
        generate double `prob1l' = $ML_y3
        generate double `prob2l' = $ML_y4
        generate double `prob3l' = $ML_y5

        generate double `prob0r' = $ML_y6
        generate double `prob1r' = $ML_y7
        generate double `prob2r' = $ML_y8
        generate double `prob3r' = $ML_y9

        * evaluate EU
        generate double `u0'  = 0
        generate double `u15' = 1

        generate double `y0' = `u0'
        generate double `y1' = `u5'
        generate double `y2' = `u10'
        generate double `y3' = `u15'

        generate double `euL' = (`prob0l'*`y0')+(`prob1l'*`y1')+(`prob2l'*`y2')+(`prob3l'*`y3')
        generate double `euR' = (`prob0r'*`y0')+(`prob1r'*`y1')+(`prob2r'*`y2')+(`prob3r'*`y3')

        if "$fechner" == "y" {

            generate double `euDiff' = (`euR' - `euL')/`mu'
            replace `lnf' = ln($cdf( `euDiff')) if `choice'==1
            replace `lnf' = ln($cdf(-`euDiff')) if `choice'==0

        }
        else {

            generate double `euDiff' = (`euR'^(1/`mu'))/((`euR'^(1/`mu'))+(`euL'^(1/`mu')))
            replace `lnf' = ln(  `euDiff') if `choice'==1
            replace `lnf' = ln(1-`euDiff') if `choice'==0

        }
        replace ll=`lnf'
    }
end
```

and estimates can be obtained in the usual manner. There are several technical innovations in this program, apart from the use of a non-parametric specification.

First, we allow non-linear transforms of the core parameter to be estimated. This allows *Stata* to happily maximize over some variable that ranges from -∞ to +∞, but for the underlying parameter in the economic model to be constrained. The error term can be constrained to be non-negative, by estimating the (natural) log of the parameter and then converting to the underlying parameter we want using

```
        generate double `mu'  = exp(`LNmu')
```

Similarly, we can constrain the estimates of u5 and u10 to lie in the open interval (0,1) by estimating the log odds transform of the parameters we are really interested in, and converting using

```
        generate double `u5'  =  1/(1+exp(`u5_'))
        generate double `u10' =  1/(1+exp(`u10_'))
```

This transform is one of several used by statisticians; for example, see Rabe-Hesketh and Everitt [2004; p. 244]. We discuss more general "parameter bounds" later, building on this unit interval transform. When one has transformed variables the *Stata* commands "nlcom" or "predictnl" are used to get the correct coefficients and their standard errors. These commands use the Delta Method from statistics to calculate the standard errors correctly: see Oehlert [1992] for an exposition and historical note, and some examples of syntax below.

Second, we introduce the notion of a global macro function in *Stata*, to flag if we want to use the Fechner or Luce error specification introduced above. We use the command

```
       global fechner "y"
```

to request the Fechner specification, and then refer to this text using $fechner. Whenever *Stata* sees $fechner it substitutes whatever text is associated with it, *without* the double quotes.[13]

Third, we use a global $cdf to identify which cumulative probability density function to use. By setting

```
global cdf “normal”
```

we replicate the earlier commands, and by changing “normal” to “invlogit” we can quickly switch to evaluate an alternative.

Returning to the estimation of the non-parametric specification, we obtain initial pooled estimates as follows:

```
. ml model lf ML_eut_np (u5_: Choices P0left P1left P2left P3left P0right P1right P2right P3right prize0 prize1 prize2
prize3 stake = ) (u10_: ) (LNmu: ), cluster(id) technique(nr) maximize

                                                  Number of obs   =       6191
                                                  Wald chi2(0)    =          .
Log pseudolikelihood = -3981.7277                 Prob > chi2     =          .

                                   (Std. Err. adjusted for 120 clusters in id)
------------------------------------------------------------------------------
             |               Robust
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
u5_          |
       _cons |  -.2355648   .0648368    -3.63   0.000    -.3626425   -.1084871
-------------+----------------------------------------------------------------
u10_         |
       _cons |     -1.285   .0573084   -22.42   0.000    -1.397322   -1.172677
-------------+----------------------------------------------------------------
LNmu         |
       _cons |  -2.381347   .0636809   -37.39   0.000    -2.506159   -2.256534
------------------------------------------------------------------------------

. nlcom (u5: 1/(1+exp([u5_]_b[_cons]))) (u10: 1/(1+exp([u10_]_b[_cons]))) (mu: exp([LNmu]_b[_cons]) )

          u5:  1/(1+exp([u5_]_b[_cons]))
         u10:  1/(1+exp([u10_]_b[_cons]))
          mu:  exp([LNmu]_b[_cons])
```

[13] To return to the issue of debugging, one could define a global $debug as equal to “noisily” or “quietly” before issuing the ML estimation command, and then in the ML program have a line such as `$debug summ `this’` in order to display summary statistics of some variable. Thus there is no need to go into the ML program and manually tell it to display one variable or not, which can become tedious if one has numerous debugging displays.

```
------------------------------------------------------------------------------
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          u5 |   .5586204   .0159864    34.94   0.000     .5272876    .5899531
         u10 |   .7832996   .0097276    80.52   0.000     .7642338    .8023654
          mu |    .092426   .0058858    15.70   0.000     .0808901    .1039619
------------------------------------------------------------------------------
```

Here we see the use of the "nlcom" command to recover the core parameters of interest, since *Stata* estimated non-linear transforms of these. The utility of $5 is estimated to be 0.559, and the utility of $10 is estimated to be 0.783, recalling that we constrained the utility of $0 to be 0 and the utility of $15 to be 1.[14]

These estimates become more interesting if we include demographics for each parameter, defined using the global command

```
global demog "Female Black Hispanic Age Business GPAlow"
```

Again, the global command saves time in typing: every time *Stata* sees $demog after this command, it substitutes the string "Female Black Hispanic Age Business GPAlow" without the quotes. Hence we have the following results:

```
. ml model lf ML_eut_np (u5_: Choices P0left P1left P2left P3left P0right P1right P2right P3right prize0 prize1 prize2
prize3 stake = $demog ) (u10_: $demog ) (LNmu: ), cluster(id) technique(dfp) maximize difficult continue

                                                  Number of obs   =       6191
                                                  Wald chi2(6)    =      18.80
Log pseudolikelihood = -3899.1552                 Prob > chi2     =     0.0045
                                   (Std. Err. adjusted for 120 clusters in id)
------------------------------------------------------------------------------
             |               Robust
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
u5_          |
      Female |  -.3237969   .1294031    -2.50   0.012    -.5774222   -.0701715
       Black |  -.1685406   .1870545    -0.90   0.368    -.5351607    .1980796
    Hispanic |  -.2403506   .2034959    -1.18   0.238    -.6391953     .158494
         Age |   .0500134    .020636     2.42   0.015     .0095675    .0904592
    Business |  -.0815643   .1243081    -0.66   0.512    -.3252037    .1620751
```

[14] One advanced exercise would be to work out how one measures the degree of risk aversion from estimates such as these.

```
      GPAlow |  -.1128523   .1260811    -0.90   0.371    -.3599668    .1342621
       _cons |  -.9422493    .423902    -2.22   0.026    -1.773082   -.1114167
-------------+----------------------------------------------------------------
u10_         |
      Female |  -.2134069   .1148061    -1.86   0.063    -.4384226    .0116089
       Black |  -.1054572   .1811237    -0.58   0.560    -.4604531    .2495386
    Hispanic |  -.2819423   .1940966    -1.45   0.146    -.6623646      .09848
         Age |   .0411205   .0166579     2.47   0.014     .0084717    .0737694
    Business |    -.05746   .1101277    -0.52   0.602    -.2733063    .1583864
      GPAlow |   -.003044   .1109343    -0.03   0.978    -.2204713    .2143833
       _cons |  -1.936219    .338377    -5.72   0.000    -2.599426   -1.273013
-------------+----------------------------------------------------------------
LNmu         |
       _cons |  -2.416682   .0638027   -37.88   0.000    -2.541733   -2.291631
------------------------------------------------------------------------------

. nlcom (mu: exp([LNmu]_b[_cons]) )

          mu:  exp([LNmu]_b[_cons])

------------------------------------------------------------------------------
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          mu |   .0892171   .0056923    15.67   0.000     .0780604    .1003738
------------------------------------------------------------------------------
```

It is then possible to predict the values of the two estimated utilities, which will vary with the characteristics of each subject, and plot them.

To calculate the predicted values we use the "predictnl" command instead of the "nlcom" command:

```
. predictnl p_u5=1/(1+exp(xb(u5_)))

. predictnl p_u10=1/(1+exp(xb(u10_)))

. predictnl p_udiff=(1/(1+exp(xb(u10_)))) - (1/(1+exp(xb(u5_))))

. summ p_* if task==1 & e(sample)

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        p_u5 |       120    .5577782     .066356   .2683486   .6723887
       p_u10 |       120    .7833645    .0381008   .5681781   .8518088
     p_udiff |       120    .2255863     .031645   .1690824   .3021796
-------------+--------------------------------------------------------
```

The nice feature of the "predictnl" command is that it evaluates each characteristic for each observation using the corresponding coefficient estimate, providing a major time-saving compared to writing out the entire linear function using "nlcom" (and having to change it when you change the variables in the $demog list). We also add a quick check of weak monotonicity, the requirement that u($10)-u($5) ≥ 0, and see that all values of p_udiff are indeed positive.[15] Figure 5 shows kernel densities of the estimated distributed of values of the utilities, reflecting the variability across subjects picked up by the observable characteristics.

One can extend the transformation used to constrain parameters to lie between 0 and 1 into a way of constraining any parameter between two values a and b. It is possible to "hardwire" a and b into the likelihood, but a better way to do this would be to use globals to pass the values. To take an example, PT advocates tattoo the estimates of λ and γ from Tversky and Kahneman [1992] on their arm, just in case they forget the third of fourth significant decimal over dinner conversation. To say that they have degenerate priors about these parameters is to be kind. So to avoid offending your PT friends, you might constrain λ to be greater than 1 and less than 4, for example. We can do this by setting the two globals

```
global lambdaLO=1
global lambdaHI=4
```

and then changing the label in the "ml model" invocation to refer to lambda_ instead of lambda. Inside the likelihood evaluator, and before you do any economics in the likelihood function, transform lambda_ as follows

```
generate double `lambda' = $lambdaLO + ($lambdaHI-$lambdaLO)*(1/(1+exp(`lambda_')))
```

and then the code will work with lambda for the rest of the evaluation. Since the likelihood evaluator is free to dance between ±∞ it is numerically happy, and your lambda values will always stay between 1 and 4. When you get a solution be sure to use "nlcom" to transform the estimate of lambda_ back to be an estimate of lambda:

[15] An advanced exercise: devise a formulation in *Stata* to ensure that monotonicity is satisfied.

```
        nlcom (lambda: $lambdaLO+($lambdaHI-$lambdaLO)*(1/(1+exp([lambda_]_b[_cons]))) )
```

This method will generally work well. The exception is if the parameter wants to be at one of the boundaries, in which case the underlying likelihood function may become flat (since any smaller or larger value of the underlying pre-transformation parameter will suffice). In this case the time has come to re-think the economics or your priors, rather than torture the data further.

E. Alternative Functional Forms for Utility

It should be clear that it is a relatively easy matter to write out likelihood functions with alternative specifications of any function used so far. To illustrate, in an important context, consider the Expo-Power function proposed by Saha [1993] and used by Holt and Laury [2002] to show evidence of increasing RRA in lab experiments. The Expo-Power function can be defined as

$$u(y) = (1-\exp(-\alpha y^{1-r}))/\alpha, \tag{10}$$

where y is income and α and r are parameters to be estimated. Relative risk aversion (RRA) is then $r + \alpha(1-r)y^{1-r}$. So RRA varies with income if $\alpha \neq 0$. This function nests CARA (as r tends to 0), is not defined for $\alpha = 0$, but tends to CRRA as α tends to 0. Using methods such as those described above, without corrections for clustering, and using the Luce error specification, we obtain the following estimates:

```
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
r            |
       _cons |   .2684983   .0173942    15.44   0.000     .2344064    .3025902
-------------+----------------------------------------------------------------
alpha        |
       _cons |   .0284649    .002428    11.72   0.000     .0237061    .0332238
-------------+----------------------------------------------------------------
mu           |
       _cons |   .1336797   .0046297    28.87   0.000     .1246056    .1427538
------------------------------------------------------------------------------
```

These estimates literally replicate those reported by Holt and Laury [2002; p.1653] using *GAUSS* and "grouped" probabilities of saying yes to the safe lottery in each row of their Multiple Price List. Hence their econometric analysis assumes no heterogeneity across individuals, and ignores the fact that the multiple responses from the same individual are likely to be correlated. The implementation in *Stata* allows one to estimate the model at the level of individual choices, and hence evaluate the effects of demographics and statistical corrections for heterogeneity.[16]

The estimate of α is positive and statistically significant, indicating IRRA over the domain of prizes used in the estimation. Figure 6 illustrates the estimates of RRA for different prize levels: the display shows the point estimates of RRA and the 95% confidence interval. The vertical lines show the actual prizes used in the experimental tasks. If one corrects for clustering at the level of the individual subject there is virtually no change in the estimates, and slightly smaller standard errors. If one just uses the data from their experiments in which payoffs were scaled by 20 times the default, and drops the tasks using scaling values of 50 or 90, the results are virtually identical.[17] And one can include demographics to better characterize the heterogeneity one expects in risk attitudes (e.g., Harrison, Lau and Rutström [2007]).

Wilcox [2008a][2008b] notes the critical role of what he calls the "stochastic identifying restriction" in these estimates: the Luce error story. We do not know very much about these, and other, error specifications, other than the computational tractability of some of the chosen specifications. Wilcox [2008a] notes that the important finding of IRRA over this domain is an artefact of the Luce assumption. We can evaluate this quickly by substituting the Fechner error assumption (and probit specification linking the EU index to the probability of observed choices). The results are striking, and confirm this caution:

[16] No doubt one can do this in *GAUSS*, or the other popular packages for ML estimation of structural micro-econometric models, such as *Mathematica*.

[17] In fact, less than 10% of the data used scaling values of 50 or 90, so the scaling of 20 is effectively what drives these results, even if it is tempting to think of these data as spanning prizes as high as $346.50 when the vast majority of decisions actually had prizes up to $77.

```
             |               Robust
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
r            |
       _cons |   .6844524   .0488332    14.02   0.000      .588741    .7801637
-------------+----------------------------------------------------------------
alpha        |
       _cons |   .0446482   .0593774     0.75   0.452    -.0717293    .1610258
-------------+----------------------------------------------------------------
mu           |
       _cons |   .1717156   .0159628    10.76   0.000     .1404289    .2030022
------------------------------------------------------------------------------
```

Clearly the estimate of α is positive again, but no longer statistically significantly different from 0 (*p*-value 0.452). Figure 7 displays the implied estimates of RRA, for comparison to Figure 6.

F. Testing Non-Nested Models

An issue that comes up when estimating choice models is how to discriminate between them. The answer we prefer is "not to," in the sense that one should allow the data-generating process to admit more than one choice model and estimate a mixture model (Harrison and Rutström [2005]). But how does this response compare to more classical responses, using formal tests to discriminate between models?

One alternative, particularly popular in political science (Clarke [2001]) is to estimate Frankenstein-like "supermodels" that simply append one model on to the other, and then apply the usual F-tests or likelihood tests of parameter restrictions. Another alternative is to use one of several tests of non-nested model specification.

The first is due to Cox [1961][1962], and assumes that one of two models is the true model for all data. It then proposes two likelihood-ratio tests, based on this premiss, and leads to one of four inferences: both models accepted as true, one model accepted as true, the other model accepted as true, or both rejected as true. The ambiguity in the resulting inference has bothered many who might otherwise use the test.

The second test is due to Vuong [1989], and has proven relatively popular in textbook expositions but surprisingly rare in statistical packages. *Stata*, for example, only implements it in a very special case (the zero-inflated negative binomial model). One reason may be that in general non-linear ML settings it requires that one access the likelihoods for each observation and for each model, data which is always generated in the `lnf' variable within the ML evaluator but not retained afterwards. The coding of the Vuong test is illustrated below.

The third test is due to Clarke [2003], and is similar in motivation to the Vuong test but ends up using a non-parametric sign test to discriminate between the models. The Vuong and Clarke tests can be compared in terms of their asymptotic efficiency as tests, and it turns out that when the distribution of the log of the likelihood ratios[18] is normally distributed that the Vuong test is better. But if this distribution exhibits sharp peaks, in the sense that it is mesokurtic, then the Clarke test is better (see Clarke [2007]). Many of the likelihood ratios we deal with have the latter shape, as we will see using the EUT and PT example from above, so it is useful to be armed with both tests.

The real coding trick here is to be aware of a dark little secret about ML coding that *Stata* does not want you to know. Within the ML evaluator program you can refer directly to variables created prior to the program, and thus feed variables directly to the program or access variables after the program has run. Thus, if one gives the command

```
generate double ll=.
```

prior to invoking the `ml model` command, and then insert

```
replace ll=`lnf'
```

just after the variable `lnf' has been generated in the likelihood evaluator, you get to play with the variable ll after the estimation stage. So after estimating the original EUT model in section A, we would have

[18] Where "ratio" refers to the likelihood of data point i under model 1 divided by the likelihood of data point i under model 2.

```
* save ll for EUT model
generate p_eut=exp(ll)
qui summ ll
local ll_eut=r(sum)
replace ll=.
```

Then after estimating the simple PT model in section B we would add

```
* save ll for PT model
generate p_pt=exp(ll)
qui summ ll
local ll_pt=r(sum)
```

Then we are ready to just calculate the test statistics as follows:

```
* get the Vuong test statistic for non-nested hypotheses -- EUT and PT
local model1 "EUT"
local model2 "PT"
generate p_`model1'=p_eut
generate p_`model2'=p_pt

* calculate Vuong numerator
generate v_tmp1 = log(p_`model1'/p_`model2')
qui summ v_tmp1
local v_num=(1/sqrt(r(N)))*r(sum)

* calculate Vuong denominator
local v_mean = (r(sum)/r(N))^2
generate v_tmp2 = log(p_`model1'/p_`model2')^2-`v_mean'
qui summ v_tmp2
local v_den = sqrt(r(sum)/r(N))

* get Vuong statistic
local z = `v_num'/`v_den'
local p = 1-normal(`z')

* get Clarke sign test
generate d=ln(p_`model1')-ln(p_`model2')
generate dsign=0
replace  dsign=1 if d>=0
qui summ dsign
local b=r(sum)
local n=r(N)
local half=r(N)/2
local p_ = Binomial(`n',`b',0.5)

* report results
di "Log-likelihood for `model1' is `ll_eut' and for `model2' is `ll_pt'"
di "Vuong statistic for test of `model' and `model2' is `z' and favors `model1' over `model2' if positive (p-value = `p'
of `model1'" as error " not " as result "being the better model)"
di "Clarke statistic for test of `model1' and `model2' is `b' and favors `model1' over `model2' if greater than `half'
(p-value = `p_' of `model1'" as error " not " as result "being the better model)"
```

The results of this code are as follows, using the same data we used earlier:

```
Log-likelihood for EUT is -9512.818929490921 and for PT is -9199.913993418244
Vuong statistic for test of  and PT is -11.37022328151276 and favors EUT over PT if positive (p-value = 1 of EUT not
being the better model)
Clarke statistic for test of EUT and PT is 6765 and favors EUT over PT if greater than 7520 (p-value = .9999999999999999
of EUT not being the better model)
```

So PT wins using these metrics. Although the two tests come to the same conclusion, they need not, and it is useful to take a look at the distribution of the logs of likelihood ratios to see which test is better. Figure 8 displays kernel densities of these distributions. We see that the empirical distribution is indeed on the sharp-peaked side compared to a Normal distribution fit to the same data.

This comparison is particularly interesting methodologically, since the mixture model estimated by Harrison and Rutström [2005], and discussed in section G below, tells us that the data is roughly characterized by a 50/50 mixture of EUT and PT, even though these tests seem to say "All the Way with PT." The reconciliation comes from thinking about what these tests are interested in: the Vuong and Clarke model specification tests presume that one or the other model are the true model, and compare likelihoods for every data point assuming one model or the other model are supposed to explain them. The mixture model admits that distinct models might be better for different observations. In fact, the mixture model logic would seem to revive the rationale for the Cox test of non-nested models, although it does not follow that the Cox test would correctly detect the true mixture. This is clearly a fertile area for some Monte Carlo tests. But first we should consider how to estimate mixture models, given their role in this discussion of non-nested hypothesis tests.

G. Mixture Models

Mixture models combine many of the tools we have developed. We illustrate with a replication of the initial model from Harrison and Rutström [2005], which assumes an EUT model with a CRRA utility function defined over the final outcomes that the subject faced (including the initial endowments to ensure no post-session losses), the simple SPT specification of PT, no stochastic errors, and the logit cumulative density function to link the latent index to the probability of the observed choice. Clearly there are a host of parametric assumptions here, and one should by now expect results to differ with alternative specifications, but the general methodological point about mixture models is most transparent in this context.[19]

We first list the program in it's entirety, and then explain each section in terms of the economics and econometrics of the problem.

```
* define mixture model of Original Recipe EUT and Spicy PT -- no error terms
program define MLmixed0

    args lnf alpha beta lambda gamma r kappa
    tempvar prob0l prob1l prob2l prob3l prob0r prob1r prob2r prob3r m0 m1 m2 m3 y0 y1 y2 y3 euL euR euDiff euRatio
    tempvar tmp lnf_eut lnf_pt p1 p2 f1 f2

    quietly {

        * construct likelihood for EUT
        generate double `prob0l' = $ML_y2
        generate double `prob1l' = $ML_y3
        generate double `prob2l' = $ML_y4
        generate double `prob3l' = $ML_y5

        generate double `prob0r' = $ML_y6
        generate double `prob1r' = $ML_y7
        generate double `prob2r' = $ML_y8
        generate double `prob3r' = $ML_y9

        generate double `m0' = $ML_y10
        generate double `m1' = $ML_y11
        generate double `m2' = $ML_y12
        generate double `m3' = $ML_y13
```

[19] It is worth noting that the use of SPT instead of CPT makes no significant difference to the results, and template code is included to demonstrate this. The CPT code is so long that it distracts from the basic programming point.

```
generate double `y0' = ($ML_y14+`m0')^`r'
generate double `y1' = ($ML_y14+`m1')^`r'
generate double `y2' = ($ML_y14+`m2')^`r'
generate double `y3' = ($ML_y14+`m3')^`r'

generate double `euL' = (`prob0l'*`y0')+(`prob1l'*`y1')+(`prob2l'*`y2')+(`prob3l'*`y3')
generate double `euR' = (`prob0r'*`y0')+(`prob1r'*`y1')+(`prob2r'*`y2')+(`prob3r'*`y3')
generate double `euDiff' = `euR' - `euL'

generate double `lnf_eut' = .
replace `lnf_eut' = ln($cdf( `euDiff')) if $ML_y1==1
replace `lnf_eut' = ln($cdf(-`euDiff')) if $ML_y1==0

* construct likelihood for PT -- SPT probability weighting with venerable S-function
generate double `tmp'    = (($ML_y2^`gamma')+($ML_y3^`gamma')+($ML_y4^`gamma')+($ML_y5^`gamma'))^(1/`gamma')
replace `prob0l' = ($ML_y2^`gamma')/`tmp'
replace `prob1l' = ($ML_y3^`gamma')/`tmp'
replace `prob2l' = ($ML_y4^`gamma')/`tmp'
replace `prob3l' = ($ML_y5^`gamma')/`tmp'

replace `tmp'            = (($ML_y6^`gamma')+($ML_y7^`gamma')+($ML_y8^`gamma')+($ML_y9^`gamma'))^(1/`gamma')
replace `prob0r' = ($ML_y6^`gamma')/`tmp'
replace `prob1r' = ($ML_y7^`gamma')/`tmp'
replace `prob2r' = ($ML_y8^`gamma')/`tmp'
replace `prob3r' = ($ML_y9^`gamma')/`tmp'

* funky-chicken value function
replace `y0' =           ( $ML_y10)^(`alpha') if `m0'>=0
replace `y0' = -`lambda'*(-$ML_y10)^(`beta')  if `m0'<0

replace `y1' =           ( $ML_y11)^(`alpha') if `m1'>=0
replace `y1' = -`lambda'*(-$ML_y11)^(`beta')  if `m1'<0

replace `y2' =           ( $ML_y12)^(`alpha') if `m2'>=0
replace `y2' = -`lambda'*(-$ML_y12)^(`beta')  if `m2'<0

replace `y3' =           ( $ML_y13)^(`alpha') if `m3'>=0
replace `y3' = -`lambda'*(-$ML_y13)^(`beta')  if `m3'<0

* they call this prospective utility, but heck, it looks like funky-chicken EU to me...
replace `euL' = (`prob0l'*`y0')+(`prob1l'*`y1')+(`prob2l'*`y2')+(`prob3l'*`y3')
replace `euR' = (`prob0r'*`y0')+(`prob1r'*`y1')+(`prob2r'*`y2')+(`prob3r'*`y3')

* eu difference
replace `euDiff' = `euR' - `euL'

* logit index function
generate double `lnf_pt' = .
replace `lnf_pt' = ln($cdf( `euDiff')) if $ML_y1==1
replace `lnf_pt' = ln($cdf(-`euDiff')) if $ML_y1==0
```

```
        * redundantly transform back from log contributions, to see the mixture logic transparently
        generate double `f1' = exp(`lnf_eut')
        generate double `f2' = exp(`lnf_pt')

        * transform to the mixture probabilities
        generate double `p1' =           1/(1+exp(`kappa'))
        generate double `p2' = exp(`kappa')/(1+exp(`kappa'))

        * take the log of the two contributions
        replace `lnf' = ln(`p1'*`f1' + `p2'*`f2')

        * save variables, and display for debugging (change “qui” to “noi”)
        qui summ `lnf' `lnf_eut' `lnf_pt' `p1' `p2'
        replace ll=`lnf'
        replace ll_eut=`lnf_eut'
        replace ll_pt=`lnf_pt'
        replace p_eut=`p1'
        replace p_pt=`p2'
        qui summ ll ll_eut ll_pt p_eut p_pt

    }

end
```

The general idea is to first be able to write a likelihood function for model A, then one for model B, much as we have done in earlier sections, tack them together and define a weighted likelihood that allows them both to explain the data. The code for the EUT and PT models is familiar from earlier examples, and indeed one should debug those initially before embarking on a mixture specification, unless you enjoy bungie-jumping without testing the cord first. The mixture probabilities are nicely constrained to be between 0 and 1, using the log-odds transformation introduced earlier.

The real novelty here is the definition of the grand likelihood contribution:

```
        replace `lnf' = ln(`p1'*`f1' + `p2'*`f2')
```

We literally just take a weighted average of the likelihood contribution of each model, where the weights are the probabilities of each model. Note that the models do not need to be nested,[20] and that one can easily see how to extend this logic to more than two models.[21]

Apart from this step, there is nothing particularly novel about estimating mixture models. They can be more numerically "fragile" than non-mixture models, but there are good economic and econometric reasons for this. For example, if you specify two models that are very similar in terms of the likelihood of each observation (e.g., a PT model assuming one flexible probability weighting function and another PT model assuming another flexible probability weighting function), the grand likelihood is going to be flat, since it can assign wildly different weights to the mixing probability and still arrive at the same grand likelihood value. But in this setting you should be asking familiar "power" questions if you view the available sample as an operationally meaningful test of these specifications against each other. Another numerical problem will arise if one of the models is just horrible: if the mixture model wants to assign 0 weight to it, then convergence problems will arise for the same reasons discussed when we first mentioned the log-odds transform (the parameter actually being estimated by *Stata*, in this case κ, can pick any one of many finite values, tending towards $\mp\infty$, and obtain the same grand likelihood). Again, the numerical problems are just a reflection of poor economic modeling in the first place.

Estimating the above model on the data used by Harrison and Rutström [2005], which is a subset[22] of the data considered here, we obtain a replication of their estimates:

```
. ml model lf MLmixed0 (alpha: Choices P0left P1left P2left P3left P0right P1right P2right P3right prize0 prize1 prize2
prize3 stake = ) (beta: ) (lambda: ) (gamma: ) (r: ) (kappa: ), cluster(id) technique(dfp) maximize init(.73 .73 1.38
.94 .84 0, copy)

                                                  Number of obs   =       9311
                                                  Wald chi2(0)    =          .
Log pseudolikelihood =  -5908.121                 Prob > chi2     =          .
```

[20] An advanced conceptual exercise: if the models can be nested, is there ever any point in estimating a mixture model, instead of estimating the general model and just testing the parametric restriction? Hint: yes.

[21] Rabe-Hesketh and Everitt [2004; p.252] note one of the standard transformation tricks needed here to ensure that all probabilities sum to 1.

[22] The main difference is that more data has been collected since 2005, and our objective here is to replicate the estimates they report. Their data includes tasks that confront subjects with gain frame, loss frame, or mixed frame lottery choices.

```
                                   (Std. Err. adjusted for 158 clusters in id)
------------------------------------------------------------------------------
             |               Robust
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
alpha        |
       _cons |   .6137602   .0572608    10.72   0.000      .501531    .7259893
-------------+----------------------------------------------------------------
beta         |
       _cons |   .3121583   .1317669     2.37   0.018     .0538999    .5704167
-------------+----------------------------------------------------------------
lambda       |
       _cons |   5.781483   1.611687     3.59   0.000     2.622634    8.940331
-------------+----------------------------------------------------------------
gamma        |
       _cons |   .6805912   .0474594    14.34   0.000     .5875725      .77361
-------------+----------------------------------------------------------------
r            |
       _cons |   .8459939   .0439017    19.27   0.000     .7599481    .9320397
-------------+----------------------------------------------------------------
kappa        |
       _cons |  -.2000094   .2911551    -0.69   0.492     -.770663    .3706442
------------------------------------------------------------------------------

. * evaluate mixture probabilities and test probEUT=0 and probPT=0
. nlcom (pEUT: 1/(1+exp([kappa]_b[_cons])))  (pPT: exp([kappa]_b[_cons])/(1+exp([kappa]_b[_cons])))

        pEUT:  1/(1+exp([kappa]_b[_cons]))
         pPT:  exp([kappa]_b[_cons])/(1+exp([kappa]_b[_cons]))

------------------------------------------------------------------------------
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
        pEUT |   .5498363   .0720657     7.63   0.000     .4085902    .6910824
         pPT |   .4501637   .0720657     6.25   0.000     .3089176    .5914098
------------------------------------------------------------------------------

. * test probEUT = probPT
. testnl 1/(1+exp([kappa]_b[_cons])) - exp([kappa]_b[_cons])/(1+exp([kappa]_b[_cons])) = 0

  (1)  1/(1+exp([kappa]_b[_cons])) - exp([kappa]_b[_cons])/(1+exp([kappa]_b[_cons])) = 0

               chi2(1) =        0.48
           Prob > chi2 =        0.4892
```

The main result is that the probability of the EUT model explaining choices is 0.55 and the probability of the PT model is 1-0.55 = 0.45.

We cannot reject the hypothesis that each model contributes equally. Since we can reject the hypothesis that the mixing probability is 0, it is not the case that this result should be viewed as "the data cannot tell which model explains the data." Instead, this model tells us that there are some choices being made with EUT logic, and some being made with PT logic. The individual parameter estimates tend to be consistent with qualitative priors from the literature, with $\lambda>1$ (in fact, $\lambda\gg1$), $\gamma<1$, $\alpha<1$ and $\beta<1$. Harrison and Rutström [2005] discuss the interpretation of these results, and more detailed models including demographic characteristics.

H. Maximum Simulated Likelihood

A popular approach to modeling unobserved individual heterogeneity involves the estimation of random *coefficients* using maximum simulated likelihood (MSL). The idea is very simple and elegant, even if the numerical implementation can become tedious. To illustrate the basic idea, go back to our first parametric EUT model in which there is one core parameter r representing CRRA risk attitudes. We estimated r as a deterministic coefficient, but allowed for sampling error. Another view, quite plausible, is that there is heterogeneity in preferences such that r is better characterized as a distribution. For simplicity and tradition, assume that this distribution is Normal[23] and that r is distributed with mean r_mean and standard deviation r_sd. Then we can estimate r_mean and r_sd by generating R simulations of values of r we will call sim_r, evaluating the likelihood of the data conditional on sim_r, and then reporting as the overall likelihood the average likelihood of these R simulations. In effect, our previous estimation assumed r_sd=0, and gave us a great initial value for r_mean.[24]

One of the constraints of this approach is that one then has to do R function evaluations every time that the ML evaluator is called, and since we are lazy and require the ML evaluator to generate numerical derivatives for us, this can add up to a significant computational

[23] For computational tractability the usual assumptions have been uniform, triangular or normal. But one could easily extend this approach to more exotic and flexible distributions, such as beta, with little coding work and substantial numerical work. But since humans do the former and computers the latter, that is fine. Such extensions may soon uncover dramatically interesting forms of heterogeneity.

[24] One can further limit things numerically by assuming that this random effect on r has mean 0 and standard deviation r_sd, and save time by constraining r_mean to be equal to the previous estimate of r assuming r_sd=0.

burden if R gets large.[25] The most important numerical insight here is to use "intelligent" random draws of random deviates so that one gets good coverage over the parameter space, and the implied integrals are good approximations for smaller R. The most popular of the intelligent ways to simulate random deviates involves Halton sequences of uniform deviates (e.g., Train [2003; ch.9]). To implement this in *Stata* we generate a Halton sequence of uniform deviates before calling the ML program, and then just refer to that sequence directly within the program.[26]

Our example builds on the non-parametric EUT model in which we estimated a utility value in the gain domain when prizes were $0, $5, $10 or $15, and $0 was normalized to 0 and $15 to 1. Two core parameters are estimated, the utility of $5 and the utility of $10. Thus we generate Halton draws for two random coefficients, using the following *Stata* code:

```
* number of random draws (twice the number in the next row, due to antithetics)
global draw_raw=50
global draw=$draw_raw*2
mdraws, draws($draw_raw) neq(2) prefix(h) replace burn(10) antithetics shuffle
```

The documentation of the command `mdraws` is provided by Capellari and Jenkins [2006], and the details about primes, seeds, antithetic and such are well explained there. The end result here is that we have some variables h1_* and h2_* that are uniform deviates in two dimensions. Here is what some of these deviates look like for the first 5 of the 100 Halton draws, and then the last 3 Halton draws:

```
   Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        h1_1 |      6191     .496128     .286413    .000782   .9999237
        h2_1 |      6191    .4988762    .2884793   .0001298   .9998927
        h1_2 |      6191    .4962779    .2905683   .0002022   .9999619
        h2_2 |      6191    .5015493    .2929573   .0000922   .9999944
```

[25] One can remain lazy and still be efficient at it, however. Gates [2006] provides methods in *Stata* that can calculate derivatives for simulated multivariate normal distributions. These methods require that one code up small functions in the matrix language of *Stata*, *Mata*, and that calls for a different mind set in terms of syntax.

[26] This step utilizes the dark, dirty secret that *Stata* does not document, that you can read variables directly in a program. These deviates could instead be passed to the program in the usual manner as data and in the form of the $ML_y* variables, but then one would have to hard-wire R and enter a tedious list of numbers (actually, K×R of them, where K is the dimensionality of the set of parameters one is estimating with random simulations).

```
        h1_3 |      6191     .499091    .2916584     .000061   .9999657
        h2_3 |      6191    .5003656    .2880811    .0003443   .9999737
        h1_4 |      6191     .502179    .2869911    .0004883   .9995651
        h2_4 |      6191    .4946281    .2875912    .0001694   .9998551
        h1_5 |      6191    .5047536    .2847204    .0001183   .9999504
        h2_5 |      6191    .5025803    .2856704    .0000583   .9999567

       h1_98 |      6191     .503584    .2881417     .000042   .9998779
       h2_98 |      6191    .4963197    .2894765     .000175   .9998796
       h1_99 |      6191    .5007305    .2888644    .0000725   .9999504
       h2_99 |      6191    .4974064    .2899968    .0002013   .9996613
      h1_100 |      6191    .5023769    .2857515    7.63e-06   .9999828
      h2_100 |      6191    .5012885    .2894015    .0000113    .999332
```

Thus in our example R=100.

The ML program is a simple extension of the program developed earlier.[27] We specify a mean and a standard deviation for each coefficient that is now estimated as a random coefficient. For u5 we now estimate u5_mean and LNu5_sd, where the latter is in a logairthmic form so that we estimate a positive standard deviation (after transforming the raw parameter). Similarly for u10, we have a mean and a standard deviation that must be estimated. In addition, we allow for the possibility that these two random coefficients are correlated, and estimate that correlation. For the correlation we use a common transformation to ensure that the latent correlation is between ∓1. We use Fechner errors, but do not let the error term be a random coefficient. Here is the ML program:

```
* define Original Recipe EUT with Fechner errors: non-parametric with random coefficients
program define ML_eut_np_randy

    args lnf u5_mean_ LNu5_sd u10_mean_ LNu10_sd atrho12 LNmu

    tempvar u5_mean u5_sd u10_mean u10_sd
    tempvar choose prob0l prob1l prob2l prob3l prob0r prob1r prob2r prob3r euL euR euDiff u0 u15 mu
    tempvar sim_u5 sim_u10 sim_lnf simmed_prob simmed_lnf u5_var u10_var cov12
```

[27] Multinomial probit and logit models using random coefficients are illustrated by Capellari and Jenkins [2006], Gates [2006] and Haan and Uhlendorff [2006] for *Stata*. They achieve computational efficiency by using specialized routines for these model specifications, substituting for the "long-hand" loops that our program codes up explicitly. Those efficiencies can be substantial in MSL, but we need the ability to embed the code for the evaluation of the utility function *within* that loop. Hole [2007] documents an implementation of the versatile "mixed logit" model in *Stata* using MSL methods.

```
quietly {

    * define parameters
    generate double `mu'       =  exp(`LNmu')
    generate double `u5_mean'  =  1/(1+exp(`u5_mean_'))
    generate double `u5_sd'    =  exp(`LNu5_sd')
    generate double `u10_mean' =  1/(1+exp(`u10_mean_'))
    generate double `u10_sd'   =  exp(`LNu10_sd')

    scalar `u5_var'  = (exp(`LNu5_sd'))^2
    scalar `u10_var' = (exp(`LNu10_sd'))^2

    scalar `cov12' = [exp(2*`atrho12')-1]/[exp(2*`atrho12')+1]*`u5_sd'*`u10_sd'

    * set up data
    generate int    `choose' = $ML_y1
    generate double `prob0l' = $ML_y2
    generate double `prob1l' = $ML_y3
    generate double `prob2l' = $ML_y4
    generate double `prob3l' = $ML_y5

    generate double `prob0r' = $ML_y6
    generate double `prob1r' = $ML_y7
    generate double `prob2r' = $ML_y8
    generate double `prob3r' = $ML_y9

    generate double `u0'  = 0
    generate double `u15' = 1

    * define covariance matrix (all elements must be scalars, not variables)
    matrix p = (`u5_var', `cov12' \ `cov12', `u10_var')

    * get the Cholesky factors calculated
    capture matrix c = cholesky(p)
    local c11 = c[1,1]
    local c21 = c[2,1]
    local c22 = c[2,2]
    $noise display "Cholesky factors: c11 is `c11' c21 is `c21' and c22 is `c22'"

    * loop over likelihood evaluator with $draw random draws and accumulate simulated likelihoods
    generate double `simmed_lnf' = 0
    forvalues d=1/$draw {

        * get the random deviate and the probability
        generate double `sim_u5'  =`u5_mean' +`c11'*invnormal(h1_`d')
        generate double `sim_u10' =`u10_mean'+`c21'*invnormal(h1_`d')+`c22'*invnormal(h2_`d')

        * construct EU
        generate double `euL' = (`prob0l'*`u0')+(`prob1l'*`sim_u5')+(`prob2l'*`sim_u10')+(`prob3l'*`u15')
        generate double `euR' = (`prob0r'*`u0')+(`prob1r'*`sim_u5')+(`prob2r'*`sim_u10')+(`prob3r'*`u15')
```

```
            * construct EU Fechner index
            generate double `euDiff' = (`euR' - `euL')/`mu'

            * get log-likelihood
            generate double `sim_lnf' = 0
            replace `sim_lnf' = ln($cdf( `euDiff')) if `choose'==1
            replace `sim_lnf' = ln($cdf(-`euDiff')) if `choose'==0

            * accumlate log-likelihood over random draws
            replace `simmed_lnf' = `simmed_lnf' + (`sim_lnf'/$draw)

            * clean up
            drop `sim_u5' `sim_u10' `euL' `euR' `euDiff' `sim_lnf'

        }

        replace `lnf' = `simmed_lnf'
        replace  ll   = `lnf'

    }
end
```

The new parts that are unique to MSL are shown in bold.

The first new part just undertakes transformations of the variables, using familiar methods. The covariance of the random coefficients is inferred from the assumed value of the correlation and standard deviations. The formula looks cryptic, but is standard from this literature. The second new part constructs the Covariance matrix and takes it's Choleski inverse, to obtain the terms that are used to construct the multivariate Normal deviate from a series of univariate uniform deviates (the Halton draws we generated earlier). The Choleski inverse is just a generalized inverse: if the off-diagonal covariance term in this 2×2 matrix example was zero, the Choleski factors c11 and c22 would just be the standard deviations of the estimates of u5 and u10. Indeed, in the general 2×2 case in which there is a non-zero correlation between the two random coefficients, we can replace the code

```
        capture matrix c = cholesky(p)
        local c11 = c[1,1]
        local c21 = c[2,1]
        local c22 = c[2,2]
```

with the verbose, but clearer, code

```
        generate double `c11' = sqrt(`u5_var')
        generate double `c21' = `cov12'/`c11'
        generate double `c22' = sqrt(`u10_var’ - (`c21'^2))
```

This step is explained in Train [2003; p. 211ff.], and again is standard to this literature.

The key step to see the MSL logic is the large loop over the Halton draws. This is a loop over the R=100 draws we stored in the global variable $draws, prior to invoking the program. So this loop is executed 100 times. Each time it is executed, we construct a random bivariate Normal deviate for u5 and u10, called u5_sim and u10_sim. These are “built up” from the uniform Halton deviates, and the Choleski factors. Once we have these we simply construct the likelihood using the same code as before, but substituting u5_sim for u5 and u10_sim for u10. The likelihood is called sim_lnf for this execution of the loop, and then these values accumulated in simmed_lnf. We divide each increment to simmed_lnf by the probability of that likelihood, which is 1/$draws = 1/100, by construction. Adding this division by 1/$draws within the loop simply insures against numeric overflow as one increases the number of Halton draws. When we close the loop the variable simmed_lnf then holds the average likelihood over the 100 draws, and we simply assign that to the variable lnf that the ML program is looking for. All that the ML program cares about is the set of parameter values it is evaluating and the “bottom line” likelihood given those values. It does not care about the fact that you did 100 loops to evaluate that likelihood. In effect, the ML program does not “see” the fact that we built up the value for lnf by taking an average of 100 random draws, although that does involve some serious computational burden of course. We have to drop the variables constructed during the 100 draws, since they get re-generated in each draw.

The estimates are as follows:

```
. ml model lf ML_eut_np_randy (u5_mean_: Choices P0left P1left P2left P3left P0right P1right P2right P3right prize0
prize1 prize2 prize3 stake = ) (LNu5_sd: ) (u10_mean_: ) (LNu10_sd: ) (atrho12: ) (LNmu: ), cluster(id) technique(bfgs)
maximize difficult init(0 -2.5 -1 -2.5 0 -1, copy) search(norescale)
```

```
                                                  Number of obs   =       6191
                                                  Wald chi2(0)    =          .
Log pseudolikelihood = -4099.9351                 Prob > chi2     =          .

                                   (Std. Err. adjusted for 120 clusters in id)
------------------------------------------------------------------------------
             |               Robust
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
u5_mean_     |
       _cons |  -.4728532   .0941395    -5.02   0.000    -.6573632   -.2883431
-------------+----------------------------------------------------------------
LNu5_sd      |
       _cons |  -4.111476   .0495288   -83.01   0.000     -4.20855   -4.014401
-------------+----------------------------------------------------------------
u10_mean_    |
       _cons |  -1.028788   .0620079   -16.59   0.000    -1.150322   -.9072551
-------------+----------------------------------------------------------------
LNu10_sd     |
       _cons |  -4.598016   .0513013   -89.63   0.000    -4.698564   -4.497467
-------------+----------------------------------------------------------------
atrho12      |
       _cons |   .7995456    .036251    22.06   0.000     .7284949    .8705962
-------------+----------------------------------------------------------------
LNmu         |
       _cons |  -2.117776          .        .       .            .           .
------------------------------------------------------------------------------

. nlcom (u5_mean: 1/(1+exp([u5_mean_]_b[_cons]))) (u5_sd: exp([LNu5_sd]_b[_cons])) (u10_mean:
1/(1+exp([u10_mean_]_b[_cons]))) (u10_sd: exp([LNu10_sd]_b[_cons])) (corr: tanh([atrho12]_b[_cons])) (mu:
exp([LNmu]_b[_cons]))

     u5_mean:  1/(1+exp([u5_mean_]_b[_cons]))
       u5_sd:  exp([LNu5_sd]_b[_cons])
    u10_mean:  1/(1+exp([u10_mean_]_b[_cons]))
      u10_sd:  exp([LNu10_sd]_b[_cons])
        corr:  tanh([atrho12]_b[_cons])
          mu:  exp([LNmu]_b[_cons])

------------------------------------------------------------------------------
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     u5_mean |   .6160588   .0222668    27.67   0.000     .5724166    .6597011
       u5_sd |   .0163836   .0008115    20.19   0.000     .0147931     .017974
    u10_mean |   .7366809   .0120284    61.25   0.000     .7131057    .7602562
      u10_sd |   .0100718   .0005167    19.49   0.000     .0090591    .0110845
        corr |   .6637826   .0202785    32.73   0.000     .6240374    .7035278
          mu |   .1202988          .        .       .            .           .
------------------------------------------------------------------------------
```

The results are different than when we estimated the same model assuming non-random coefficients. There we estimated u5 to be 0.55 and u10 to be 0.78, and here we estimate u5_mean to be 0.62 and u10_mean to be 0.74. Thus illustrates the importance of allowing the mean of the random coefficient to be estimated jointly with the standard deviation of the coefficient, as well as jointly with the distribution of the other coefficient. That is, we might have constrained u5_mean to 0.55 and u10_mean to 0.78, and estimated their standard deviations and correlation under that point restriction, but that would only be valid as an interim numerical step to find better starting values for the complete estimation reported here where u5_mean and u10_mean were free to vary.

It is an easy matter to extend the MSL estimation to employ more Halton draws, and this is commonly done once the code has been debugged with a handful of draws. It is slightly more tedious to extend the estimation to 3 or more random coefficients, but the logic is essentially the same. It is also possible to include MSL estimation after one has included the effects of observable characteristics, so that the random coefficient estimation is interpreted as accommodating *unobserved* heterogeneity after allowing observed heterogeneity.

I. Conclusions

The tools illustrated here can be used to estimate a wide range of latent, structural models of choice behavior. The primary methodological contribution is to encourage the systematic estimation of multiple parameters in a consistent theoretical structure. One hopes to see an end to the piecemeal estimation of individual parameters, common in the behavioral literature. The ability to write out an explicit likelihood also allows one to immediately consider a wide range of "process models" from psychology and related fields, which have generally not been subjected to maximum likelihood estimation. This should encourage a more systematic comparison of traditional and non-traditional models of choice under uncertainty.

One extensions to the estimation toolkit seem particularly valuable. This extension is to embed the estimation of core parameters into strategic settings, in which the likelihood is generated in a manner consistent with some equilibrium notion from game theory (e.g., Goeree, Holt and Palfrey [2003]). This topic has been an area of active research, and comparable tools using *Stata* to examine these issues are likely to be soon available.

Figure 1: Normal and Logistic CDF Functions

Dashed line is Normal, and Solid line is Logistic

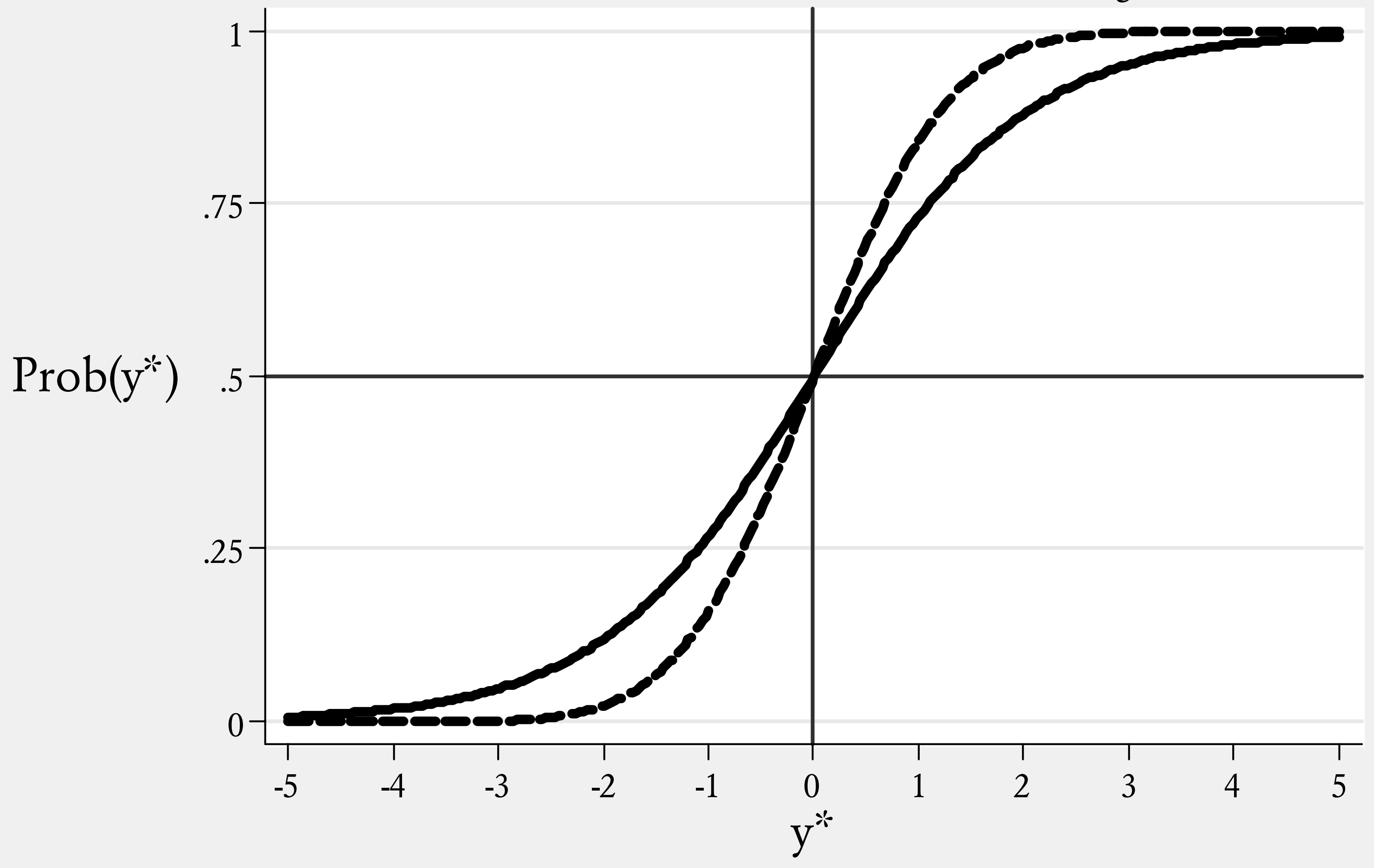


Figure 2: Decision Weights under RDU
Assuming TK Probability Weighting Function,
Equally Probable Lottery Outcomes,
and No Stochastic Errors

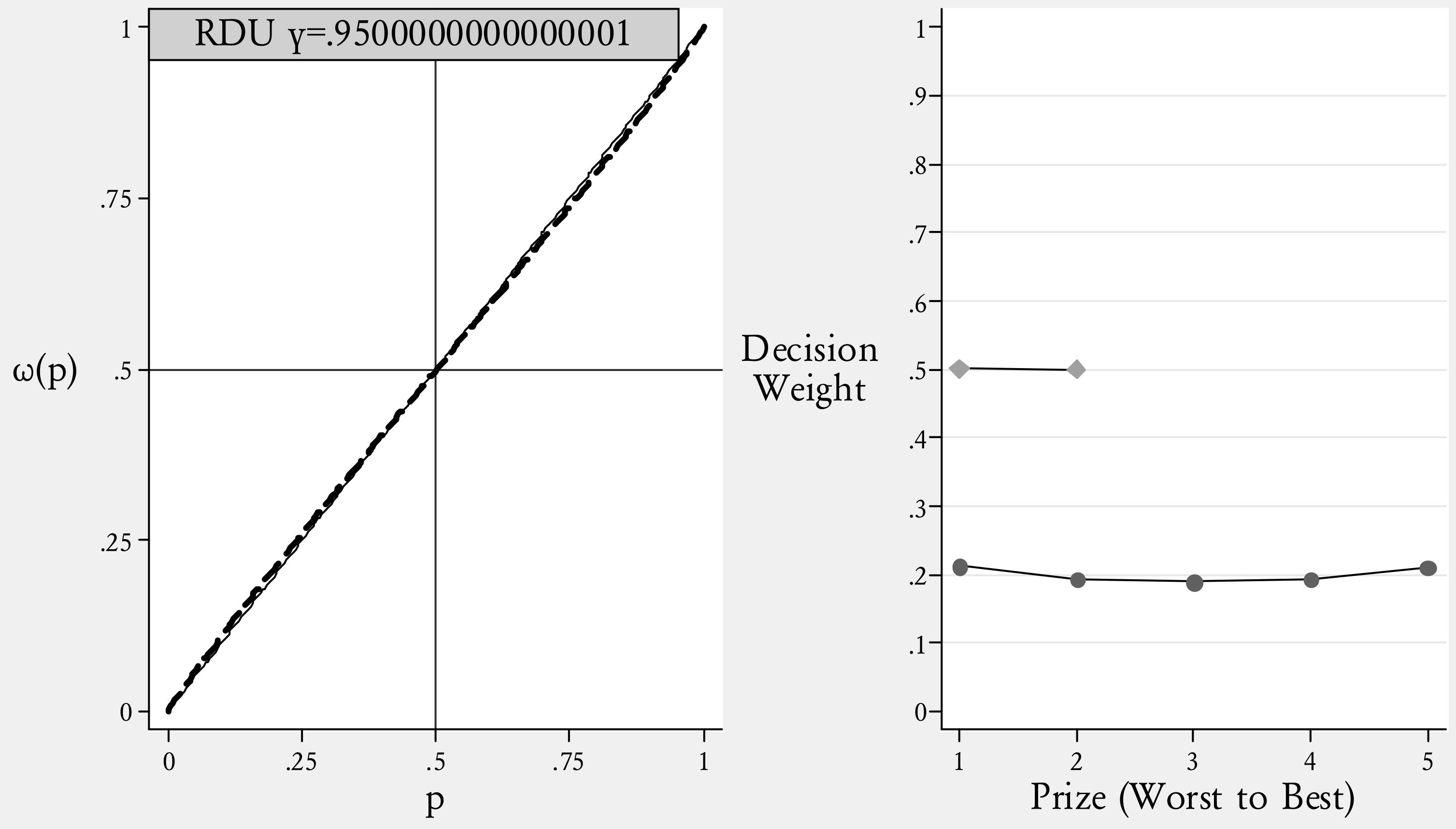


Figure 3: Decision Weights under RDU
Assuming TK Probability Weighting Function,
Equally Probable Lottery Outcomes,
and Fechner Stochastic Errors

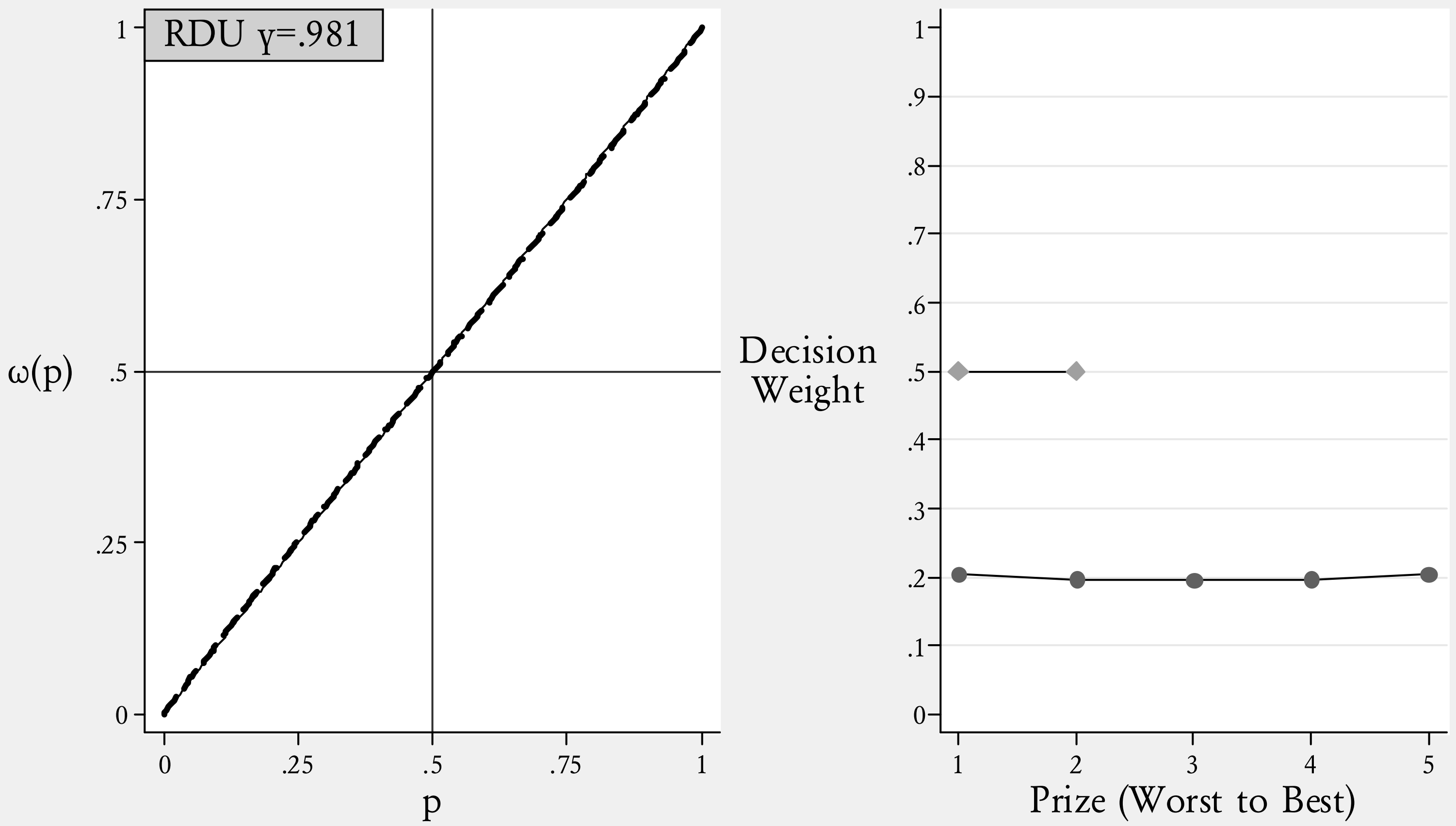


Figure 4: Decision Weights under RDEV Assuming TK Probability Weighting Function, Equally Probable Lottery Outcomes, and Fechner Stochastic Errors

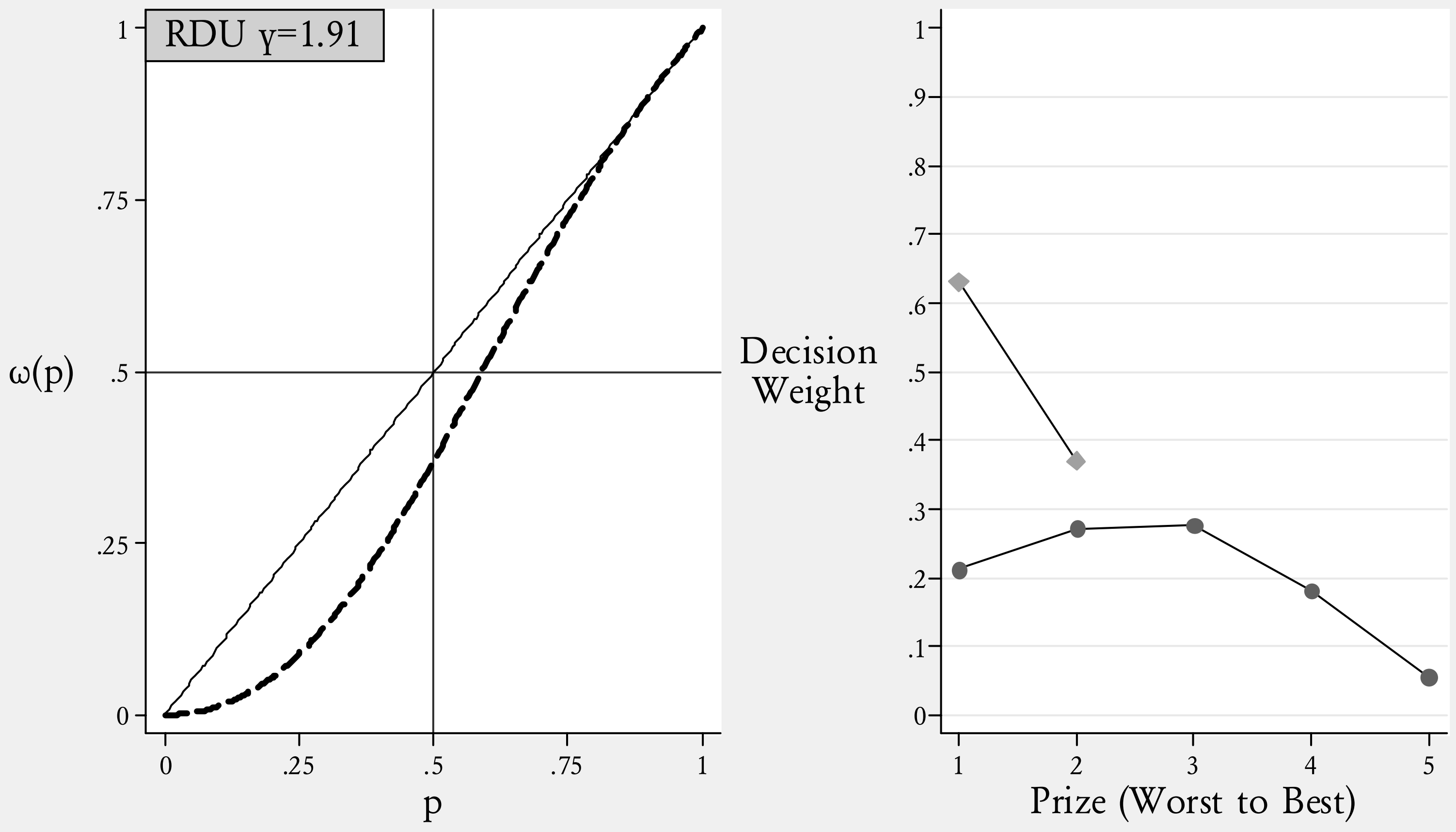


Figure 5: Maximum Likelihood Estimates of Utility of $5 and $10

Assuming EUT and normalized so that U($0)=0 and U($15)=1
Kernel density of predicted utility estimates for N=120 subjects

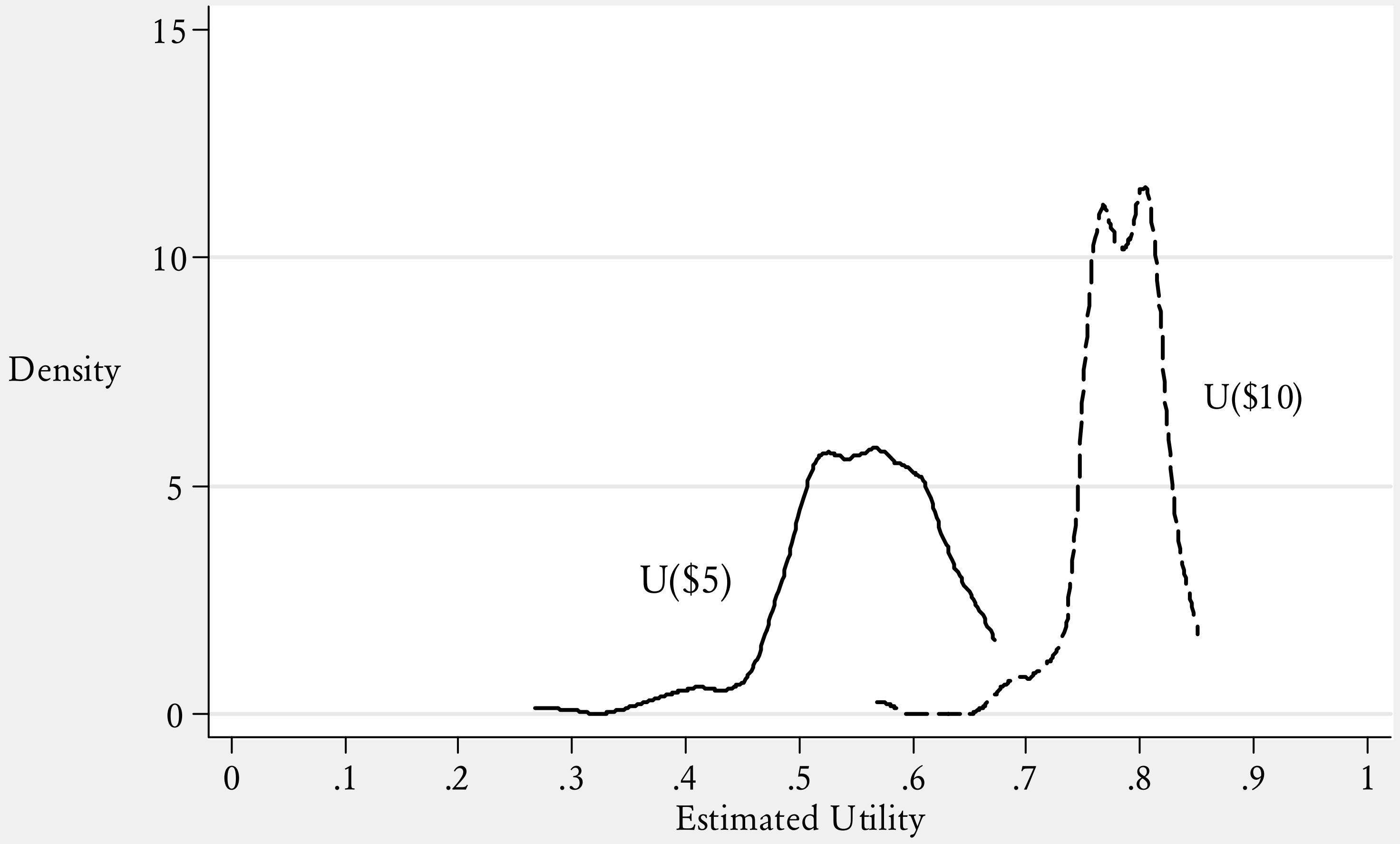


Figure 6: RRA in the Lab Assuming Luce Stochastic Error Story

Estimated from experimental data of Holt & Laury [2002]
Replicating estimation assumptions and results of Holt & Laury [2002]

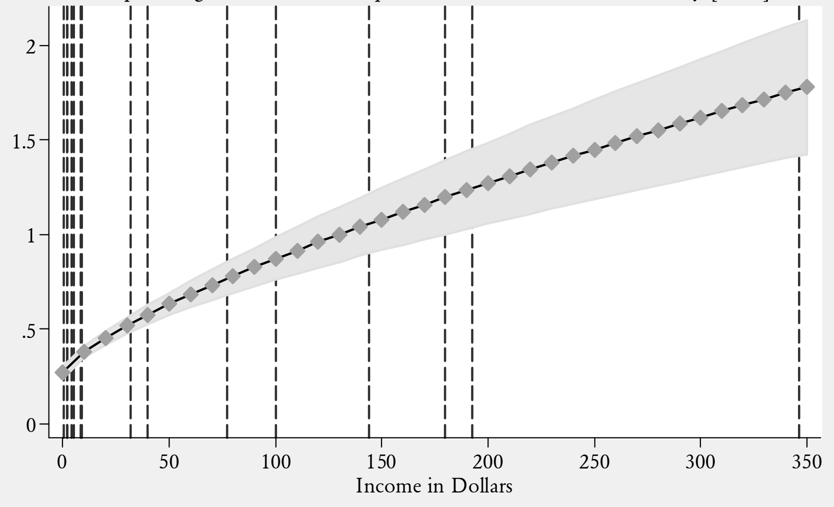


Figure 7: RRA in the Lab Assuming Fechner Stochastic Error Story

Estimated from experimental data of Holt & Laury [2002]
Assuming probit likelihood function and Fechner noise

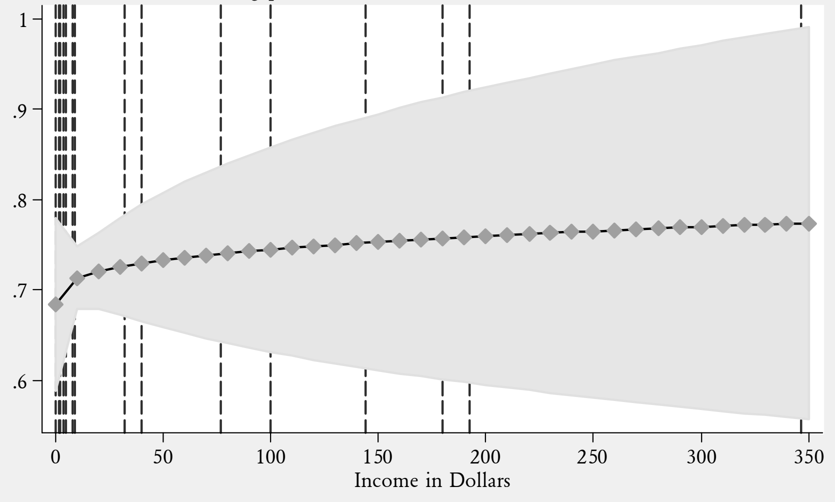
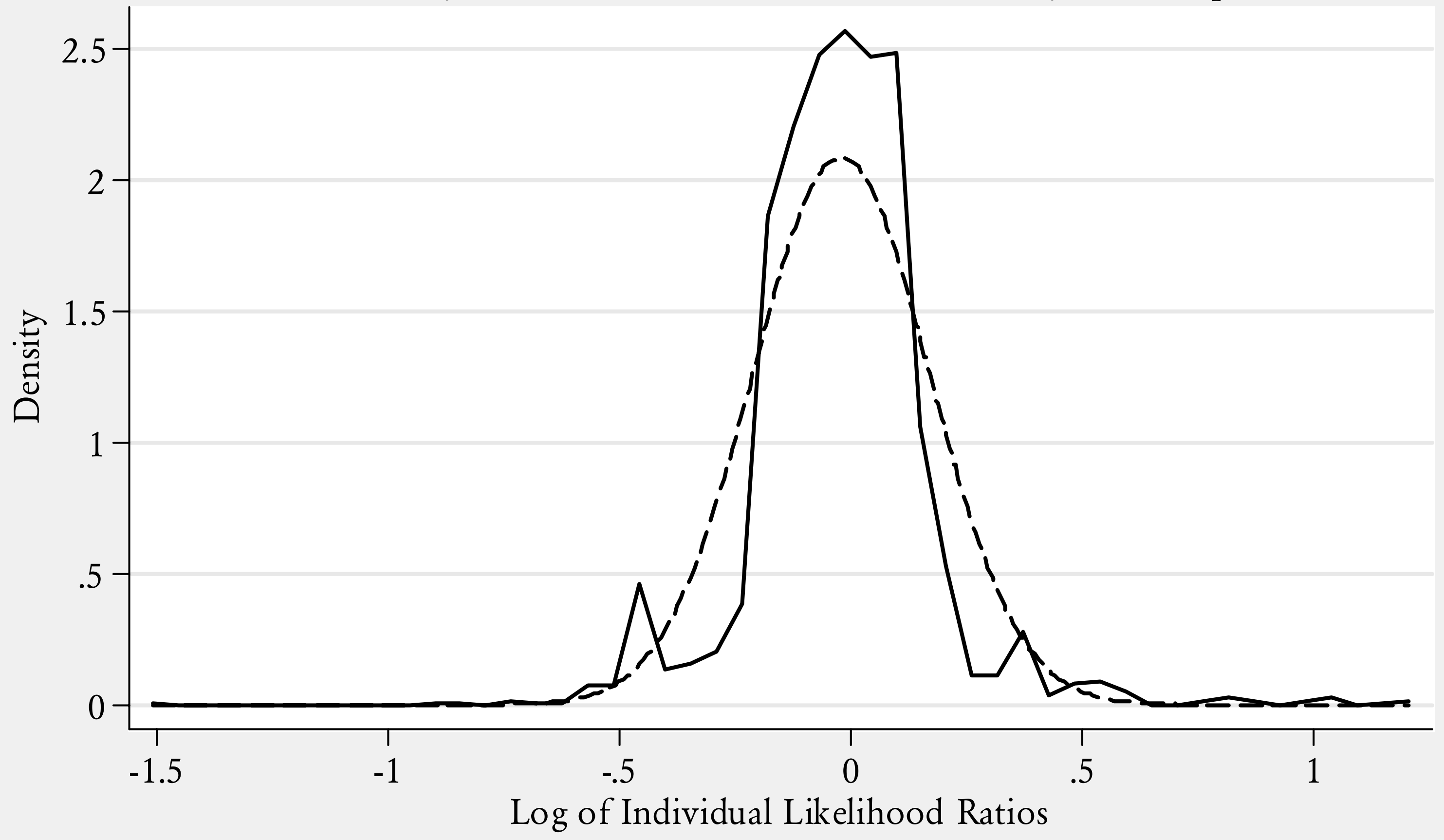


Figure 8: Distribution of Log Likelihood Ratios
for Non-Nested Tests of EUT Model and PT Model

Kernel density estimate of data and normal density for comparison



References

Allais, Maurice, "The Foundations of Positive Theory of Choice Involving Risk and a Criticism of the Postulates and Axioms of the American School," in M. Allais & O. Hagen (eds.), *Expected Utility Hypotheses and the Allais Paradox* (Dordrecht, the Netherlands: Reidel, 1979).

Andersen, Steffen; Harrison, Glenn W.; Lau, Morten Igel, and Rutström, E. Elisabet, "Eliciting Risk and Time Preferences," *Econometrica*, 76(3), May 2008, 583-618.

Atkinson, A.C., "A Method for Discriminating Between Models," *Journal of the Royal Statistical Society, Series B*, 32, 1970, 323-344.

Camerer, Colin F., and Ho, Teck-Hua, "Violations of the Betweeness Axiom and Nonlinearity in Probability," *Journal of Risk and Uncertainty*, 8, 1994, 167-196.

Capellari, Lorenzo, and Jenkins, Stephen P., "Calculation of Multivariate Normal Probabilities by Simulation, with Applications to Maximum Simulated Likelihood Estimation," *Stata Journal*, 6(2), 2006, 156-189.

Clarke, Kevin A., "Testing Nonnested Models of International Relations: Reevaluating Realism," *American Journal of Political Science*, 45(3), July 2001, 724-744.

Clarke, Kevin A., "Nonparametric Model Discrimination in International Relations," *Journal of Conflict Resolution*, 47(1), February 2003, 72-93.

Clarke, Kevin A., "A Simple Distribution-Free Test for Non-Nested Model Selection," *Political Analysis*, 15(3), 2007, 347-363.

Cox, David R., "Tests of Separate Families of Hypotheses," in E.G. Charatsis (ed.), *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability* (Berkeley: University of California Press, Volume 1, 1961, 105-123).

Cox, David R., "Further Results on Tests of Separate Families of Hypotheses," *Journal of the Royal Statistical Society, Series B*, 24, 1962, 406-424.

Fishburn, Peter C., "Additive Utilities with Incomplete Product Sets: Application to Priorities and Assignments," *Operations Research*, 15(3), May-June 1967, 537-542.

Gates, Richard, "A Mata Geweke-Hajivassiliou-Keane Multivariate Normal Simulator," *Stata Journal*, 6(2), 2006, 190-213.

Goeree, Jacob K., Holt, Charles A., and Palfrey, Thomas R., "Risk Averse Behavior in Generalized Matching Pennies Games," *Games and Economic Behavior*, 45, 2003, 9-113.

Gould, William; Pitblado, Jeffrey, and Sribney, William, *Maximum Likelihood Estimation With Stata* (College Station, TX: Stata Press, Third Edition, 2006).

Haan, Peter, and Uhlendorff, Arne, "Estimation of Multinomial Logit Models with Unobserved Heterogeneity Using Maximum Simulated Likelihood," *Stata Journal*, 6(2), 2006, 229-245.

Harrison, Glenn W., "Risk Attitudes in First-Price Auction Experiments: A Bayesian Analysis," *The*

Review of Economics & Statistics, 72, August 1990, 541-546.

Harrison, Glenn W.; Lau, Morten I., and Rutström, E. Elisabet, "Estimating Risk Attitudes in Denmark: A Field Experiment," *Scandinavian Journal of Economics*, 109(2), June 2007, 341-368.

Harrison, Glenn W.; Lau, Morten Igel; Rutström, E. Elisabet, and Sullivan, Melonie B., "Eliciting Risk and Time Preferences Using Field Experiments: Some Methodological Issues," in J. Carpenter, G.W. Harrison and J.A. List (eds.), *Field Experiments in Economics* (Greenwich, CT: JAI Press, Research in Experimental Economics, Volume 10, 2005).

Harrison, Glenn W.; Lau, Morten Igel, and Williams, Melonie B., "Estimating Individual Discount Rates for Denmark: A Field Experiment," *American Economic Review*, 92(5), December 2002, 1606-1617.

Harrison, Glenn W., and Rutström, E. Elisabet, "Expected Utility Theory *and* Prospect Theory: One Wedding and A Decent Funeral," *Working Paper 05-18*, Department of Economics, College of Business Administration, University of Central Florida, 2005; forthcoming, *Experimental Economics*.

Harrison, Glenn W., and Rutström, E. Elisabet, "Risk Aversion in the Laboratory," in J.C. Cox and G.W. Harrison (eds.), *Risk Aversion in Experiments* (Bingley, UK: Emerald, Research in Experimental Economics, Volume 12, 2008).

Hey, John D., and Orme, Chris, "Investigating Generalizations of Expected Utility Theory Using Experimental Data," *Econometrica*, 62(6), November 1994, 1291-1326.

Hole, Arne Risa, "Fitting Mixed Logit Models by Using Maximum Simulated Likelihood," *Stata Journal*, 7(3), 2007, 388-401.

Holt, Charles A., and Laury, Susan K., "Risk Aversion and Incentive Effects," *American Economic Review*, 92(5), December 2002, 1644-1655.

Kahneman, Daniel, and Tversky, Amos, "Prospect Theory: An Analysis of Decision under Risk," *Econometrica*, 1979, 47(2), 263-291.

Leamer, Edward E., *Specification Searches: Ad Hoc Inference with Nonexperimental Data* (New York: Wiley, 1978).

Lopes, Lola L., "Risk and Distributional Inequality," *Journal of Experimental Psychology: Human Perception and Performance*, 10(4), August 1984, 465-484.

Luce, R. Duncan, and Fishburn, Peter C., "Rank and Sign-Dependent Linear Utility Models for Finite First-Order Gambles," *Journal of Risk & Uncertainty*, 4, 1991, 29-59.

Oehlert, Gary W., "A Note on the Delta Method," *The American Statistician*, 46(1), February 1992, 27-29.

Quiggin, John, "A Theory of Anticipated Utility," *Journal of Economic Behavior & Organization*, 3(4), 1982, 323-343.

Rabe-Hesketh, Sophia, and Everitt, Brian, *A Handbook of Statistical Analyses Using Stata* (New York: Chapman & Hall/CRC, Third Edition, 2004).

Saha, Atanu, "Expo-Power Utility: A Flexible Form for Absolute and Relative Risk Aversion," *American Journal of Agricultural Economics*, 75(4), November 1993, 905-913.

StataCorp, *Stata Statistical Software: Release 9* (College Station, TX: StataCorp, 2005).

Starmer, Chris, and Sugden, Robert, "Violations of the Independence Axiom in Common Ratio Problems: An Experimental Test of Some Competing Hypotheses," *Annals of Operational Research*, 19, 1989, 79-102.

Stott, Henry P., "Cumulative Prospect Theory's Functional Menagerie," *Journal of Risk and Uncertainty*, 32, 2006, 101-130.

Train, Kenneth E., *Discrete Choice Methods with Simulation* (New York: Cambridge University Press, 2003).

Tversky, Amos, and Kahneman, Daniel, "Advances in Prospect Theory: Cumulative Representations of Uncertainty," *Journal of Risk & Uncertainty*, 5, 1992, 297-323; references to reprint in D. Kahneman and A. Tversky (eds.), *Choices, Values, and Frames* (New York: Cambridge University Press, 2000).

Vuong, Quang H., "Likelihood Ratio Tests for Model Selection and Non-Nested Hypotheses," *Econometrica*, 57(2), March 1989, 307-333.

Wilcox, Nathaniel T., "Stochastic Models for Binary Discrete Choice Under Risk: A Critical Primer and Econometric Comparison," in J. Cox and G.W. Harrison (eds.), *Risk Aversion in Experiments* (Bingley, UK: Emerald, Research in Experimental Economics, Volume 12, 2008a).

Wilcox, Nathaniel T., 'Stochastically More Risk Averse:' A Contextual Theory of Stochastic Discrete Choice Under Risk," *Journal of Econometrics*, 142, 2008b forthcoming.